Math. Prog. Comp. (2010) 2:259–290
DOI 10.1007/s12532-010-0019-z

FULL LENGTH PAPER

Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems

Artur Pessoa · Eduardo Uchoa · Marcus Poggi de Aragão · Rosiane Rodrigues

Received: 13 August 2010 / Accepted: 6 October 2010 / Published online: 24 October 2010

Abstract This paper presents an exact algorithm for the identical parallel machine scheduling problem over a formulation where each variable is indexed by a pair of jobs and a completion time. We show that such a formulation can be handled, in spite of its huge number of variables, through a branch cut and price algorithm enhanced by a number of practical techniques, including a dynamic programming procedure to fix variables by Lagrangean bounds and dual stabilization. The resulting method permits the solution of many instances of the $P||\sum w_j T_j$ problem with up to 100 jobs, and having 2 or 4 machines. This is the first time that medium-sized instances of the $P||\sum w_j T_j$ have been solved to optimality.

Mathematics Subject Classification (2000) 90C10 · 90C57

A. Pessoa · E. Uchoa (✉)
Departamento de Engenharia de Produção,
Universidade Federal Fluminense, Niterói, Brazil
e-mail: uchoa@producao.uff.br

A. Pessoa
e-mail: artur@producao.uff.br

M. P. de Aragão
Departamento de Informática, PUC Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: poggi@inf.puc-rio.br

R. Rodrigues
Departamento de Ciência da Computação,
Universidade Federal do Amazonas, Manaus, Brazil
e-mail: rosiane@dcc.ufam.edu.br

R. Rodrigues
COPPE-Engenharia de Sistemas e Computação,
Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil

1 Introduction

Let $J = \{1, \ldots, n\}$ be a set of jobs to be processed in a set of parallel identical machines $M = \{1, \ldots, m\}$ without preemption. Each job has a positive integral processing time p_j and is associated with a cost function $f_j(C_j)$ over its completion time. Each machine can process at most one job at a time and each job must be processed by a single machine. The scheduling problem considered in this paper consists of sequencing the jobs in the machines (perhaps introducing idle times) in order to minimize $\sum_{j=1}^{n} f_j(C_j)$.

The generality of the cost function permits one to view many classical scheduling problems as particular cases of the above defined problem. In fact, any single machine or parallel identical machines problem where the cost function is based on penalties for job earliness or lateness (possibly including an infinity penalty for a job started before its release date or finished after its deadline) is included in that general definition. In this paper, we mainly focus on weighted-tardiness scheduling problems. In this case each job has a due date d_j and a weight w_j, and the cost function of job j is defined as $w_j T_j$, where $T_j = \max\{0, C_j - d_j\}$ is the tardiness of job j with respect to its due date. This strongly NP-hard problem [8] is referred in the scheduling literature as $1||\sum w_j T_j$ for the single machine case and as $P||\sum w_j T_j$ for the parallel identical machines case.

Mathematical programming exact approaches for these scheduling problems use two distinct types of formulations [19]: (i) MIP formulations where the job sequence is represented by binary variables and completion times by continuous variables; (ii) IP time-indexed formulations, where the completion time of each job is represented by binary variables indexed over a discretized time horizon (for examples, [5,21,24,29]). The latter formulations are known to yield much better bounds, but cannot be directly applied to many instances due to their pseudo-polynomially large number of variables. Van den Akker, Hurkens and Savelsbergh [28] used column generation to solve the time-indexed formulation introduced in [5]. Avella, Boccia and D'Auria [1] used the same formulation to find near-optimal solutions using Lagrangean relaxation. More recently, Pan and Shi [12] showed that the time-indexed bound can be exactly computed by solving a cleverly crafted transportation problem, while Bigras, Gamache and Savard [3] proposed a column generation scheme that improves upon the one in [28] by the use of a temporal decomposition to improve convergence. Note that the time-indexed bound may still leave a significant duality gap and all exact algorithms based on it sometimes need to explore large enumeration trees.

The branch-cut-and-price (BCP) algorithm presented here works over an extended formulation having an even larger number of variables, one for each pair of jobs and each possible completion time. This formulation was independently introduced in Tanaka, Fujikuma and Araki [26], Sourd [23] (for the single machine case) and in a previous version of this paper [15]. That *arc-time-indexed formulation* is at least as strong as the usual time-indexed formulation. We remark that the arc-time-indexed formulation is almost isomorphic to the arc-capacity-indexed formulation for the Vehicle Routing Problem [13]; the machines correspond to the vehicles, jobs to the clients, and the processing times to client demands. This means that known VRP inequalities

for the arc-capacity-indexed formulation can be easily adapted and used to further strengthen the arc-time-indexed formulation.

By applying a Dantzig-Wolfe decomposition on the arc-time-indexed formulation, a reformulation with an exponential number of columns, corresponding to paths in a suitable network, is obtained. However, solving this reformulation using standard column generation techniques is not practical even for medium-sized instances of the problem. We attribute this to following reasons: (i) extreme degeneracy, in fact, it is often the case that an optimal basis has more than 80% of its variables with value 0; and (ii) an expensive pricing with complexity $\Omega(n^3 p_{avg}/m)$, where p_{avg} is the average job processing time. Therefore, in order to obtain the desired bounds in reasonable times, we used the following additional techniques:

- A dual stabilization procedure to speed-up column generation. We use a procedure, introduced by Wentges [30], that differs from those more frequently found in the literature (like [2,10]) by its simplicity; just one parameter has to be tuned. As a theoretical contribution, we prove here a result concerning the convergence of the procedure that is stronger than the one found in [30].
- An efficient dynamic programming procedure to fix variables in the original formulation by computing conditional Lagrangean bounds (akin to reduced costs) for each such variable. Those fixations are powerful and have a direct impact on the complexity of the pricing process.

The bounds obtained by the arc-time-indexed formulation in the single machine case are very tight [15,26], almost all instances from the literature can be solved without further effort. This is not true for the case with parallel machines; the bounds are usually good but there are still significant duality gaps. Reducing those gaps, by adding cuts, is fundamental for the overall performance of the branch-cut-and-price algorithm. We use Extended Capacity Cuts, a very generic family of cuts presented in [13,27]. Strong branching is also an important component of the algorithm on difficult instances. Since the number of unfixed variables after solving the root node may be relatively small, we also tested the alternative of finishing the instance by feeding an LP with the reduced arc-time-indexed formulation to a MIP solver.

Extensive computational results on $P||\sum w_j T_j$ instances derived from the OR-Library are presented. We also run the instances of Souayah et al. [22] and Tanaka and Araki [25]. The new algorithm performs significantly better than previous algorithms, being very consistent on solving all instances with up to 50 jobs and even solving several instances with 100 jobs.

2 Formulation and cuts

The classical time-indexed formulation for the single machine scheduling problem [5] assumes that all jobs must be processed in a given time horizon ranging from 0 to T. Let binary variables y_j^t indicate that job j starts at time t on some machine.

$$\text{Minimize} \quad \sum_{j \in J} \sum_{t=0}^{T-p_j} f_j(t+p_j) y_j^t \tag{1a}$$

S.t.

$$\sum_{t=0}^{T-p_j} y_j^t = 1 \qquad (j \in J), \tag{1b}$$

$$\sum_{\substack{j \in J, \\ t+p_j \le T}} \sum_{s=\max\{0, t-p_j+1\}}^{t} y_j^s \le 1 \ (t = 0, \ldots, T-1), \tag{1c}$$

$$y_j^t \in \{0, 1\} \qquad (j \in J;\ t = 0, \ldots, T-p_j). \tag{1d}$$

This formulation can be used for the identical parallel machines scheduling problem by changing the right-hand side of (1c) to m.

