Discrete Optimization

A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times

Manuel J. Pereira Lopes a, J.M. Valério de Carvalho b,*

a Dept. de Engenharia Mecânica, Instituto Superior de Engenharia do Porto, Portugal
b Dept. de Produção e Sistemas, Universidade do Minho, Portugal

Received 30 August 2004; accepted 9 November 2005

Abstract

We consider the problem of scheduling \( n \) independent jobs on \( m \) unrelated parallel machines with sequence-dependent setup times and availability dates for the machines and release dates for the jobs to minimize a regular additive cost function. In this work, we develop a new branch-and-price optimization algorithm for the solution of this general class of parallel machines scheduling problems. A new column generation accelerating method, termed “primal box”, and a specific branching variable selection rule that significantly reduces the number of explored nodes are proposed. The computational results show that the approach solves problems of large size to optimality within reasonable computational time.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Parallel machines scheduling; Setup times; Column generation; Branch-and-price

1. Introduction

The study of parallel machine problems is relevant from both the theoretical and the practical points of view. From the practical point of view, it is important because we can find many examples of the use of parallel machines in the real world. The motivation for this work was a real textile industry problem involving the production of tissues of different colors on unrelated parallel machines (the production bottleneck) with long sequence dependent setup times and availability dates for the machines and release dates for the jobs. From the theoretical point of view, it is a generalization of the single machine problem and a
particular case of problems arising in flexible manufacturing systems. For a literature review on parallel machines, see Hoogeveen et al. (1997).

Column generation is used to solve models obtained by a Dantzig–Wolfe decomposition (Dantzig and Wolfe, 1960). For a survey on its use to solve integer programming problems, see Barnhart et al. (1998). A recent publication on column generation is Lübbecke and Desrochers (in press). It has been also successfully applied to solve parallel machines scheduling problems, as follows.

van den Akker et al. (1999) studied the problem of minimizing the total weighted completion time of \( n \) jobs on \( m \) identical parallel machines. The problem was formulated as a set-covering problem with an exponential number of binary variables, \( n \) covering constraints, and a single side constraint. The computational results show that the lower bound is singularly strong, and that the outcome of the linear program is often integer. Their algorithm solves problems with \( n = 100 \) and \( m = 10 \) within reasonable computational time.

Chen and Powell (1999a) considered the problem of scheduling \( n \) jobs on \( m \) identical, uniform, or unrelated machines with two particular objectives: to minimize the total weighted completion time and to minimize the weighted number of tardy jobs. They first formulate these problems as integer programs, and then reformulate them, using a Dantzig–Wolfe decomposition, as set partitioning problems. Each column represents a machine schedule, and is generated by solving a single machine subproblem. Branching is conducted on the variables of the original integer formulation. The computational results indicate that this approach is promising and capable of solving problems with \( n = 100 \) and \( m = 20 \) for the first objective, and \( n = 100 \) and \( m = 10 \) for the second objective.

Chen and Powell (1999b) proposed an approach for the problem of scheduling \( n \) jobs with an unrestricted large common due date on \( m \) identical parallel machines to minimize the total weighted earliness and tardiness. The problem was first formulated as an integer program and then reformulated as a set partitioning problem with side constraints. Columns represent partial schedules on single machines and are generated by solving two single machine subproblems. A branch-and-bound algorithm is used to find an optimal integer solution for the problem. The computational results show that the algorithm solves problems with up to 60 jobs in reasonable time.

Chen and Lee (2002) considered the problem of scheduling a set of independent jobs on identical parallel machines to minimize the total earliness-tardiness penalty of the jobs. All the jobs have a given common due window. The problem is formulated as a set partitioning type problem. The computational results show that the approach solves problems with \( n = 40 \) and \( m = 6 \) within a reasonable computational time.

Chen and Powell (2003) studied the multiple job families scheduling problem on identical parallel machines, with sequence dependent or sequence independent setup times, to minimize the total weighted completion time of the jobs. The computational results show that the approach solves problems with 8 families, \( n = 40 \) and \( m = 6 \) within reasonable computational time.

In this work, we develop a new branch-and-price optimization algorithm for the problem of scheduling a set of independent jobs, with release dates and due dates, on unrelated parallel machines with availability dates and sequence-dependent setup times, to minimize the total weighted tardiness. This problem is strongly NP-hard, because its special case \( P_{\text{m1}} \sum w_j T_j \) is known to be strongly NP-hard, even when \( m = 1 \) (Lenstra et al., 1977).

The problem considered here is more general, and has a more complex structure than the parallel machines scheduling problems found in the literature: its binary integer model has a larger scale, since machines are non-identical; the subproblems are more difficult to handle, because there are availability dates for the machines, release dates for the jobs and sequence-dependent setup times. Furthermore, it lacks the structural properties that were essential to the success of the works mentioned above. In fact, the identification of structural dominance properties (usually corresponding to some job ordering restrictions) is a key step to reduce the solution state space of the subproblems and, consequently, to make possible the solution of larger instances.
To our knowledge, the exact solution of this problem has never been attempted before. The algorithm developed finds optimal solutions for large instances. Its success is not only due to the very good lower bounds obtained in the linear relaxation, but also to accelerating techniques that explore some properties of the problem.

The solution of the linear relaxation of column generation processes could be accelerated appreciably by acting in the course of the dual variables, as follows.

Du Merle et al. (1999) presented a stabilization procedure, which has the following interpretation: first, a dual box is defined, and a penalty is imposed when the dual variables lie outside the box; if that box does not contain the optimal dual solution, it is re-adjusted, as well as the penalties, and the process is repeated, until an optimal dual solution is found. Ben Amor and Desrosiers (2003) use other stabilization functions to get improved results.

Valério de Carvalho (2005) derived optimal dual cuts for the cutting stock problem, and showed that adding the corresponding extra columns to the model before starting the column generation, thus constraining the dual space during the entire process, accelerates the algorithm. Ben Amor et al. (in press) showed that if a dual optimal solution is known a priori, and if that information is given to the model, the process is driven to an optimal primal solution much faster. In all cases, there is an extraordinary reduction in the number of columns generated, and in the computing times.

In this paper, we propose a column generation accelerating method, denoted by “primal box”, which may be particularly important in hard problems, like the one presented here, where it is not easy to identify structural dominance properties. Besides this method, we also present a specific branching variable selection rule that significantly reduces the number of nodes explored.

This paper is organized as follows. In the next section, we describe the problem. In Section 3, we formulate it, and describe the decomposition approach. In Section 4, we develop a branch-and-bound algorithm for solving the formulation, based on column generation. In Section 5, we present stabilization and accelerating techniques. In Section 6, we describe the computational experiments, and report the results. Finally, in Section 7, we conclude the paper.

2. Problem description

We consider the problem of scheduling \( n \) independent jobs \( N = \{1,2,\ldots,n\} \) on \( m \) unrelated parallel machines \( M = \{1,2,\ldots,m\} \) to minimize the total weighted tardiness of the jobs. Each machine \( k \in M \) has an availability date \( d_k \) and a configuration status \( l_k \) that represents the last job processed. Each job \( j \in N \) is to be processed by exactly one of the machines, and has \( m \) processing times \( p_{jk} \) \( (k \in M) \), a weight \( w_j \), a release date \( r_j \), and a due date \( d_j \). A sequence dependent setup time \( s_{ij} \) is incurred whenever \( i \neq j \). The initial machine configuration status is represented by the vector \( J \), and \( s_{l_{ij}}, l \in J \), is the setup time of the first job processed by machine \( k \). The setup of the machine can be started and completed during the idle time. If a machine is ready to process a job but the job has not been released yet, it stays idle until the release date of the job. If a job is completed after date \( d_j \) it will incur a tardiness penalty \( w_j \) per unit of tardiness time. All the parameters are deterministic non-negative integers. As commonly assumed in the literature (Potts and Kovalyov, 2000), setup times satisfy the triangle inequality: \( s_{ij} + s_{jk} \geq s_{ik} \), for all \( i,j,k \in N \).