The proposed formulation also assumes an execution time horizon from 0 to T; the machines are idle at time 0 and must be idle again at time T. Let binary variables x_{ij}^t, $i \neq j$, indicate that job i completes and job j starts at time t on the same machine. Variables x_{0j}^t indicate that job j starts at time t on a machine that was idle from time $t-1$ to t, in particular, x_{0j}^0 indicate that j starts on some machine at time 0. Variables x_{i0}^t indicate that job i finishes at time t at a machine that will stay idle from time t to $t+1$, in particular, variables x_{i0}^T indicate that i finishes at the end of the time horizon. Finally, integral variables x_{00}^t indicate the number of machines that were idle from time $t-1$ to t that will remain idle from time t to $t+1$. Define set J_+ as $\{0, 1, \ldots, n\}$ and $p_0 = 0$. The formulation follows:

$$\text{Minimize} \quad \sum_{i \in J_+} \sum_{j \in J \setminus \{i\}} \sum_{t=p_i}^{T-p_j} f_j(t+p_j) x_{ij}^t \tag{2a}$$

S.t.

$$\sum_{i \in J_+ \setminus \{j\}} \sum_{t=p_i}^{T-p_j} x_{ij}^t = 1 \qquad (\forall\, j \in J), \tag{2b}$$

$$\sum_{\substack{j \in J_+ \setminus \{i\}, \\ t-p_j \ge 0}} x_{ji}^t - \sum_{\substack{j \in J_+ \setminus \{i\}, \\ t+p_i+p_j \le T}} x_{ij}^{t+p_i} = 0 \quad \begin{matrix}(\forall\, i \in J; \\ t = 0, \ldots, T-p_i),\end{matrix} \tag{2c}$$

$$\sum_{\substack{j \in J_+, \\ t-p_j \ge 0}} x_{j0}^t - \sum_{\substack{j \in J_+, \\ t+p_j+1 \le T}} x_{0j}^{t+1} = 0 \quad (t = 0, \ldots, T-1), \tag{2d}$$

$$\sum_{j \in J_+} x_{0j}^0 = m \tag{2e}$$

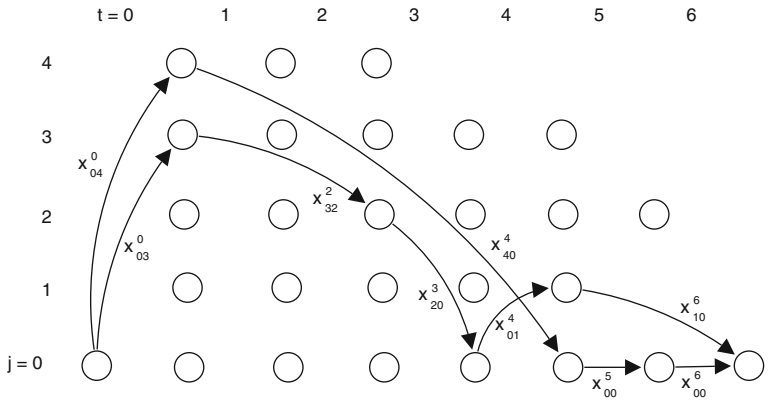

Fig. 1 Example of an integral solution of the arc-time-indexed formulation represented as paths in the layered network

$$x_{ij}^t \in Z_+ \qquad (\forall\, i \in J_+;\ \forall\, j \in J_+ \setminus \{i\};\ t = p_i, \ldots, T - p_j), \tag{2f}$$

$$x_{00}^t \in Z_+ \qquad (t = 0, \ldots, T-1). \tag{2g}$$

Equations (2c), (2d), (2e) and the redundant equation

$$\sum_{i \in J_+} x_{i0}^T = m \tag{3}$$

can be viewed as defining a network flow of m units over an acyclic layered graph $G = (V, A)$. As this flow has just one source and one destination, any integral solution can be decomposed into a set of m paths corresponding to the schedules (sequences of jobs and idle times) of each machine. Constraints (2b) impose that every job must be visited by exactly one path, and therefore must be processed by one machine. An example of the network corresponding to an instance with $m = 2, n = 4, p_1 = 2, p_2 = 1, p_3 = 2, p_4 = 4$, and $T = 6$ is depicted in Fig. 1. The possible integral solution shown has variables $x_{03}^0, x_{04}^0, x_{32}^2, x_{20}^3, x_{01}^4, x_{40}^4, x_{00}^5, x_{00}^6, x_{10}^6$ with value 1. The paths in this solution correspond to the machine schedules $(3, 2, 0, 1)$ and $(4, 0, 0)$, where each unit of idle time in the schedule is represented by a 0.

Proposition 1 *The arc-time-indexed formulation dominates the time-indexed formulation.*

Proof Any solution $\bar{x}$ of the linear relaxation of (2) with cost z can be converted into a solution $\bar{y}$ of the linear relaxation of (1) with the same cost by setting $\bar{y}_j^t = \sum_{i \in J_+ \setminus \{j\}} \bar{x}_{ij}^t$, $j \in J, t = 0, \ldots, T - p_j$. As $\bar{x}$ satisfies (2b), $\bar{y}$ satisfies (1b). In a similar way, the constraints (2c), (2d) and (2e) on $\bar{x}$ imply constraints (1c) (with right-hand size m) on $\bar{y}$.

In order to show that the arc-time-indexed formulation can be strictly better than the time-indexed formulation, define an instance of the $1||\sum w_j T_j$ problem where

$n = 3$; $p_1 = 100$, $p_2 = 300$, $p_3 = 200$; $d_1 = 200$, $d_2 = 300$, $d_3 = 400$; $w_1 = 6$, $w_2 = 3$, $w_3 = 2$; and $T = 600$. The optimal solution of this instance costs 700 and has job sequence (1, 2, 3). The solution of the linear relaxation of the time-indexed formulation with cost 650 is a half-half linear combination of two pseudo-schedules (sequences of jobs and idle times where jobs may repeat): (1, 1, 3, 3) and (2, 2) (the variables with value 0.5 are y_1^0, y_2^0, y_1^{100}, y_3^{200}, y_2^{300}, y_3^{400}). Those pseudo-schedules, where jobs repeat in consecutive positions, cannot appear in the path decomposition of a solution of the linear relaxation of the arc-time-indexed formulation because there are no variables of format x_{jj}^t, $j \in J$. The optimal solution of the arc-time-indexed relaxation is integral for this instance. □

It can be seen that if the arc-time-indexed formulation is weakened by introducing the missing x_{jj}^t variables, one would obtain a formulation that is equivalent to the time-indexed formulation. This means that formulation (2) is just slightly better than formulation (1). For example, by introducing two additional jobs 4 and 5 with zero weights and unit processing times to the previous instance (that now has $n = 5$ and $T = 602$), the linear relaxation of the arc-time-indexed formulation would yield a fractional solution of cost 657.5 corresponding to a half-half combination of two pseudo-schedules: (1, 4, 1, 3, 4, 3) and (2, 5, 2, 5) (the variables with value 0.5 are x_{01}^0, x_{02}^0, x_{14}^{100}, x_{41}^{101}, x_{13}^{201}, x_{25}^{300}, x_{52}^{301}, x_{34}^{401}, x_{43}^{402}, x_{25}^{601}, x_{30}^{602}, x_{50}^{602}). However, the following simple preprocessing steps significantly improve the arc-time-indexed formulation:

Proposition 2 *For jobs i and j in J, $i < j$, let x_{ij}^t and $x_{ji}^{t-p_i+p_j}$ be a pair of variables defined in (2) and let $\Delta = (f_i(t) + f_j(t + p_j)) - (f_j(t - p_i + p_j) + f_i(t + p_j))$. If $\Delta \geq 0$ variable x_{ij}^t can be removed; if $\Delta < 0$, $x_{ji}^{t-p_i+p_j}$ can be removed.*

Proof For any given schedule, swapping two consecutive jobs in the same machine does not affect the rest of the scheduling. Therefore, a schedule where j follows i at time t can be compared with the schedule where i and j are swaped (which means that i follows j at time $t - p_i + p_j$) by computing the Δ expression. If $\Delta > 0$ the first schedule is worse than the second, so x_{ij}^t never appears in an optimal solution. If $\Delta < 0$, $x_{ji}^{t-p_i+p_j}$ never appears in an optimal solution. If $\Delta = 0$, both schedules are equivalent. Therefore, for each pair of variables x_{ij}^t and $x_{ji}^{t-p_i+p_j}$, where $i < j$, (all those pairs are mutually disjoint), one variable can be eliminated without removing at least one optimal solution. □

A similar reasoning shows the following result:

Proposition 3 *For job j in J, let x_{j0}^t and $x_{0j}^{t-p_j+1}$ be a pair of variables defined in (2) and let $\Delta = f_j(t) - f_j(t + 1)$. If $\Delta > 0$ variable x_{j0}^t can be removed; if $\Delta \leq 0$, $x_{0j}^{t-p_j+1}$ can be removed.*

From now on we refer to (2) minus the variables removed by applying Propositions 2 and 3 as the arc-time-indexed formulation. Similarly, the arc set A in the associated acyclic graph $G = (V, A)$ is assumed to not contain arcs corresponding

to the removed variables. It can be seen that the optimal solution of the linear relaxation of this amended formulation on the above defined instance with $n = 5$ is integral. The pseudo-schedules (1, 4, 1, 3, 4, 3) and (2, 5, 2, 5) cannot appear anymore because variables x_{41}^{101}, x_{52}^{301}, x_{43}^{402} are removed by Proposition 2. As far as we know, this formulation was independently introduced in Tanaka, Fujikuma and Araki [26], Sourd [23] (for the single machine case) and in a previous version of this paper [15].

The pseudo-polynomially large number of variables and constraints makes the direct use of this formulation prohibitive. However, one can rewrite it in terms of variables associated to the pseudo-schedules corresponding to the possible source-destination paths in G. Let P be the set of those paths. For every $p \in P$, define a variable λ_p. Define q_a^{tp} as one if arc a^t (associated to the variable x_a^t, $a = (i, j)$) appears in the path p and zero otherwise. Define $f_0(t)$ as zero for any t.

$$\text{Minimize} \sum_{(i,j)^t \in A} f_j(t + p_j) x_{ij}^t \tag{4a}$$

S.t.

$$\sum_{p \in P} q_a^{tp} \lambda_p - x_a^t = 0 \quad (\forall\, a^t \in A), \tag{4b}$$

$$\sum_{(j,i)^t \in A} x_{ji}^t = 1 \quad (\forall\, i \in J), \tag{4c}$$

$$\sum_{(0,j)^0 \in A} x_{0j}^0 = m \tag{4d}$$

$$\lambda_p \geq 0 \quad (\forall\, p \in P), \tag{4e}$$

$$x_a^t \in Z_+ \quad (\forall\, a^t \in A). \tag{4f}$$

Formulation (4), containing both λ and x variables, is said to be in the explicit format [17]. Eliminating the x variables and relaxing the integrality, the Dantzig-Wolfe Master (DWM) LP is written as:

$$\text{Minimize} \sum_{p \in P} \left(\sum_{(i,j)^t \in A} q_{ij}^{tp} f_j(t + p_j) \right) \lambda_p \tag{5a}$$

S.t.

$$\sum_{p \in P} \left(\sum_{(j,i)^t \in A} q_{ji}^{tp} \right) \lambda_p = 1 \quad (\forall\, i \in J), \tag{5b}$$

$$\sum_{p \in P} \left(\sum_{(0,j)^0 \in A} q_{0j}^{0p} \right) \lambda_p = m \tag{5c}$$

$$\lambda_p \geq 0 \quad (\forall\, p \in P). \tag{5d}$$

Note that $\sum_{(0,j)^0 \in A} q_{0j}^{0p} = 1$ for any $p \in P$. A generic constraint l of format $\sum_{a^t \in A} \alpha_{al}^t x_a^t \geq b_l$ can also be included in the DWM as $\sum_{p=1}^{P} (\sum_{a^t \in A} \alpha_{al}^t q_a^{tp}) \lambda_p \geq b_l$,

using the same variable substitution used to convert (4c) into (5b) and (4d) into (5c). Suppose that, at a given instant, there are $r+1$ constraints in the DWM. Assume also that the constraint (5c) has the dual variable π_0, the constraint (5b) corresponding to $i \in J$ has the dual variable π_i, and each possible additional constraint l, $n < l \leq r$, has the dual variable π_l. The reduced cost of an arc $a^t = (i, j)^t$ is defined using the α coefficients of the constraints in the x format as:

$$\bar{c}_a^t = f_j(t + p_j) - \sum_{l=0}^{r} \alpha_{al}^t \pi_l. \tag{6}$$

In the experiments described in this article we separated Extended Capacity Cuts, a very generic family of cuts that have already been shown to be effective on the capacitated minimum spanning tree problem [27] and on many vehicle routing problem variants [13,14].

2.1 Extended capacity cuts

Let $S \subseteq J$ be a set of jobs. Define $\delta^-(S) = \{(i, j)^t \in A : i \notin S, j \in S\}$, and $\delta^+(S) = \{(i, j)^t \in A : i \in S, j \notin S\}$. It can be seen that

$$\sum_{a^t \in \delta^+(S)} t x_a^t - \sum_{a^t \in \delta^-(S)} t x_a^t = p(S), \tag{7}$$

where $p(S) = \sum_{i \in S} p_i$, is satisfied by the solutions of (2). An *Extended Capacity Cut* (ECC) over S is any inequality valid for the polyhedron given by the convex hull of the 0-1 solutions of (7) [27]. In particular, the *Homogeneous Extended Capacity Cuts* (HECCs) are the subset of the ECCs where all entering variables with the same time-index have the same coefficients, similarly for the leaving variables. For a given set S, define aggregated variables v^t and z^t as follows:

$$v^t = \sum_{a^t \in \delta^+(S)} x_a^t \quad (t = 1, \ldots, T), \tag{8}$$

$$z^t = \sum_{a^t \in \delta^-(S)} x_a^t \quad (t = 1, \ldots, T). \tag{9}$$

The balance equation over those variables is:

$$\sum_{t=1}^{T} t v^t - \sum_{t=1}^{T} t z^t = p(S). \tag{10}$$

For each possible pair of values of T and $D = p(S)$, a polyhedron $P(T, D)$ induced by the non-negative integral solutions of (10) is defined. The inequalities that are valid for these polyhedra are HECCs. Dash, Fukasawa and Gunluk [4] recently showed that one can separate over $P(T, D)$ in pseudo-polynomial time, but this would be too time

consuming on the scheduling problems, where T and D can be large. Therefore, we separated HECCs using the same heuristics (they are also responsible for choosing candidate sets S) presented in [27] and also used in [13,14].

3 Branch-cut-and-price algorithm

The pricing subproblem in the BCP algorithm consists of finding a minimum origin-destination path in the acyclic network $G = (V, A)$ with respect to the reduced costs given by (6). This can be done in $\Theta(|A|)$ time as follows. Let $F(j, t)$ denote the minimum-reduced cost subpath that starts at the origin and finishes with an arc $(i, j)^t$. Assume that $\min_{i:(i,j)^t \in A}$ evaluates to infinity when $\{i : (i, j)^t \in A\}$ is an empty set. We use the following dynamic programming recursion:

$$F(j,t) = \begin{cases} 0, & \text{if } j = 0 \text{ and } t = 0; \\ \min\limits_{i:(i,j)^t \in A} \{F(i, t - p_i) + \bar{c}_{ij}^t\}, & \text{otherwise.} \end{cases}$$

If $Z_{sub} = F(0, T) < 0$, there is a column with negative reduced cost. As $|A| = \Theta(n^2 T)$ and T itself is $\Omega(np_{avg}/m)$, where p_{avg} is the average job processing time, solving this pricing problem can be quite time consuming.

Severe convergence problems can be observed when solving the DWM by standard column generation techniques, specially when m is small. In those cases, it is quite possible to have instances where any optimal basis has just m variables with a positive value. This extreme degeneracy is not limited to the final basis; the intermediate bases found during column generation are also very degenerate. It is not unusual to observe situations where hundreds of expensive pricing iterations are necessary to escape from such degenerate points. We also conjecture that the poor convergence of column generation in problems like $P||\sum w_j T_j$ is also related to a kind of variable symmetry, in the sense that there is usually many alternative pseudo-schedules with similar costs (and reduced costs).