The mathematical notation for this general class of problems is \( R|a_k,r_j,s_{ij}|\sum w_jT_j \). A machine schedule is a subset of jobs of \( N \) processed by the machine in a sequence that satisfies the machine availability constraints and the release dates of the jobs. A feasible machine schedule is a machine schedule in which each job is processed only once. We want to find the set (\( \leq m \)) of feasible machine schedules that processes all jobs, and minimizes the total weighted tardiness of the jobs.
3. Problem formulation

3.1. Integer programming formulation

Define the following binary variables for \( i, j \in \mathcal{N} \) and \( k \in \mathcal{M} \). The variable \( x_{kij}^j = 1 \) if job \( j \) is scheduled immediately after job \( i \) on machine \( k \); \( x_{kij}^j = 0 \), otherwise. The variable \( x_{k0j}^j = 1 \) if the first job processed by machine \( k \) is job \( j \); \( x_{k0j}^j = 0 \), otherwise; the variable \( x_{kn+1}^j = 1 \) if the last job processed by machine \( k \) is job \( j \); \( x_{kn+1}^j = 0 \), otherwise. \( C_j \) is the completion time of job \( j \).

An integer programming formulation (IP) for the problem is the following:

\[
\text{IP:} \quad \min \sum_{j \in \mathcal{N}} \max \left(C_j - d_j, 0 \right) w_j, \quad (3.1)
\]

s.t.:

\[
\sum_{k \in \mathcal{M}} \sum_{i \in \mathcal{N} \cup \{0\} \cap i \neq j} x_{kij}^j = 1 \quad \forall j \in \mathcal{N}, \quad (3.2)
\]

\[
\sum_{j \in \mathcal{N}} x_{k0j}^j \leq 1 \quad \forall k \in \mathcal{M}, \quad (3.3)
\]

\[
\sum_{i \in \mathcal{N} \cup \{0\} \cap i \neq j} x_{kij}^j = \sum_{i \in \mathcal{N} \cup \{n+1\} \cap i \neq j} x_{kij}^j \quad \forall j \in \mathcal{N}, \quad \forall k \in \mathcal{M}, \quad (3.4)
\]

\[
C_j = \sum_{k \in \mathcal{M}} \sum_{i \in \mathcal{N} \cup \{0\} \cap i \neq j} \left[ \max(C_i + s_{ij}, r_j) + p_{jk} \right] x_{kij}^j \quad \forall j \in \mathcal{N} \quad (3.5)
\]

with \( C_0 = a_k \) and \( s_{0j} = s_{i,j} \) \( \forall l \in \mathcal{J} \),

\[
x_{kij}^j \in \{0,1\} \quad \forall i \in \mathcal{N} \cup \{0\}, \quad \forall j \in \mathcal{N} \cup \{n+1\}, \quad \forall k \in \mathcal{M}. \quad (3.6)
\]

The cost function (3.1) minimizes the total weighted tardiness of the jobs. Constraints (3.2) ensure that each job is processed exactly once, and constraints (3.3) guarantee that each machine is used at most once. Constraints (3.4) are flow conservation constraints that ensure that the jobs are correctly ordered within a machine schedule. Eq. (3.5) define the completion time of the jobs considering the machine availability and job release date constraints. Constraints (3.6) impose the binary integrality for the decision variables.

This is a large-scale integer programming problem which is not easy to solve, because it involves a non-linear expression (3.5). We can also observe that this problem is composed by similar subsystems, represented by the \( m \) machines, where the technological coefficient matrices have a block angular structure, i.e., one or more independent blocks linked by coupling equations, with large submatrices of zeros. We can exploit this special structure, and apply a Dantzig–Wolfe decomposition to decompose the problem into a master problem (MP) consisting of the objective function (3.1) and constraints (3.2) and (3.3), and \( m \) subproblems with feasible solution space defined by constraints (3.4) and (3.6) and a cost function based on (3.5). A Dantzig–Wolfe decomposition will be presented in next section.

3.2. Dantzig–Wolfe decomposition

This (IP) problem can be represented on a network as follows (a general reference for problems for which solutions can be represented as paths on networks is Desrosiers et al., 1995). Let \( G = (\mathcal{N}, \mathcal{A}) \) be a directed network where \( \mathcal{A} \) is a set of arcs, and \( \mathcal{N} \) is the set of nodes. The node set contains \( n + 2 \) nodes, \( \mathcal{N} = \{0, 1, \ldots, n, n+1\} \). Node 0 represents the machine start node and node \( n + 1 \) represents the machine end node, respectively. The other nodes represent the jobs to be processed. The arc set \( \mathcal{A} \) consists of the arcs...
A path \( p \) in the network \( G \) is defined as a sequence of nodes \((j_0, j_1, \ldots, j_H, j_{n+1})\), such that each arc \((j_h, j_{h+1})\) belongs to \( A \) and \( H \leq n \). A path that starts at node 0, visits exactly once a subset of jobs of \( N \), and ends at node \( n+1 \), is called an acyclic path. We can now define a feasible machine schedule as an acyclic path in the network \( G \).

In our approach, we will allow machine schedules in which a job may be processed more than once. Let \( P_k \) be the set of machine schedules \( p \) for machine \( k \). Let also \( p_0 \) be the “null path”, a path that starts at node 0 and ends at node \( n+1 \) not processing any job of \( N \) (just the arc \((0, n+1)\)). Each of the machine schedules \( p \in P_k \) correspond to a path in network \( G \), where the variables \( x_{ij}^k \) represent the flow value on arc \((i, j)\). Let \( X^k \) be the set of arc flow values, \( x_{ij}^k \), for machine \( k \). From the network flow theory (Ahuja et al., 1993), we know that a path in a network corresponds to an extreme point, \( X^k_p = (X_{ij}^{kp}) \), of the polytope defined by the convex hull of set \( X^k - \text{conv}(X^k) \). Therefore, we can express any arc flow, \( x_{ij}^k \), as a convex combination of path flows

\[
x_{ij}^k = \sum_{p \in P_k} X_{ij}^{kp} y_{ij}^k \quad \forall (i, j) \in A, \quad k \in M, \tag{3.7}
\]

\[
\sum_{p \in P_k} y_{ik}^k = 1 \quad \forall k \in M, \tag{3.8}
\]

\[
y_{ik}^k \geq 0 \quad \forall p \in P_k, \quad k \in M, \tag{3.9}
\]

where expression (3.8) is referred to as the convexity constraint. Note that, if machine \( k \) is not used, the variable \( y_{ik}^k = 1 \), corresponding to the “null path”—\( p_0 \), is the extreme point that corresponds to the origin of the coordinates. As the \( p_0 \) cost is zero, it will be left out of the formulation and, therefore, the convexity constraint will be represented as an inequality (\( \leq \)).

As \( x_{ij}^k \) can only take binary values, all integral points in the bounded set \( X^k \) are already extreme points of \( \text{conv}(X^k) \), and, therefore, \( y_{ij}^k \) are also binary, because each of the \( x_{ij}^k \) is obtained through a convex combination of only one extreme point of \( \text{conv}(X^k) \), and takes the value 1 if machine schedule \( p \) is used, and 0 otherwise.