3.1 Fixing x variables by reduced costs

The following procedure is devised to eliminate x_{ij}^t variables (and the corresponding arcs from A) by proving that they cannot assume positive values in any integral solution that improves upon the current best known integral solution. This procedure can be applied at any point during the column generation. The direct benefit of this elimination is to reduce the pricing effort at subsequent iterations. However, we verified that the fixing procedure also gives substantial indirect benefits:

- As less origin-destination paths in G are allowed, the number of nearly-equivalent alternative pseudo-schedules is also reduced, improving column generation convergence.
- It is possible that x_{ij}^t variables with a positive value in an optimal fractional solution are removed (this can only happen if this value is strictly less than 1, unless the current best known integral solution is already optimal). This means that the

fixing procedure may improve the lower bounds provided by the arc-time-indexed formulation.

We first define the Lagrangean lower bound $L(\pi)$ that is obtained whenever the pricing subproblem is solved with a vector π of multipliers corresponding to the current constraints in the DWM. While multipliers π_0 to π_n are unrestricted, if $l > n$, π_{n+1} to π_l must be non-negative. This Lagrangean subproblem can be viewed as arising from (4) by dualizing constraints (4c), (4d) and any other x constraint that may have been added. Constraints (4b), (4d), variable bounds and the integrality constraints are kept in this problem. Note that (4d) is both dualized and kept as constraint.

$$L(\pi) = \text{Min} \sum_{a^t \in A} \bar{c}_a^t x_a^t + \sum_{l=0}^{r} b_l \pi_l \tag{11a}$$

$$\text{S.t.}$$

$$\sum_{p \in P} q_a^{tp} \lambda_p - x_a^t = 0 \quad (\forall\, a^t \in A), \tag{11b}$$

$$\sum_{(0,j)^0 \in A} x_{0j}^0 = m \tag{11c}$$

$$\lambda_p \geq 0 \qquad (\forall\, p \in P), \tag{11d}$$

$$x_a^t \in Z_+ \qquad (\forall\, a^t \in A). \tag{11e}$$

For each possible π, an optimal solution of (11) can be constructed by setting $\lambda_{p^*} = m$, where p^* is a path of minimum reduced cost, all other λ variables are set to zero, the x variables are set in order to satisfy (11b). Therefore, the lower bound $L(\pi)$ is equal to $m.Z_{sub} + \sum_{l=0}^{r} b_l \pi_l$. In particular, if π_{RM} is an optimal dual solution (with value Z_{RM}) of the restricted DWM in some iteration of column generation, then $L(\pi_{RM}) = m.Z_{sub} + Z_{RM}$. Of course, any optimal dual solution for the DWM yields $Z_{sub} = 0$, giving a lower bound equal to the optimal value of the DWM.

For a given $a^t \in A$, define $L(\pi, a, t)$ as the solution of (11) plus the constraint $x_a^t \geq 1$. If $L(\pi, a, t) \geq Z_{INC}$, where Z_{INC} is the value of the best known integral solution, variable x_a^t can be removed because it cannot appear in any improving solution. Let Z_{sub}^{at} be the minimum reduced cost of a path that includes the arc $a^t \in A$. An optimal solution can now be defined by setting the λ variable corresponding to a path through a^t with minimum reduced cost to 1 and setting $\lambda_p^* = m - 1$. Then $L(\pi, a, t) = Z_{sub}^{at} + (m-1).Z_{sub} + \sum_{l=0}^{r} b_l \pi_l$. If π_{RM} is an optimal dual solution of the restricted DWM, this expression reduces to $L(\pi_{RM}, a, t) = Z_{sub}^{at} + (m-1).Z_{sub} + Z_{RM}$.

The main point of this subsection is showing that, if we already have solved the above mentioned forward dynamic programming recursion to obtain Z_{sub}, we can obtain Z_{sub}^{at} values for all arcs $a^t \in A$ in $\Theta(|A|)$ time by just solving a second backward dynamic programming as follows. Let $B(i, t)$ denote the minimum reduced cost of a subpath that starts with an arc $(i, j)^t$ and finishes at the destination. Assume that $\min_{j:(i,j)^t \in A}$ evaluates to infinity when $\{j : (i, j)^t \in A\}$ is an empty set. We have the

following dynamic programming recursion:

$$B(i,t) = \begin{cases} 0, & \text{if } i = 0 \text{ and } t = T; \\ \min\limits_{j:(i,j)^t \in A} \{B(j, t + p_j) + \bar{c}_{ij}^t\}, & \text{otherwise.} \end{cases}$$

The value of $Z_{sub}^{(ij)t}$ is then given by $F(i, t - p_i) + \bar{c}_{ij}^t + B(j, t + p_j)$.

Summarizing, this is a strong arc fixing procedure that is about two times as costly as the ordinary pricing step (after the B values are computed, checking which variables are fixed takes additional $\Theta(|A|)$ time). In practice, this cost can be diluted by not trying to fix variables at every column generation iteration.

It must be mentioned that Irnich et al. [7] have also proposed a similar fixing procedure for problems where the columns have a path structure. In their work, the computational experiments were performed for the Vehicle Routing Problem with Time Windows.

3.2 Dual stabilization

Du Merle et al. [10] proposed a *dual stabilization* technique to alleviate the convergence difficulties in column generation, based on a simple observation: the columns that will be part of the final solution are only generated in the last iterations, when the dual variables are already close to their optimal values. They also observed that dual variables may oscillate wildly in the first iterations, leading to "extreme columns" that have no chance of being in the final solution. Their dual stabilization is based on introducing artificial variables forming positive and negative identities in the restricted DWM; the costs and upper bounds of those artificial variables are chosen in order to model a *stabilizing function* that penalizes the dual variables from moving away from the *stability center* $\bar{\pi}$ (the current best estimate on the optimal dual values). The stabilizing function and the stability center are changed during the column generation, until $\bar{\pi}$ converges to an optimal dual solution while the stabilizing function becomes a null function. In fact, several other techniques based on stabilizing functions penalizing dual moves far away from stability centers have been proposed since the 1970s in the context of non-differentiable optimization, as surveyed in [9].

A drawback of all mentioned techniques is the existence of many parameters to be calibrated. Recent implementations of Du Merle et al.-like column generation stabilization [2,11] strongly recommend using a 5-piecewise linear stabilizing function for each dual variable l. Even assuming that those functions will have a symmetry axis at the point $\bar{\pi}_l$, there are still four parameters to be chosen per dual variable. Moreover, one has to determine when and how the stabilizing functions and center should be updated during the column generation. A second drawback of this technique is the increase of the size of the restricted DWM, a 5-piecewise linear dual penalizing function requires four additional variables (that are never removed) by constraint.

The stabilization technique used in this work is much simpler. There is a single scalar parameter α to be controlled and it does not require any change (like additional artificial columns) in the restricted DWM. This tecnique was proposed by Wentges

[30] under the name of *weighted Dantzig-Wolfe decomposition* and can be explained as follows.

Let $\bar{\pi}$ be the current best known vector of dual multipliers, that is, the vector providing the greatest lower bound $L(\cdot)$ among all vectors that have been evaluated. Let π_{RM} be an optimal dual solution (with value Z_{RM}) of the restricted DWM on some iteration of column generation. Instead of solving the next pricing subproblem as usual using the π_{RM} values, one solves the pricing subproblem using the vector

$$\pi_{ST} = \alpha.\pi_{RM} + (1 - \alpha).\bar{\pi},$$

where $0 < \alpha \leq 1$ is a chosen constant. Let s be the path with minimum reduced cost with respect to π_{ST} found by the pricing procedure. If s has a negative reduced cost with respect to π_{RM}, the corresponding column is added to the DWM for the next iteration. Moreover, if $L(\pi_{ST}) > L(\bar{\pi})$, then π_{ST} is an improving dual vector and it becomes the new center of stabilization ($\bar{\pi} \leftarrow \pi_{ST}$). A more algorithmic description of the technique follows:

Algorithm 1 Stabilized Column Generation

Input: parameters α, $0 < \alpha \leq 1$ and $\varepsilon > 0$.