If \( C_{jk, p}^k \) is the completion time of job \( j_h \) on a machine \( k \) for schedule \( p \), the completion time of the next job in the sequence, \( C_{j_{h+1}, p}^k \), can be computed by the following relations:

\[
C_{j_{h+1}, p}^k = \max(r_{j_{h+1}, C_{j_k, p}^k + s_{j_{k+1}, j_{h+1}}} + p_{j_{h+1}, k}) \quad \text{with } C_{j_k, p}^k = a_k \quad \text{and } C_{j_{h+1}, p}^k = C_{j_{h+1}, p} \quad \forall k \in M, \tag{3.10}
\]

\[
1 \leq h \leq H.
\]

The cost of an arc \((j_h, j_{h+1})\) is a function of the completion time of job \( j_{h+1} \) in the sequence of jobs. Therefore, we can define the cost of an arc \((j_h, j_{h+1})\) of machine schedule \( p \) as the cost of completing the processing of job \( j_{h+1} \) immediately after the processing of job \( j_h \) at time \( C_{j_k, p}^k \), on machine \( k \)

\[
c_{(j_h, j_{h+1}), p}^k = w_j \left[ \max \left\{ \max(r_{j_{h+1}, C_{j_k, p}^k + s_{j_{k+1}, j_{h+1}}} + p_{j_{h+1}, k} - d_{j_{h+1}, 0}) \right\} \right]. \tag{3.11}
\]

The cost of a machine schedule \( p \), \( c_p^k \), is equal to the sum of the cost of the arcs belonging to \( p \), and can be obtained by the expression

\[
c_p^k = \sum_{h=0}^N c_{(j_h, j_{h+1}), p}^k. \tag{3.12}
\]
An equivalent integer set partition type formulation (ISP) can be written,

\[
\text{ISP:} \quad \min \sum_{k \in M} \sum_{p \in P^k} c_{p}^k y_p^k, \tag{3.13}
\]

\[
\text{s.t.:} \quad \sum_{k \in M} \sum_{p \in P^k} v_{jp}^k y_p^k = 1 \quad \forall j \in N, \tag{3.14}
\]

\[
\sum_{p \in P^k} y_p^k \leq 1 \quad \forall k \in M, \tag{3.15}
\]

\[
y_p^k \in \{0, 1\} \quad \forall p \in P^k, \quad k \in M, \tag{3.16}
\]

where

\[
v_{jp}^k = \sum_{i \in N \cup \{0\}, i \neq j} x_{ijp}^k \quad \forall j \in N, \quad p \in P^k, \quad k \in M, \tag{3.17}
\]

and

\[
x_{ij}^k = \sum_{p \in P^k} x_{ijp}^k y_p^k \quad \forall (i, j) \in A. \tag{3.18}
\]

The first set of constraints (3.14) ensures that each job is processed exactly once; the second set of constraints (3.15) ensures that each machine is used at most once. The relation to the original variables \( x_{ij}^k \) is guaranteed through expression (3.18). We note that the cost function is now linear. The number of columns, represented by the variables \( y_p^k \) corresponding to the machine schedules, grows exponentially with respect to the number of jobs and the number of machines, being extremely large for all but very small sized problems. In practice, this problem cannot be solved directly, i.e., by approaches involving exhaustive column enumeration. It should be addressed by a column generation algorithm instead.

This linear integer program, the MP, a set partitioning problem with side constraints, denoted as (ISP), selects a minimal cost set of machine schedules. It “coordinates” the generation of machine schedules \( p \) in each of the \( m \) subproblems. As it will be detailed, the columns added to the restricted master problem (RMP) will be selected from both acyclic and cyclic machine schedules. Cyclic machine schedules have coefficients, \( v_{jp}^k \), larger than one in the ISP first set of constraints. However, the binary integrality of the decision variables \( y_p^k \), enforced during the branch-and-price phase, guarantees that all the columns in any optimal integer solution will not have coefficients larger than one, i.e., optimal integer solutions will not include cyclic machine schedules.

When we relax the integrality constraints (3.16) of \( y_p^k \) we cannot guarantee the integrality of the original variables through the coupling constraints (3.18), and, therefore, they can be dropped.

The method starts by solving a linear relaxation of the MP with a restricted number of variables (columns), the RMP, and uses the simplex dual prices to generate new columns, in a set of independent subproblems, the pricing algorithms. Attractive columns are added to the RMP if they can potentially decrease the solution value, i.e., if they have a negative reduced cost. The RMP combines the columns added with the previous ones in an optimal way, and computes new dual prices. These are again passed to the subproblems, and the iterations proceed until no more columns with negative reduced cost can be generated. The RMP optimal solution thus obtained is also the optimal solution for the linear relaxation of the MP. The value of this solution is used as a lower bound in a branch-and-bound algorithm to solve the integer problem. At each node of the branch-and-bound tree, additional constraints are added to the linear relaxation problem, and the new problem is again solved by a column generation procedure. In the next section, we discuss the design of the pricing algorithm to solve the subproblems.
3.3. The pricing algorithm

In this section, we will present three dynamic programming models for the pricing algorithm, inspired on previous works, but tailored for the problem under study. The first one (Model 1) is able to price out all feasible machine schedules, but is very time consuming. The solution space of the second one (Model 2) contains more than the feasible machine schedules, but has a pseudopolynomial worst-case complexity. The last one (Model 3) is a 2-cycle elimination version of the second model. Finally, we develop a method to increase the efficiency of the third dynamic programming model.

3.3.1. Model 1

The MP supplies the set of prices (dual variables values) \( p_j \) and \( v_k \), for all \( j \in N \) and \( k \in M \), associated with the first and second sets of constraints of the linear programming program, respectively, needed to solve the subproblems.

Let \( F^k_j(S, t) \), be the minimum reduced cost of the partial machine schedule, going from job “0” to job \( j \), processing only once all jobs in set \( S \) and having completed the processing of job \( j \) at time \( t \), on machine \( k \). \( F^k_j(S, t) \) can be computed by solving the following recurrence relations:

\[
F^k_0(\emptyset), a_k = -v_k, \\
F^k_j(S, t) = \min_{(j,k) \in A} \{F^k_i(S - \{j\}, t') + \max(t - d_j, 0)w_j - \pi_j | t' \leq t - s_{ij} - p_{jk}\}, \\
t = \min_{(j,k) \in A} \{\max(t' + s_{ij} + p_{jk}, r_j + p_{jk}) | t' \in \{F^k_i(S - \{j\}, t')\}\}
\]

for all \( j, S, k \) such that \( j \in N, S \subseteq N, k \in M \).

A special case of this problem is the single machine scheduling problem \( 1||\sum w_j T_j \). As mentioned, this problem is strongly NP-hard, and no pseudopolynomial algorithm is known to solve it.

3.3.2. Model 2

For a vehicle routing problem, Christofides et al. (1981) proposed a state-space relaxation for the dynamic programming subproblems, so as to manage to compute lower bounds, which are not so strong as those that would be obtained with Model 1, but still of good quality. This technique defines a mapping of the original state-space \((S, t)\) onto a new state-space \((j, t)\). In Model 1, states are defined by the set of jobs already processed, \( S \), the last job in the sequence, \( j \) and its completion time, \( t \). In this model, set \( S \) is relaxed and states are defined by the last job processed, \( j \), and its completion time, \( t \). Because variables that define the space do not contain information about the states already visited (belonging to \( S \)), it may happen that, for a given machine schedule, a job is processed more than once, i.e., the solution can be a machine schedule with cycles, and, thus, the solution space of this second model is larger than the first one.

Let \( G^k(j, t) \) be the minimum reduced cost of the partial machine schedule going from node “0” to job \( j \), having completed job \( j \) at time \( t \). The new recurrence relations are

\[
G^k(0, a_k) = -v_k, \\
t = \min_{(j,k) \in A} \{\max(t' + s_{ij} + p_{jk}, r_j + p_{jk}) | t' \in \{G^k(i, t')\}\} | t \leq T, \\
G^k(j, t) = \min_{(j,k) \in A} \{G^k(i, t') + \max(t - d_j, 0)w_j - \pi_j | t' \leq t - p_{jk} - s_{ij}\}
\]

for all \( j, k \) such that \( j \in N, k \in M \),

where \( T \) is an upper bound on the makespan.

Desrochers et al. (1992) and Chen and Powell (2003) have used a similar approach in formulating their problems in the column generation framework. This new problem is a special case of the multiple knapsack problem.
problem, and thus is also NP-hard. However, there are pseudopolynomial algorithms to solve it, based on dynamic programming. In the next subsection we describe a 2-cycle elimination version of this model.

3.3.3. Model 3

A significant number of cyclic machine schedules include parts of the form \( (i, j, i) \). These cycles are known as 2-cycles. For the TSP path relaxation, Houck et al. (1980) have shown that for a \( n \) node problem, the ratio of the number of feasible paths with 2-cycles to the number of feasible paths without 2-cycles is \((1 + 1/(n - 3))^{n-3}\). This ratio is indicative of the reduction in the number of feasible paths in a model when the 2-cycles are eliminated. We now describe a 2-cycle elimination version of the dynamic programming algorithm based on the procedure proposed by Houck et al. (1980). The 2-cycle elimination procedure basically records, for each state \((j, t)\), two partial machine schedules going from node “0” to job \( j \), having completed job \( j \) at time \( t \): \( H^k(j, t) \), the minimum reduced cost partial machine schedule (the best), which completes job \( j \) at time \( t \) after having completed job \( i \) at time \( t' \), and \( H^k_2(j, t) \), the minimum reduced cost partial machine schedule (the next best) which completes job \( j \) at time \( t \) after having completed job \( l \) at time \( t'' \), such that \( l \neq i \). Define \( \text{pred}(j, t) \) as the predecessor job associated with state \((j, t)\). The new recurrence relations are

\[
H^k(0, a_k) = -v_k,
\]

\[
t = \min_{(i,j) \in A} \{ \max(t + s_{ij} + p_{jk}, r_j + p_{jk}) | t' \in \{H^k(i, t')\} | t \leq T, \}
\]

\[
H^k(j, t) = \min_{(i,j) \in A} \{ [H^k(i, t') + \max(t - d_j, 0)w_j - \pi_j | j \neq \text{pred}(i, t')], t' \leq t - p_{jk} - s_{ij} \}, \]

\[
H^k_2(j, t) = \min_{(i,j) \in A} \{ [H^k_2(i, t') + \max(t - d_j, 0)w_j - \pi_j | j \neq \text{pred}(i, t')], t' \leq t - p_{jk} - s_{ij} \}, \]

\[
t' \leq t - p_{jk} - s_{ij}, \]

\[
H^k_2(j, t) = \min_{(i,j) \in A} \{ [H^k_2(i, t') + \max(t - d_j, 0)w_j - \pi_j | j \neq \text{pred}(i, t')], t' \leq t - p_{jk} - s_{ij} \}, \]

\[
for all j, k such that j \in N, k \in M. \]

This new dynamic programming model has a smaller solution space than the previous model since it eliminates the 2-cycles machine schedules. Our preliminary computer results show that the effect of the 2-cycle elimination procedure is bigger for the larger instances, and can reduce the computational time and the number of columns generated by more than 20%. It is also a special case of the multiple knapsack problem, and, thus, NP-hard. However, there are also pseudopolynomial algorithms to solve it, based on dynamic programming (Desrochers et al., 1992).

3.3.4. A method to improve the efficiency of the dynamic programming algorithm

Model 1 is not practical, because it is too hard to solve. Using Model 3 leads to a weaker linear programming relaxation, because cyclic machine schedules are allowed, but represents a compromise to obtain a subproblem that can be solved routinely. Using a cyclic network, we can only stop the column generation process when there are no more machine schedules (cyclic or acyclic) with negative reduced costs. This means more columns generated and, consequently, more computational time. Therefore, the efficiency of the subproblem is of uttermost importance.

The reduced cost of arc \((j_h, j_{h+1})\) of a general machine schedule \( p \), on machine \( k \), \( c^k_{j_h, j_{h+1}} \), can be obtained by the expression

\[
c^k_{j_h, j_{h+1}} = c^k_{j_h, j_{h+1}} - \pi_{j_{h+1}} \quad (3.19)
\]
and the reduced cost of a machine schedule \( p \), \( \bar{c}_p^k \), is

\[
\bar{c}_p^k = -v_k + \sum_{h=0}^{H} \bar{c}_j^k j_{h+1}.
\]

We will show that, during the linear relaxation phase, while searching for an attractive machine schedule in a subproblem, at any labeled state \((i, t')\), we only need to consider negative reduced cost arcs (leading to successor states \((j, t)\)), to obtain “decreasing reduced cost schedules”.

**Definition 1.** Consider a machine schedule \( p \in P^k \). This schedule is called a *decreasing reduced cost schedule* if the reduced costs of all arcs \((j_h, j_{h+1})\), \( \bar{c}_j^k j_{h+1} \), for all \( h \) such that \( 1 \leq h \leq H \) and \( H \geq 2 \), or \( 0 \leq h \leq H \) and \( H = 1 \), are negative.

The following proposition shows that, during the linear relaxation, any machine schedule with a non-negative reduced cost arc is dominated by a decreasing reduced cost schedule.

**Proposition 1.** For additive regular cost functions, the acyclic machine schedule with the most negative reduced cost is a decreasing reduced cost schedule.

**Proof.** The reduced cost of a machine schedule \( \bar{c}_p^k \) is a function of the reduced costs of the arcs \((j_h, j_{h+1})\) included in it, \( \bar{c}_j^k j_{h+1} \), which in turn are a function of the respective job completion times \( C_h^k, C_{h+1}^k, \ldots, C_H^k \). Recall that the setup times satisfy the triangle inequality, \( s_j + s_v \geq s_{jv} \) for all \( i, j, v \in N \), and the processing times are greater than zero. It follows that \( C_h^k < C_{h+1}^k \), and, more generally, \( C_1^k < C_2^k < \cdots < C_N^k \). Consider the following two cases: the second case is necessary, because the triangle inequality does not hold for \( s_{0j} + s_{ij} \geq s_{0i} \), for all \( j, i \in N \).