- Initialize the restricted DWM, perhaps using artificial variables, to ensure its feasibility;
- $\bar{\pi} \leftarrow \mathbf{0}$;
- Repeat
  - Solve the restricted DWM, obtaining the value Z_{RM} and the vector π_{RM};
  - $\pi_{ST} \leftarrow \alpha.\pi_{RM} + (1 - \alpha).\bar{\pi}$;
  - Solve the pricing procedure using the vector π_{ST}, obtaining $L(\pi_{ST})$ and the variable with minimum reduced cost s;
  - If $L(\pi_{ST}) > L(\bar{\pi})$ Then $\bar{\pi} \leftarrow \pi_{ST}$;
  - If s has negative reduced cost with respect to π_{RM} Then add s to the restricted DWM;
- Until $Z_{RM} - L(\bar{\pi}) < \varepsilon$

Wentges used the following lemma to prove that the above algorithm is correct, that is, it terminates in a finite number of iterations with an optimal solution of the DWM if ε is sufficiently small.

Lemma 1 (Wentges97) *At a certain iteration of Algorithm 1, if the solution of the pricing subproblem with vector π_{ST} gives a column that already exists in the restricted DWM (a column duplication), then $L(\pi_{ST}) \geq L(\bar{\pi}) + \alpha(Z_{RM} - L(\bar{\pi}))$.*

We prove here a stronger result.

Lemma 2 *At a certain iteration of Algorithm 1, if the solution of the pricing subproblem with vector π_{ST} does not give a column with negative reduced cost with respect to vector π_{RM} (a "misprice"), then $L(\pi_{ST}) \geq L(\bar{\pi}) + \alpha(Z_{RM} - L(\bar{\pi}))$.*

Proof Suppose that the restricted DWM at a certain iteration contains variables corresponding to a set $S \subseteq P$. Define

$$L(S,\pi) = \text{Min} \sum_{a^t \in A} \bar{c}_a^t x_a^t + \sum_{l=0}^{r} b_l \pi_l \tag{12a}$$

S.t.

$$\sum_{p \in S} q_a^{tp} \lambda_p - x_a^t = 0 \quad \left(\forall\, a^t \in A\right), \tag{12b}$$

$$\sum_{(0,j)^0 \in A} x_{0j}^0 = m \tag{12c}$$

$$\lambda_p \geq 0 \qquad (\forall\, p \in S), \tag{12d}$$

$$x_a^t \in Z_+ \qquad \left(\forall\, a^t \in A\right). \tag{12e}$$

Of course, $L(P,\pi) = L(\pi)$. Define $S^+ = S \cup \{s\}$, where s is the path with minimum reduced cost with respect to π_{ST}. We will use the following properties of the functions $L(S,\pi)$, $L(S^+,\pi)$ and $L(\pi)$:

1. For all π, $L(S,\pi) \geq L(S^+,\pi) \geq L(\pi)$.
2. For any fixed $X \subseteq P$, $L(X,\pi)$ is a concave function of π.
3. $L(S,\pi_{RM}) = Z_{RM}$.
4. $L(S^+,\pi_{ST}) = L(\pi_{ST})$.
5. If $L(S,\pi_{RM}) > L(S^+,\pi_{RM})$, path s has a negative reduced cost with respect to π_{RM}.

Now, assume that $L(\pi_{ST}) < L(\bar{\pi}) + \alpha(Z_{RM} - L(\bar{\pi}))$. Since $L(\bar{\pi}) \leq L(S^+,\bar{\pi})$, we obtain that

$$L(\pi_{ST}) = L\left(S^+,\pi_{ST}\right) < \alpha L(S,\pi_{RM}) + (1-\alpha) L\left(S^+,\bar{\pi}\right). \tag{13}$$

By the concavity of $L(S^+,\pi)$, we have

$$L(S^+,\pi_{ST}) = L\left(S^+,\alpha\pi_{RM} + (1-\alpha)\bar{\pi}\right) \geq \alpha L\left(S^+,\pi_{RM}\right) + (1-\alpha) L\left(S^+,\bar{\pi}\right). \tag{14}$$

From (13) and (14), we obtain that $L(S,\pi_{RM}) > L\left(S^+,\pi_{RM}\right)$. □

A *misprice* happens when there are columns with negative reduced cost with respect to vector π_{RM} but the solution of the pricing subproblem using π_{ST} does not provide such a column. Lemma 1 does not rule out the occurrence of an exponentially large sequence of misprices, each followed by a minimal improvement of $\bar{\pi}$, without any column duplication. Lemma 2 is much stronger, because it implies that each misprice guarantees that the gap $Z_{RM} - L(\bar{\pi})$ is reduced by at least a factor of $1/(1-\alpha)$. Therefore, the total number of misprices in the algorithm is polynomially bound.

Some additional remarks about the used stabilization:

– In order to obtain the desired stabilizing effect, the value of α should be small, so the vector π_{ST} does not deviate much from the current stability center $\bar{\pi}$. However, in the first iterations of column generation, when π_{RM} is usually a poor dual solution and Z_{RM} is large, we found that it is better to be more aggressive, using even smaller values of α. Let Z_{INC} be the value of the best known integral solution. If $Z_{RM} > Z_{INC}$ then we set

$$\alpha = 0.1 \frac{Z_{INC} - L(\bar{\pi})}{Z_{RM} - L(\bar{\pi})},$$

otherwise we set $\alpha = 0.1$. Note that a misprice in the first iterations, when Z_{RM} is still larger than Z_{INC}, only guarantees that $L(\pi_{ST}) \geq L(\bar{\pi}) + 0.1(Z_{INC} - L(\bar{\pi}))$.
– It is possible that the solution of the restricted DWM is already an optimal basic solution for the DWM, but the gap $Z_{RM} - L(\bar{\pi})$ is still larger than zero. In this case, every subsequent iteration will be a misprice, reducing this gap by a factor of $1/(1 - \alpha)$. In practice, to avoid the need of estimating a proper value for ε that guarantees optimality, we simply change to the standard column generation algorithm (that is, set α to 1) when the gap is smaller than 0.0005. This usually proves the optimality of the current restricted DWM in a single iteration.
– When this column generation stabilization is combined with the fixing procedure described in Subsection 3.1, it can be interesting to adopt a criterion that only attempt the fixing process with improving vectors π_{ST} (that is, those that became the new center of stabilization). In practice this keeps the computational cost of the fixing process very low, without a significant impact on the number of variables that are eliminated.

3.3 Why not using an approximated Lagrangean algorithm instead of exact column generation?

The above described stabilization can be initialized with any $\bar{\pi}$ (for example, zero) and will still converge to an optimal solution of the DWM. However, for the case $m = 1$, when the slow convergence problem is particularly severe, we found an advantage in hot-starting $\bar{\pi}$ by performing a number of iterations of the Volume algorithm, as described in [15]. For the multi-machine case treated in this paper, where $m \geq 2$, no significant advantage was found, so we omitted the hot start for the sake of simplicity.

It is important to explain why we believe that it is critical to solve the DWM to optimality by stabilized column generation instead of using potentially faster Lagrangean multiplier adjustment algorithms to approximate its solution. For example, the subgradient algorithm was successfully used in [26] for the case $m = 1$. The answer is that the duality gaps of the arc-time-indexed formulation are much larger in the multi-machine case, so we need to reduce that gap by adding cuts. The efficiency of cut separation requires the correct fractional solution of the DWM. Separating cuts using the approximated fractional primal solutions provided by those methods can lead to the separation of cuts that are not violated by the true DWM solution and prevent the separation of cuts that are violated. Of course, Lagrangean algorithms can

often be parameterized in order to give dual and primal solutions with arbitrarily good precision, but this impacts their practical performance.

3.4 Primal heuristic

The procedure of fixing by reduced costs, essential for the overall performance of the exact algorithm, requires the value Z_{INC} of a good primal integral solution. We used the values obtained by an external heuristic for the $P||\sum w_j T_j$ problem, fully described in [20].