**Case 1:** \( 1 \leq h \leq H \) and \( H \geq 2 \)—Consider an acyclic machine schedule \( p \) including a single arc \((j_h, j_{h+1})\) with a non-negative reduced cost, \( \bar{c}_j^k j_{h+1} \geq 0 \). If arcs \((j_h, j_{h+1})\) and \((j_{h+1}, j_{h+2})\) are removed from machine schedule \( p \) and arc \((j_h, j_{h+2})\) is added, we will get an acyclic machine schedule \( p' \) such that \( C_{h+2}^k \leq C_h^k \leq C_{h+1}^k \leq C_{h+2}^k \) and, therefore, \( \bar{c}_{j_h,j_{h+2}}^k \leq \bar{c}_{j_{h+1},j_{h+2}}^k \). As \( \bar{c}_{j_{h+1},j_{h+2}}^k \) is negative and \( \bar{c}_{j_h,j_{h+2}}^k \) is a function of the respective job completion times \( C_h^k, C_{h+1}^k, \ldots, C_N^k \), it follows that \( \bar{c}_{j_h,j_{h+2}}^k \leq \bar{c}_{j_{h+1},j_{h+2}}^k \). For \( H = 1 \), we have that \( \bar{c}_{j_h,j_{h+2}}^k \leq 0 \) and, consequently, machine schedule \( p' \) is a decreasing reduced cost schedule.

**Case 2:** \( 0 \leq h \leq H \) and \( H = 1 \)—This case considers the single job processing machine schedules which, by definition, are acyclic. These machine schedules only have two arcs: \((j_0, j_1)\) and \((j_1, j_n)\). As, by definition, \( \bar{c}_{j_0,j_1}^k = 0 \), any machine schedule with \( \bar{c}_{j_0,j_1}^k < 0 \) has a reduced cost more negative than when \( \bar{c}_{j_0,j_1}^k \geq 0 \), which proves the result.

**Corollary 1.** For additive regular cost functions, if there is not any decreasing reduced cost machine schedule, with \( \bar{c}_{j_0,j_1}^k < 0 \), then there is not any acyclic machine schedule with negative reduced cost.

This is an important result because it reduces significantly the solutions space of subproblems by eliminating the *non-decreasing reduced cost* schedules. Preliminary computational results show that it accelerates the column generation process by a factor greater than 7. As we will see, this result does not hold during branch-and-price, where attractive schedules may have non-negative reduced cost arcs. In Section 4, we will describe the needed adjustments.
4. A branch-and-bound algorithm

In this section, we propose a branch-and-bound algorithm to solve the integer problem to optimality.

When the subproblems are not able to generate any more negative reduced cost machine schedules, the simplex algorithm provides the optimal solution for the linear relaxation of the MP. If the values of the decision variables, $y_k$, are all integer, then this solution is also optimal for the original integer program (IP). Otherwise, some of the values of the decision variables, $y_p$, are fractional and we need to explore a branch-and-bound tree to find the optimal integer solution.

The design of the branch-and-bound scheme is very important in a column generation process because the branching decisions are based on columns with fractional solution values, and these decisions have to be compatible with the subproblem structure for the generation of new columns. A branching strategy based on the variables of the MP creates computational difficulties. Note that it is perfectly possible to fix a fractional value variable $y_p$ at one because this information is easily transferable to the subproblem by fixing the value of the flow on each arc of the machine schedule at one. However, fixing $y_p$ at value zero is not easy to accomplish because it is necessary to explore many combinations of the flow values of the arcs of the machine schedules.

4.1. Branching strategy

We follow Chen and Powell (1999a) approach, branching on the decision variables of the original program (IP)—$x^k_{ij}$. We know that if all the arc flow values, $x_{ij}$, are binary integer, the decision variables $y_p$ will also be binary integer (Chen and Powell, 1999a). Therefore, an optimal integer solution for the original program (IP) is obtained when all the arc flow variables have binary integer values.

The branching strategy is based on the underlying network. The root node of the tree corresponds to network $G$, and the child nodes are versions of $G$ modified by imposing restrictions on possible paths. At each node of the branching tree, two new nodes are created, and solved by column generation. If the node optimal solution value is greater than or equal to the incumbent solution value, the node is pruned; otherwise, the lowest value (best-lower-bound rule) active node in the branch-and-bound tree is selected as the next node to be explored, and the branching process continues by selecting a fractional variable to branch on and creating two new branches on this node. The selection of the next variable to branch on is as follows.

The total flow on arc $(i,j)$ of machine $k$ schedules, $x^k_{ij}$, is expressed as a function of $y^k_p$ by the expression (3.18). For each non-zero basic variable ($y^k_p > 0$), we sum the value of the flow on the respective arcs, $x^k_{ij}$. Then, we give a score to each $x^k_{ij}$, which is highest for the most fractional flow values ($\min |x^k_{ij} - 0.5|$) and for the arcs closer to the machine node “0”; in a machine schedule, $(j_0,j_1,j_2,j_3,\ldots,j_{H-1})$, arc $(j_0,j_1)$ is closer to node “0” than arc $(j_1,j_2)$ which is closer to node “0” than arc $(j_2,j_3)$, and so on. The fractional flow value variable with the higher score is selected for branching.

For the variable selected two branches are created: one imposing that the total flow value on the arc $(i,j)$, at machine $k$, is equal to one—$x^k_{ij} = 1$, i.e., the arc $(i,j)$ can only be used on machine $k$ subproblem network, and the other imposing that the total flow value on the arc $(i,j)$, at machine $k$, is equal to zero—$x^k_{ij} = 0$, i.e., the arc $(i,j)$ cannot be used on machine $k$ subproblem network.

In the subproblem, the respective network $G$ must be modified to reflect the restrictions imposed by the branching decisions. If $x^k_{ij}$ is fixed at zero, the arc $(i,j)$ is simply removed from the subproblem network of machine $k$ eliminating the generation of schedules for machine $k$ which include arc $(i,j)$. If $x^k_{ij}$ is fixed at one, the arcs $(i,v) \in A|v \neq j$ and $(l,j) \in A|l \neq i$ are removed from the subproblem network of machine $k$.

Finally, if $x^k_{ij}$ is fixed at zero, the cost of all the columns in the RMP that use arc $(i,j)$ and machine $k$ are penalized, i.e., take a very high cost compared to best integer solution found so far. If $x^k_{ij}$ is fixed at one, the cost of all the columns in the RMP that use machine $k$ and arcs $(i,v) \in A|v \neq j$ and $(l,j) \in A|l \neq i$ are
penalized. In both cases, the solution of the RMP is still feasible, but at a very high cost. After optimizing over the current columns, new columns are generated by the column generation process to obtain a new solution.

After branching constraints are added, we have to search other than decreasing reduced cost schedules, or we may overlook attractive acyclic columns. For example, in the following situations, the only attractive acyclic negative reduced cost schedule may include a non-negative reduced cost arc:

1. arc \((i, j)\) for machine \(k\) is fixed at one:
   a. non-negative reduced cost arc \((i, j)\) for machine \(k\) has to be considered, because all the arcs \((i, v) \in A| v \neq j\) are removed from the subproblem network of machine \(k\), and there is an acyclic negative reduced cost schedule including the partial job sequence \((i, j, v)\);
   b. all non-negative reduced cost arcs \((v, i)\) for machine \(k\) have to be considered, as the arcs \((l, j) \in A| l \neq i\) are removed from the subproblem network of machine \(k\), and there is an acyclic negative reduced cost schedule including the partial job sequence \((v, i, j)\).

2. arc \((i, j)\) for machine \(k\) is fixed at zero:
   a. non-negative reduced cost arc arcs \((i, l)\) for machine \(k\) have to be considered, because the arc \((i, j)\) is removed from the subproblem network of machine \(k\), and there is an acyclic negative reduced cost schedule including partial job sequence \((i, l, j)\).