3.5 Strong branching

Branching is performed be choosing a vertex $j \in J$ and partitioning the variables x_{ij}^t (those entering j) into two sets S_1 and S_2 such that $\sum_{(i,j,t)\in S_1} \bar{x}_{ij}^t$ and $\sum_{(i,j,t)\in S_1} \bar{x}_{ij}^t$ are close to 0.5. The variables in S_1 are fixed to zero in the left child node; variables in S_2 are fixed to zero in the right child node.

Computational experiments showed that a good choice of candidate branch sets S_1 and S_2 is critical for the performance of the enumeration. It is often the case that some choices lead to no improvement at all in the lower bounds of both child nodes, whereas another apparently similar choice closes a significant fraction of the gap in both children. In order to obtain a consistent algorithm we had to resort to a strong branching, testing eight possible choices before each branch.

3.6 Switching to a MIP solver

When the root node is solved and the current solution is still fractional, the default decision is to perform the above mentioned strong branching process. In principle, in a branch-cut-and-price algorithm, column and cut generation should be also performed in the child nodes in order to better improve the lower bounds, which can be slow in a large branch tree.

There is an alternative to that, at least when the fixation procedure in the root node has fixed a large number of x variables. In those cases, it is often advantageous to complete the solution of the instance by feeding the arc-time formulation, restricted to the non-fixed variables, and including all Extended Capacity Cuts that are active in the current restricted DWM, to a MIP solver (we used CPLEX 11.1). Even though this potentially leads to larger search trees (since Extended Capacity Cuts are not separated anymore), the time to solve each node can be significantly smaller. This alternative also benefits from several advanced algorithmic features implemented in modern commercial MIP solvers. In this study, the CPLEX MIP solver was adopted with default parameters, except by forcing the use of strong branching (parameter CPX_PARAM_VARSEL set to 3). We verified that the use of strong branching improves significantly upon the CPLEX default option for choosing branching variables. In some instances where the optimal solution value is larger than 10^4, we also had to

decrease the MIP relative-optimality tolerance (parameter CPX_PARAM_EPGAP) from 10^{-4} to 10^{-6} to avoid the risk of missing the true optimal solution.

The threshold (maximum number of non-fixed variables) for testing this alternative was set to 200,000 in our experiments. The MIPs for the cases where more than that number of variables remain are too difficult to be handled by CPLEX.

4 Computational experiments

All experiments were performed on machines with an Intel Xeon E5410 Quad Core processor (but using a single core) with a clock of 2.33 GHz, 8 GB of RAM, and the Linux operating system. The linear-programming solver used inside the BCP code was CPLEX 11.1.

The OR-Library has 375 instances of the classical $1||\sum w_j T_j$ problem. This set was generated by Potts and Wassenhove [18] and contains 125 instances for each $n \in \{40, 50, 100\}$. In fact, for each such n, five similar instances were created (changing the seed) for each of 25 parameter configurations of the random instance generator. Therefore, for each value of n there are 25 groups composed of five similar instances. These parameter configurations have influence on the distribution of the due dates. Processing times and weights are always picked from the discrete uniform distribution on [1,100] and [1,10], respectively.

In order to perform our main experiments on the $P||\sum w_j T_j$ problem, we derived 150 instances from the OR-Library instances. For $n \in \{40, 50, 100\}$, and $m \in \{2, 4\}$, we pick the first $1||\sum w_j T_j$ instance in each group (those with numbers ending with the digit 1 or 6) and divided each due date d_j by m (and rounded down the result); processing times p_j and weights w_j are kept unchanged. For example, from instance wt40-1, we produced instances wt40-2m-1 and wt40-4m-1 by dividing due dates by 2 and 4, respectively.

There is no need for idle times between consecutive jobs on the same machine in these tardiness problems. So, we assume that idle times may appear only at the end of the schedule. For the $1||\sum w_j T_j$ problem, T is defined as $\sum_{j=1}^{n} p_j$. For the $P||\sum w_j T_j$ problem, we can set T as $\lfloor(\sum_{j=1}^{n} p_j - p_{max})/m\rfloor + p_{max}$, where p_{max} is the maximum processing time of a job. This value of T is valid because if a job i completes after $\lfloor(\sum_{j=1}^{n} p_j - p_i)/m\rfloor + p_i$ in a certain machine, then at least one other machine was available since time $\lfloor(\sum_{j=1}^{n} p_j - p_i)/m\rfloor$. That job can be moved to that machine, reducing its completion time.

Tables 1–6 present detailed results of the proposed BCP algorithm over the 150 newly derived instances. The first column gives the initial upper bound provided by our heuristic. It should be noted that 24 of those instances (those with numbers 51, 76, 101, or 106) are almost trivial, in the sense that they have a solution of value 0 that can be found by the heuristic in less than a millisecond. Checking if a $1||\sum w_j T_j$ instance has a solution of value 0 can be done in polynomial time [16], but this is not the case for a $P||\sum w_j T_j$ instance. Since a solution of value 0 must be optimal, the BCP algorithm is not run on those instances. The next set of columns give information about the first column generation run (stabilized and with fixing by reduced costs) used to solve the arc-time-indexed formulation: the lower bound obtained, the number of

Table 1 Detailed results over the OR-Lib instances with $m = 2$ and $n = 40$

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
1	606	584	243	17.4	13085	606	7	63.5	0	1	80.9			606
6	3886	3875	290	15.9	49131	3886	3	11.8	0	1	27.7			3886
11	9617	9592	320	15.9	29788	9617	2	6.4	0	1	22.3			9617
16	38356	38279	393	17.3	21291	38356	12	41.8	0	1	59.1			38356
21	41048	41048	320	10.8	0	41048	0	0.0	0	1	10.8			41048
26	87	87	172	14.0	0	87	0	0.0	0	1	14.0			87
31	3812	3758	303	20.0	55479	3812	5	29.2	0	1	49.2			3812
36	10713	10662	350	15.0	14690	10713	12	50.7	0	1	65.7			10713
41	30802	30802	384	14.8	0	30802	0	0.0	0	1	14.8			30802
46	34146	34146	451	8.6	0	34146	0	0.0	0	1	8.6			34146
51	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
56	1279	1272	250	16.0	75069	1279	2	8.3	0	1	24.3			1279
61	11488	11311	367	20.9	37326	11394	25	684.8	26980	69	3104.2	421	2699.7	11488
66	35279	35130	348	19.8	38904	35197	17	200.6	27345	15	528.2	595	1996.4	35279
71	47952	47935	425	14.7	6168	47952	1	0.9	0	1	15.6			47952
76	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
81	573	452	299	16.1	38812	571	10	67.6	0	1	83.7			571
86	6048	5996	341	14.3	8149	6048	13	31.9	0	1	46.2			6048
91	26075	26075	354	20.8	0	26075	0	0.0	0	1	20.8			26075
96	66116	66110	396	21.6	4186	66116	2	1.2	0	1	22.8			66116
101	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
106	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
111	17936	17898	427	24.6	9230	17936	3	3.5	0	1	28.1			17936
116	25874	25786	394	22.2	17389	25870	9	44.4	0	1	66.6			25870
121	64516	64507	471	23.3	3466	64516	1	1.0	0	1	24.3			64516
Avg. total time											172.7		215.3	