5. Stabilizing and accelerating the column generation algorithm

5.1. A set covering type model

Our parallel machine scheduling problem has been formulated as a set partitioning type problem in Section 3. However, in the implementation of the algorithm, for the linear relaxation phase, we use an integer set covering type (ISC) (set covering problem with side constraints) formulation that performs better than the ISP. When an optimal linear solution is found, we revert to the ISP program. We use this two step process because the ISC program optimal solution cost can be lower than the optimal solution cost when two or more machines have the same initial configuration status, \(l_k\), and the same job \(j = l_k\) is assigned to more than one machine with a zero setup time (Lopes, 2004). Therefore, the ISP formulation has to be used during branch-and-price to find the optimal integer solution of the \(R|a_k, r_j, s_{ij}| \sum_w w_j T_j\) problem.

The ISC type formulation performs better in the linear relaxation phase, because the dual variables values corresponding to the equality constraints of ISP formulation are less constrained, as they are unrestricted in sign, which impacts negatively the convergence of the column generation algorithm (Valério de Carvalho, 2005). In fact, the linear relaxation of the ISC model has faster convergence and is far more stable, with smaller dual variables oscillation. Our experimental results show that the convergence is approximately 15% faster for the ISC formulation (Lopes, 2004).

5.2. Accelerating the column generation algorithm—primal box

In this section, we introduce a strategy to temporarily reduce the primal solution space termed primal box. The idea behind the primal box strategy is to identify structural characteristics that are likely to occur (verisimilar) in a primal optimal solution and translate those characteristics into additional primal constraints defining a restricted primal solution space (primal box). First, the problem is optimized in this restricted space. Then, this restricted solution space is progressively relaxed until its optimal solution is also the optimal solution of the original solution space. This kind of strategies can be particularly important in
problems, like the one we are studying, in which it is not easy to identify structural dominance properties (usually corresponding to some job ordering restrictions) that are essential to reduce the solution state space of the subproblems and, therefore, to soften up the burden of solving larger problems.

Conceptually, a primal space reduction strategy can be implemented by adding the identified constraints to the compact formulation (IP). Instead, we passed them to the subproblems, reducing their state space, and improving their efficiency. The motivation is the following: in a fast way, we generate a valid primal solution, which is likely to have a near-optimal solution value. It has the same nature of generating a heuristic solution using dual information.

This strategy relaxes the dual solution space, because primal constraints are added to the model. It has been shown that constraining the dual space is advantageous (Valério de Carvalho, 2005). Nevertheless, the net effect of this strategy is positive, because we may expect many columns that might be generated during the standard process (but which are eventually not attractive) not be considered this way.

The solution of the subproblems represents a very large computational burden. The dynamic programming algorithm is pseudopolynomial, and its efficiency depends crucially on the value of $T$ (time upper bound for machine schedules generated in the subproblems). Upper bounds on the value of $T$ impose a limit on the length of the schedules generated by the subproblems. Of course, the value of the makespan of the optimal solution would be an upper bound, but that value is not known a priori. A different way of limiting the length of the schedules generated by the subproblems is using a limit on the number of jobs. Recall that we use a cyclic network to generate the machine schedules in the subproblems. A trivial upper bound for the number of jobs in a machine schedule is $n$ (the total number of jobs), but that usually corresponds to a very large value of $T$.

Our strategy is as follows. In the first phase, we start by restricting the set of columns that can be generated by the subproblems imposing a tentative value of $T$. The optimal solution restricted to this set of columns is determined. Then, it is assessed, if that restricted optimal solution is the optimal solution for the complete problem. If not, the value of $T$ is increased, enlarging the set of machine schedules that can be generated by the subproblem. This process is successively repeated until an optimal solution is found.

This idea was pursued because we intuitively know that an optimal solution should show some kind of machine load balance (the verisimilar characteristic). It explores the following property for parallel machines scheduling problems with regular cost functions:

**Property 1.** Let $i_{H}$ be the last job in machine $k'$ schedule sequence that is completed at time $C'_{i_{H}}$ and $j_{H}$ the last job in machine $k$ schedule sequence that is completed at time $C_{j_{H}}$. If job $j_{H}$ can be moved from the last position of machine $k$ schedule to last position of the machine $k'$ schedule and completed at time $C'_{j_{H}}$ such that $C_{j_{H}} \geq C'_{j_{H}} = \max(C'_{i_{H}} + s_{i_{H}j_{H}}, r_{j_{H}}) + p_{j_{H}k'}$ then, the total cost of this new schedule will not increase.

**Proof.** As the cost function is regular (non-decreasing with the completion times) and is a function of the completion time of the jobs, and the move does not affect other jobs since it is performed at the last positions of the two machine schedules involved in the move, it is obvious that, if $C'_{j_{H}} \leq C_{j_{H}}$, the total cost of the new schedule will not increase. \( \square \)

Assessing if the current restricted optimal solution is the optimal solution for the problem has to be done in different ways during the linear relaxation phase and during the branch-and-price phase, as described in Sections 5.2.1 and 5.2.2, respectively.

### 5.2.1. Linear programming relaxation

Given an optimal solution using the current value of $T$, let $C'_{j_{H}}$ be the makespan of the longest machine schedule found so far. To evaluate if there is any attractive column, the value of $T$ is increased to make more room to include a new job in the longest machine schedule found. For the new value of $T$,
Proposition 2. During the linear relaxation phase, if the subproblems are not able to generate more negative reduced cost machine schedules under the value of \( T \) (restricted problem), nor to generate new decreasing reduced cost machine schedules under the value of an enlarged \( T = T^0 \), then there will be no attractive columns for any value of \( T \), and the current solution is optimal to the entire problem.

Proof. Recall that all machine schedules generated during the solution of the linear relaxation problem should have decreasing reduced costs and the initial (estimated) value of \( T \) covers the processing of all the jobs and, consequently covers all the release dates of the jobs. Let \( L^k_T \) be the schedules generated under the value of \( T \) on machine \( k \). As all machine schedules \( p \in L^k_T \) have non-negative reduced costs (are not attractive) the time upper bound is enlarged to \( T' \), giving room to the add of any of the jobs belonging to \( N \). As the cost function is regular and additive, it better to add the new job in the new time slot than at any latter time and, therefore, if there is a job \( j \) corresponding to a negative reduced cost arc when added to machine schedule \( p \), it will be added to \( p \) generating a new decreasing reduced cost machine schedule.

An initial heuristic solution (the heuristic is described in Section 6.1), is used to set the first tentative upper-bound for the time \( T \) as follows: \( T = \max_{k \in M} \{ C^k_{j_H} \} + \max_{i,j \in N} \{ s_{ij} \} + \max_{k \in M,j \in N} \{ p_{jk} \} \), where \( C^k_{j_H} \) is the completion time of last job processed by the machine \( k \) schedule given by the heuristic. Notice that we may include a new job after the longest schedule of any machine.

This value of \( T \) is the first one used by the subproblems to generate attractive columns such that \( C^k_{j_H} \) is less or equal to \( T \), until the optimal solution is found; then, the value of \( T \) is successively adjusted to \( T' \), until Proposition 2 is observed. The number of jobs in a machine schedule is kept less than or equal to \( n \). Our experiments show that typically few adjustments of \( T \) are needed.

5.2.2. Branch-and-price

During the branch-and-price phase, there may be attractive negative reduced cost acyclic machine schedules including a non-negative reduced cost arc, and Proposition 2 does not apply. After every adjustment of \( T \), the dynamic programming algorithm is used to try to generate more negative reduced cost columns; if negative reduced cost columns are found, they are added to the RMP, until the new optimal solution is found. The adjustment of \( T \) is repeated until the number of jobs in a machine schedule is less than or equal to \( n \).

This accelerating strategy was used, because preliminary computational results show that, taking into account both phases, it accelerates the column generation process by a factor greater than 4, when compared with the strategy of using the upper bound of \( n \) jobs in each machine throughout the entire algorithm.

6. Computational implementation and results

In this section, we describe the method to obtain the first valid RMP solution to initialize the column generation procedure and we report the computational results.

All the algorithms involved were coded in Microsoft Visual C++ Version 6.0, and the CPLEX callable library ("CPLEX Callable Library Version 6.0") CPLEX (1998) was used to solve the linear programming
problems. The computational tests were performed on a portable computer with a 1.8 GHz Pentium Mobile processor and 512 Mb of RAM.

6.1. Initialization of the column generation algorithm

To initialize the column generation algorithm we generate a valid RMP solution through a simple and easy to implement heuristic: first, the jobs are sorted in increasing order of due dates; then, they are assigned by earliest due date to the machine that becomes free first, obeying the release dates of the jobs and the availability dates of the machines.

The main idea is to determine an initial valid solution to the RMP and, at same time, to try to get a balanced load of the $m$ machines. This balanced load scheduling program will be used to calculate an estimate of the scheduling program makespan—max{$C_j$} in the primal box technique presented in Section 5.2.

6.2. Test problems

The test problems were generated as follows. Number of jobs $n \in \{20, 30, 40, 50, 60, 70, 90, 100, 120, 150\}$, number of machines $m \in \{2, 4, 6, 8, 10, 20, 50\}$, job weights $w_j = U[1, 100]$, processing times $P_{jk} = U[10, 80]$, machine availability dates $a_k = U[0, 50]$, machine initial status $l_k = U[1, n]$, job release dates $r_j = U[0, 100]$, and sequence-dependent setup times $s_{ij} = U[20, 40]$. The due dates, $d_j$, were generated following an adapted version of Ho and Chang (1995) model: $d_j = \max\{a_j, b_q\}$ such that $a_j = \min_{k \in M}(p_{jk} + \max(a_k, r_j) + \max_{i \in N(s_{ij})})$ and $b_q = U[1, 160r/q]$, where $r = n/m$, and $q$ indicates the congestion level of the scheduling system. The larger the $q$, the more congested the system will be, and the more tardy jobs will result, because the $b_q$

![Fig. 6.1. % Late jobs versus congestion level.](image)

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>0.00%</td>
<td>40%</td>
<td>342</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>0.00%</td>
<td>10%</td>
<td>871</td>
<td>23</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.00%</td>
<td>50%</td>
<td>388</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0.00%</td>
<td>0%</td>
<td>917</td>
<td>20</td>
<td>67</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>0.00%</td>
<td>0%</td>
<td>724</td>
<td>9</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0.00%</td>
<td>0%</td>
<td>1135</td>
<td>28</td>
<td>84</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>0.00%</td>
<td>0%</td>
<td>595</td>
<td>5</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>0.00%</td>
<td>0%</td>
<td>873</td>
<td>15</td>
<td>76</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>0.00%</td>
<td>0%</td>
<td>742</td>
<td>9</td>
<td>70</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.00%</td>
<td>11%</td>
<td>732</td>
<td>13</td>
<td>58</td>
</tr>
</tbody>
</table>
due dates time window will be reduced. Five levels of the system congestion were tested \((q = 1, 2, 3, 4, 5)\). Fig. 6.1 illustrates the relationship between the congestion level and the number of tardy jobs, as obtained in our computational experiments. For each congestion level, a pair of values, \(m\) and \(n\), are selected and 10 test problems are randomly generated based on the distributions of the associated parameters.

Table 6.2
Medium size problems and \(q = 2\)

<table>
<thead>
<tr>
<th>(m)</th>
<th>(n)</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>0.20%</td>
<td>30%</td>
<td>436</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1.10%</td>
<td>0%</td>
<td>1525</td>
<td>51</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.00%</td>
<td>0%</td>
<td>571</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0.60%</td>
<td>0%</td>
<td>1598</td>
<td>35</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>0.40%</td>
<td>10%</td>
<td>820</td>
<td>10</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0.30%</td>
<td>0%</td>
<td>1511</td>
<td>32</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>1.00%</td>
<td>0%</td>
<td>527</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>0.40%</td>
<td>0%</td>
<td>1075</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>0.00%</td>
<td>0%</td>
<td>755</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.44%</td>
<td>4%</td>
<td>980</td>
<td>18</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 6.3
Medium size problems and \(q = 3\)

<table>
<thead>
<tr>
<th>(m)</th>
<th>(n)</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>1.40%</td>
<td>80%</td>
<td>479</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1.50%</td>
<td>10%</td>
<td>529</td>
<td>135</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.40%</td>
<td>40%</td>
<td>427</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>1.20%</td>
<td>10%</td>
<td>965</td>
<td>66</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>1.20%</td>
<td>50%</td>
<td>587</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>1.50%</td>
<td>10%</td>
<td>1033</td>
<td>81</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>1.00%</td>
<td>30%</td>
<td>447</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>1.90%</td>
<td>20%</td>
<td>884</td>
<td>41</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>1.20%</td>
<td>20%</td>
<td>650</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>1.26%</td>
<td>30%</td>
<td>667</td>
<td>41</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 6.4
Medium size problems and \(q = 4\)

<table>
<thead>
<tr>
<th>(m)</th>
<th>(n)</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>0.03%</td>
<td>80%</td>
<td>263</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>0.40%</td>
<td>50%</td>
<td>812</td>
<td>57</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.10%</td>
<td>60%</td>
<td>376</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0.20%</td>
<td>50%</td>
<td>846</td>
<td>55</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>0.09%</td>
<td>40%</td>
<td>511</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0.20%</td>
<td>20%</td>
<td>861</td>
<td>66</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>0.10%</td>
<td>60%</td>
<td>488</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>0.40%</td>
<td>10%</td>
<td>756</td>
<td>34</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>0.70%</td>
<td>10%</td>
<td>665</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.25%</td>
<td>42%</td>
<td>620</td>
<td>29</td>
<td>6</td>
</tr>
</tbody>
</table>
6.3. Test results

We have split the test results in two classes: the medium and the large size problems. Tables 6.1–6.5 show the test results for the medium size problems. Tables 6.6–6.10 show the test results for the large size problems. The meaning of the headings is the following:

### Table 6.5
Medium size problems and \( q = 5 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>0.10%</td>
<td>80%</td>
<td>294</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>0.20%</td>
<td>40%</td>
<td>819</td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>0.01%</td>
<td>70%</td>
<td>373</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0.10%</td>
<td>30%</td>
<td>785</td>
<td>56</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>0.10%</td>
<td>50%</td>
<td>550</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0.20%</td>
<td>30%</td>
<td>981</td>
<td>44</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>0.20%</td>
<td>50%</td>
<td>491</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>0.20%</td>
<td>20%</td>
<td>717</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>0.20%</td>
<td>40%</td>
<td>666</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.15%</td>
<td>46%</td>
<td>631</td>
<td>27</td>
<td>5</td>
</tr>
</tbody>
</table>