Table 2 Detailed results over the OR-Lib instances with $m = 4$ and $n = 40$

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
1	439	438	228	9.7	52690	439	1	1.7	0	1	11.5			439
6	2374	2372	271	8.8	10624	2374	1	2.3	0	1	11.1			2374
11	5737	5735	332	9.8	9618	5737	1	0.5	0	1	10.3			5737
16	21493	21484	389	10.5	4822	21490	5	19.9	0	1	31.2			21493
21	22793	22793	356	8.1	0	22793	0	0.0	0	1	8.1			22793
26	88	88	153	7.5	0	88	0	0.0	0	1	7.6			88
31	2525	2496	268	10.8	30168	2500	6	39.3	29447	203	2982.7	1285	12222.4	2525
36	6420	6355	334	10.8	18124	6364	5	26.5	15010	3103	18836.8	5554	2867.0	6420
41	17685	17634	334	11.8	17708	17637	5	24.3	16797	335	1892.1	156	164.1	17685
46	19124	19124	326	6.1	0	19124	0	0.0	0	1	6.1			19124
51	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
56	826	798	220	9.1	60371	817	7	45.5	55627	>1311	>86400	8	932.6	826
61	7357	7316	358	11.0	12855	7322	5	22.8	11708	41	264.0	353	176.8	7357
66	20251	20247	364	11.2	2074	20251	1	1.7	0	1	12.9			20251
71	26740	26740	365	9.2	0	26740	0	0.0	0	1	9.2			26740
76	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
81	565	550	308	8.5	7067	560	13	37.9	2669	3	56.1	1	47.3	564
86	4725	4719	338	8.5	1113	4725	2	6.8	0	1	15.3			4725
91	15569	15557	361	12.4	2988	15562	7	27.0	1657	3	49.0	10	40.1	15569
96	36266	54158	344	13.0	0	36266	0	0.0	0	1	13.0			36266
101	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
106	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0

Table 2 continued

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
111	11263	11212	350	13.4	13562	11219	10	58.6	11754	41	375.9	199	177.3	11263
116	15566	15539	356	11.6	8198	15545	5	19.7	6171	19	132.7	183	71.7	15566
121	35751	35739	393	12.9	5709	35741	11	39.1	4263	9	94.6	20	56.7	35751
Avg. total time											>4448.8		1675.6	

Table 3 Detailed results over the OR-Lib instances with $m = 2$ and $n = 50$

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
1	1268	1232	266	49.2	306844	1268	8	296.6	0	1	346.6			1268
6	14272	14261	443	59.7	88682	14272	7	63.1	0	1	123.0			14272
11	23028	23000	454	45.8	38025	23028	4	21.3	0	1	67.1			23028
16	46072	46011	465	34.3	24148	46072	6	25.1	0	1	59.4			46072
21	111069	111067	539	35.4	3554	111069	1	0.7	0	1	36.2			111069
26	26	26	217	36.8	0	26	0	0.0	0	1	36.8			26
31	5378	5289	364	50.0	165396	5349	28	1060.2	142821	103	12719.1	107	13107.7	5378
36	18956	18895	465	46.5	31182	18956	4	26.8	0	1	73.3			18956
41	38058	37968	444	39.9	38423	38050	24	341.6	7677	3	398.2	1	425.0	38058
46	82105	82085	513	34.5	13909	82105	8	31.5	0	1	66.0			82105
51	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
56	761	730	278	34.7	205969	761	14	232.7	0	1	267.9			761
61	13682	13589	417	47.4	37625	13619	16	614.0	33201	4271	83784.2	3259	36756.0	13682
66	40907	40904	543	36.9	2404	40907	1	0.6	0	1	37.5			40907
71	78532	78532	528	45.2	0	78532	0	0.0	0	1	45.2			78532
76	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
81	542	538	307	35.2	101422	542	1	5.3	0	1	40.7			542
86	12557	12277	456	52.6	54115	12427	21	946.7	35269	185	14817.0	4703	24069.3	12557
91	47349	47294	512	60.3	29500	47330	19	492.3	11892	7	763.2	27	757.1	47349
96	92822	92803	526	54.6	11299	92822	3	6.3	0	1	60.9			92822
101	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
106	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0

Table 3 continued

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
111	15564	15544	550	63.1	7382	15564	2	3.5	857	1	66.6			15564
116	19609	19524	506	55.5	33410	19573	17	453.3	22588	77	2899.5	2404	9048.3	19608
121	41696	41696	395	41.6	0	41696	0	0.0	1143	1	41.6			41696
Avg. total time											4670.0		3421.3	

Table 4 Detailed results over the OR-Lib instances with $m = 4$ and $n = 50$

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
1	785	777	256	31.6	216432	785	2	24.8	0	1	56.6			785
6	8317	8298	367	36.3	56798	8304	8	110.8	54630	>3668	>86400	1643	78287.1	8317
11	12879	12871	392	27.7	19353	12875	7	71.2	16401	685	5790.5	2	142.7	12879
16	25376	25375	449	23.2	1976	25376	1	8.0	0	1	31.2			25376
21	59440	59440	446	27.5	0	59440	0	0.0	0	1	27.5			59440
26	54	54	189	18.8	0	54	0	0.0	0	1	19.0			54
31	3061	3061	294	28.3	0	3061	0	0.0	0	1	28.4			3061
36	10796	10795	415	30.1	3141	10796	1	5.0	0	1	35.2			10796
41	21806	21783	431	27.8	18111	21785	5	44.9	16198	23	305.1	70	140.8	21806
46	44455	44452	478	25.4	3507	44455	3	15.1	0	1	40.4			44455
51	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
56	570	538	248	22.0	152733	541	4	45.8	151805	>879	>86400	13	18193.8	570
61	7898	7850	389	31.9	21988	7857	7	81.8	17866	>6392	>86400	38969	28111.1	7898
66	23138	23138	453	62.4	0	23138	0	0.0	0	1	62.4			23138
71	42645	42625	451	80.4	17805	42628	10	138.4	13486	23	555.4	31	261.1	42645
76	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
81	495	478	252	54.0	80616	495	3	64.9	0	1	119.2		118.9	495
86	8369	8330	437	78.0	10948	8336	14	163.3	8949	41	816.5	1093	596.9	8369
91	26552	26546	430	76.0	4252	26548	9	111.6	0	1	188.9			26551
96	50326	50312	459	92.2	10053	50317	8	91.0	6193	19	444.3	86	215.6	50326
101	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
106	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0

Table 4 continued

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
111	10069	10049	463	71.4	5576	10051	7	63.2	5102	5	186.6	63	178.4	10069
116	11552	11520	437	59.9	19246	11523	5	54.4	18062	4489	45807.8	6607	4201.5	11552
121	23792	23769	429	26.8	14591	23775	6	58.5	12365	45	523.8	390	333.8	23792
Avg. total time											>12569.6		5250.8	

Table 5 Detailed results over the OR-Lib instances with $m = 2$ and $n = 100$

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
1	3339	3314	643	791.4	2439402	3322	4	814.6	2412912	>18	>14400			≤3339
6	30665	30644	685	673.4	1053912	30665	2	328.1	0	1	1003.0			30665
11	93894	93892	874	595.8	269129	93894	1	17.2	0	1	613.4			93894
16	209100	209062	1120	516.9	47887	209100	12	5086.7	0	1	5603.6			209100
21	457836	457814	9373	1224.4	42326	457836	3	24.3	0	1	1248.7			457836
26	92	92	450	481.0	2662496	92	0	0.0	0	1	485.2			92
31	12729	12725	608	717.2	1329457	12729	1	210.5	0	1	928.4			12729
36	56671	56576	805	631.0	356151	56590	12	804.7	348097	>133	>14400			≤56671
41	237964	237773	1107	751.1	182120	237841	17	19685.2	130127	>1	>20436	2747	141158	237964
46	422831	422804	1133	552.6	49644	422830	17	1525.3	3583	3	2122.0	1	2129.8	422831
51	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
56	5047	4983	784	785.4	1540157	5047	3	796.9	0	1	1585.2			5047
61	45573	45424	899	535.6	141301	45481	34	6818.2	113100	>146	>14400	>255	>86400	≤45573
66	126522	126408	1079	556.8	128031	126477	40	6969.7	64733	266	30952.1	>1273	>86400	126512
71	327305	327300	1266	627.1	8554	327305	1	3.1	0	1	630.3			327305
76	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
81	908	792	686	330.3	443553	830	11	968.9	257499	>2512	>14400			≤908
86	36581	36218	1000	671.2	233519	36322	18	3234.0	179288	>87	>14400	>265	>86400	≤36581
91	129931	129619	1021	642.3	249541	129752	16	3617.1	174798	>100	>14400	>440	>86400	≤129931
96	254194	254141	1137	630.3	74712	254194	12	400.2	0	1	1030.6			254194
101	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
106	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0