### Table 6.6
Large size problems and \( q = 1 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>40</td>
<td>0.00%</td>
<td>50%</td>
<td>1035</td>
<td>40</td>
<td>74</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>0.00%</td>
<td>10%</td>
<td>1127</td>
<td>58</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>0.00%</td>
<td>0%</td>
<td>1385</td>
<td>62</td>
<td>104</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>0.00%</td>
<td>10%</td>
<td>1115</td>
<td>31</td>
<td>96</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>0.00%</td>
<td>0%</td>
<td>1379</td>
<td>55</td>
<td>118</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>0.00%</td>
<td>0%</td>
<td>2156</td>
<td>191</td>
<td>162</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>0.00%</td>
<td>0%</td>
<td>1684</td>
<td>89</td>
<td>160</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>0.00%</td>
<td>0%</td>
<td>2037</td>
<td>218</td>
<td>206</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>0.00%</td>
<td>0%</td>
<td>1856</td>
<td>114</td>
<td>205</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.00%</td>
<td>8%</td>
<td>1530</td>
<td>95</td>
<td>135</td>
</tr>
</tbody>
</table>

### Table 6.7
Large size problems and \( q = 2 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>40</td>
<td>1.00%</td>
<td>10%</td>
<td>2490</td>
<td>192</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>0.00%</td>
<td>0%</td>
<td>1906</td>
<td>70</td>
<td>72</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>0.20%</td>
<td>0%</td>
<td>2211</td>
<td>78</td>
<td>73</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>0.40%</td>
<td>0%</td>
<td>1701</td>
<td>37</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>0.70%</td>
<td>0%</td>
<td>1924</td>
<td>50</td>
<td>67</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>1.38%</td>
<td>0%</td>
<td>3065</td>
<td>162</td>
<td>127</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>0.40%</td>
<td>0%</td>
<td>1836</td>
<td>48</td>
<td>77</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>0.10%</td>
<td>0%</td>
<td>3000</td>
<td>151</td>
<td>135</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>0.10%</td>
<td>0%</td>
<td>1528</td>
<td>42</td>
<td>48</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.48%</td>
<td>1%</td>
<td>2185</td>
<td>92</td>
<td>78</td>
</tr>
</tbody>
</table>
**LP-IP gap:** average gap, in percentage, between the linear relaxation and the optimal integer solution values;

*Solved at root:* average percentage of problems solved without any branching;

*Columns generated:* average number of columns generated to obtain the optimal integer solution;

*cpu time:* average computational time to obtain the optimal integer solution, in seconds;

*Branch nodes:* average number of branching nodes explored to obtain the optimal integer solution.

---

### Table 6.8
Large size problems and \( q = 3 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>40</td>
<td>1.10%</td>
<td>0%</td>
<td>2210</td>
<td>524</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>2.80%</td>
<td>10%</td>
<td>2106</td>
<td>331</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>1.80%</td>
<td>0%</td>
<td>1807</td>
<td>272</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>1.80%</td>
<td>0%</td>
<td>1159</td>
<td>110</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>1.70%</td>
<td>0%</td>
<td>1354</td>
<td>140</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>1.18%</td>
<td>0%</td>
<td>2515</td>
<td>563</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>1.00%</td>
<td>0%</td>
<td>1576</td>
<td>158</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>1.80%</td>
<td>0%</td>
<td>3008</td>
<td>637</td>
<td>84</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>1.70%</td>
<td>0%</td>
<td>2433</td>
<td>473</td>
<td>378</td>
</tr>
</tbody>
</table>

**Avg** 1.65% 1% 2019 356 69

---

**Fig. 6.2.** Time versus number of jobs.

**Fig. 6.3.** Time versus number of machines.
From the results presented in Tables 6.1–6.10, we can make the following observations:

- The efficiency of the algorithm decreases with the number of jobs for a fixed number of machines (Fig. 6.2—a fixed number of 10 machines) and increases with the number of machines for a fixed number of jobs (Fig. 6.3—a fixed number of 50 jobs).
- The ratio between the number of jobs and the number of machines (Fig. 6.2), and the problem size have both a direct influence on the efficiency of the algorithm (Fig. 6.4—a fixed ratio \( \frac{n}{m} = 5 \)).
- The efficiency of the algorithm is higher for the lower congestion levels (Fig. 6.5). The scenario of lower congestion levels, \( q = 1 \) and \( q = 2 \), which correspond to percentages of late jobs less than 15\%, is the more realistic in practice, and, for these levels of congestion, the algorithm takes, on average, no more than 218 seconds to solve any of the large size problems.
- The lower bound given by the solution value of the linear relaxation problem is very close to the solution value of the integer problem (global average LP-IP gap is 0.57\%).
- Our algorithm solves large problems in a reasonable computational time.

We also observed that a very substantial part of the computational time is spent on the solution of the subproblem, which indicates a research direction for improvement.

Table 6.9
Large size problems and \( q = 4 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>40</td>
<td>0.50%</td>
<td>10%</td>
<td>2552</td>
<td>608</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>0.50%</td>
<td>0%</td>
<td>1418</td>
<td>277</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>0.90%</td>
<td>10%</td>
<td>1711</td>
<td>347</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>0.30%</td>
<td>40%</td>
<td>1078</td>
<td>106</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>0.60%</td>
<td>0%</td>
<td>1271</td>
<td>181</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>1.90%</td>
<td>0%</td>
<td>3736</td>
<td>2708</td>
<td>98</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>1.00%</td>
<td>0%</td>
<td>2049</td>
<td>407</td>
<td>72</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>1.70%</td>
<td>0%</td>
<td>3125</td>
<td>637</td>
<td>84</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>0.60%</td>
<td>0%</td>
<td>2213</td>
<td>343</td>
<td>263</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.89%</td>
<td>7%</td>
<td>2128</td>
<td>624</td>
<td>66</td>
</tr>
</tbody>
</table>

Table 6.10
Large size problems and \( q = 5 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>LP-IP gap</th>
<th>Solved at root</th>
<th>Columns generated</th>
<th>cpu time</th>
<th>Branch nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>40</td>
<td>0.10%</td>
<td>40%</td>
<td>1401</td>
<td>286</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>0.30%</td>
<td>10%</td>
<td>1382</td>
<td>266</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>0.50%</td>
<td>0%</td>
<td>1695</td>
<td>389</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>0.20%</td>
<td>30%</td>
<td>1078</td>
<td>112</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>0.50%</td>
<td>0%</td>
<td>1264</td>
<td>176</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>0.40%</td>
<td>0%</td>
<td>2365</td>
<td>1332</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>0.40%</td>
<td>0%</td>
<td>1600</td>
<td>210</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>0.80%</td>
<td>0%</td>
<td>3601</td>
<td>2706</td>
<td>210</td>
</tr>
<tr>
<td>50</td>
<td>150</td>
<td>0.40%</td>
<td>0%</td>
<td>2165</td>
<td>271</td>
<td>191</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>0.40%</td>
<td>9%</td>
<td>1839</td>
<td>639</td>
<td>57</td>
</tr>
</tbody>
</table>
7. Conclusion

In this work we developed a branch and price approach to the problem of scheduling a set of independent jobs, with release dates and due dates, on unrelated parallel machines with availability dates and sequence-dependent setup times, to minimize the total weighted tardiness. We propose an accelerating technique (primal box) for the column generation algorithm based on the restriction of the primal space. Furthermore, an original and specific branching variable selection rule was developed.

The algorithm solves problems with 50 machines and 150 jobs in a reasonable computational time. Extensive computational results show the behavior of the algorithm for instances with different sizes and levels of congestion.

Acknowledgements

We thank anonymous referees for their constructive comments, which led to a clearer presentation of the material. This work was partially supported by the Portuguese Science and Technology Foundation (Project POSI/1999/SRI/35568) and by the Algoritmi Research Center of the University of Minho, and was developed in the Industrial and Systems Engineering Group.

References