Table 5 continued

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
111	84274	84005	1101	846.3	205069	84097	20	6578.0	157461	>82	>14400	>606	>86400	≤84250
116	191198	191085	1020	621.6	118313	191173	25	11118.6	33700	>7	>14400	250	44307.9	191186
121	242022	241954	1279	620.7	74688	241997	29	56938.9	31793	>121	>86400	384	61632.0	242018

Table 6 Detailed results over the OR-Lib instances with $m = 4$ and $n = 100$

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
1	2001	1990	520	669.2	2280314	2001	2	668.1	0	1	1341.1			2001
6	16893	16893	580	595.6	845743	16893	0	0.0	0	1	596.9			16893
11	50236	50199	763	488.1	299798	50203	6	951.3	293335	>102	>14400			≤50236
16	110222	110114	951	498.6	151036	110118	8	950.6	144621	>633	>14400	>1039	>86400	≤110222
21	237392	237389	994	446.0	12716	237390	9	1047.8	8096	3	1499.3	1	1522.8	237392
26	141	143	359	409.7	2365690	141	0	0.0	0	1	413.8			141
31	7130	7081	641	744.8	1496648	7082	5	1156.7	1474884	>27	>14400			≤7130
36	30791	30773	778	547.1	240745	30783	10	1654.2	229134	>168	>14400			≤30791
41	126193	126130	884	570.2	114212	126144	6	923.4	92088	587	12534.9	884	43176.5	126185
46	219537	219526	913	462.0	35811	219529	7	910.0	24884	41	1628.3	75	2172.9	219536
51	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
56	3076	3021	625	624.8	1458684	3030	5	1102.5	1431807	>25	>14400			≤3076
61	24868	24806	931	587.6	76423	24821	9	1094.9	64374	>1216	>14400	>3795	>86400	≤24868
66	67979	67948	872	448.0	65161	67954	8	819.7	53565	210	4153.4	219	5005.9	67967
71	170699	170674	901	464.1	66878	170675	4	398.8	62426	>1401	>14400	5921	81891.0	170689
76	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
81	819	754	673	351.9	332924	772	10	1242.5	287794	>369	>14400			819
86	21299	21209	897	580.4	95102	21218	7	798.6	88728	>1805	>14400	>6289	>86400	≤21299
91	70612	70582	961	478.7	50370	70589	13	1640.7	36760	133	3555.7	1452	9437.4	70606
96	133591	133572	1002	434.8	41742	133575	5	443.8	35748	351	4034.9	241	2808.9	133587
101	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0
106	0	0	0	0.0	0	0	0	0.0	0	1	0.0			0

Table 6 continued

Inst	*Heu UB*	1st.CG				Remaining root node				BCP		Root+CPLEX		*Opt*
		LB	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Nd*	*Time*	
111	46763	46616	947	610.3	159405	46630	9	1094.3	146895	>1269	>14400	>1956	>86400	≤46747
116	101563	101514	873	502.9	80871	101520	8	871.7	73686	>1158	>14400	15186	157814.7	101546
121	127639	127593	902	462.9	75138	127597	8	926.7	67466	>1160	>14400	11516	109453.5	127618

iterations, the running time, and the number of non-fixed arcs in the end of the column generation process. In a few cases (instances wt40-2m-21, wt40-2m-26 and wt40-2m-91) this is enough to solve the instance. The next set of columns give information about the remaining work performed at the root node: the root node lower bound, the number of cut rounds, the running time and the number of non-fixed arcs at the end. Several instances are solved to optimality by the combination of the arc-time-indexed formulation with Extended Capacity Cuts. If this is the case, the next columns will indicate that a single node was solved and the BCP time will be the sum of the two previously reported times. Otherwise, those columns will indicate how many nodes were explored in the search tree and the complete BCP time. The next two columns give information about the alternative of feeding the residual arc-time formulation and the cuts already separated to the CPLEX MIP solver in order to finish the optimization: the number of nodes and the overall time, already including the time already spent in the root node of the BCP. The last column reports the value of the optimal solution. When neither alternative (BCP or Root node+CPLEX) could determine that solution, the symbol $\leq$ indicates that the value is actually the best upper bound found. In those cases, the columns BCP *Time* and CPLEX *Time* actually indicate the limit imposed on the run, where BCP *Nd* and CPLEX *Nd* are the number of nodes explored until the limit. The last row in the Tables 1–4 presents the averages for the total times of full BCP and Root+CPLEX alternatives. Some comments about these results:

- All instances with 40 and 50 jobs could be solved to optimality in less than 86,400 s (= 1 day) by the alternative that used CPLEX to solve the MIP obtained after solving the root node. The alternative of performing a full BCP run with strong branching, although faster in some instances, could not solve four instances with $m = 4$ within that time limit. Note that the instance wt50-4m-11 illustrates how critical the branching decisions can be in these problems. After the solution of the root node, the integrality gap was already reduced to four units. While the BCP run with strong branching explored 685 nodes in order to close that small gap, the CPLEX MIP solver (also with strong branching) could solve the instance in a single branch. There are less frequent cases when the branching decisions took by the BCP run were much better than those by CPLEX.
- The alternative of using CPLEX was also better on most instances with $n = 100$. Overall, 15 out of the 50 instances could not be solved in reasonable times by either method. In order to save computing time, the BCP run was aborted with 14,400 s on unpromising instances. On the other hand, we allowed a little more than 86,400 s to finish the optimization of instances wt100-2m-41, wt100-4m-116 and wt100-4m-121 by CPLEX.

The BCP algorithm was also tested on the set of $P||\sum w_j T_j$ instances used by Souayah et al. [22]; with $n \in \{20, 25, 30, 35\}$, and $m \in \{2, 3\}$. There are 250 instances for each combination of n and m. Table 7 shows for each combination: the number of instances that the best method in [22] could solve within within 1,200 s of CPU time (of a Pentium IV, 1.8 GHz), the number of instances that the BCP algorithm could solve in 400 s of CPU time (one third of the time, in order to roughly compensate for the faster machine we used), the average and the maximum time to solve all instances.

Table 7 Results on the Souayah et al. set of $P||\sum w_j T_j$ instances

n	m	[22]	BCP		
		Solved 1,200 s	Solved 400 s	Avg. time (s)	Max Time (s)
25	2	165	250	12.1	381.7
	3	154	249	13.4	1,691.2
30	2	146	241	51.9	1,177.1
	3	144	247	46.8	4,294.2
35	2	145	237	117.9	6,653.4

Table 8 Results on the Tanaka and Araki set of $P||\sum T_j$ instances

n	m	[25]		Root node + CPLEX	
		Avg. time (s)	Max time (s)	Avg. time (s)	Max time (s)
25	2	1.61	15.0	65.8	4,995.2
	3	21.6	885.1	13.5	646.8
	4	53.2	4,703.3	6.1	210.6
	5	16.0	1,719.3	2.4	50.6
	6	0.47	31.9	1.3	7.2

We also compare our algorithms with the one by Tanaka and Araki [25] on a set of $P||\sum T_j$ instances with $n = 25$ and $m \in \{2, 3, 4, 5, 6\}$, where there are 125 instances for each combination. Table 8 shows for each combination the average and the maximum time (on a Pentium 2.4 GHz processor) for the algorithm in [25], and the average and the maximum time taken by our algorithm (always using the alternative of solving of performing the enumeration using CPLEX). We remark that our general code does not take advantage of the many properties that are known to be valid for the unweighted tardiness objective function.

Table 9 gives statistics on the integrality gaps provided by different relaxations over each set of 25 instances, for $m \in \{2, 4\}$, and $n \in \{40, 50, 100\}$. The first set of columns gives results for the linear relaxation of the time-indexed formulation (1): average integrality gap (*Avg. Gap %*) and maximum integrality gap (*Max Gap %*). The second set of columns shows results obtained by solving the DWM corresponding to the arc-time-indexed formulation, the third set of columns shows the results of the same formulation after the addition of rounds of Extended Capacity Cuts. This table also includes statistics for the case $n = 100, m = 1$. It can be seen that the single-machine and multi-machine problems, even with $m = 2$, are quite different. The bounds provided by the arc-time formulation are so tight on the $1||\sum w_j T_j$ problem that most OR-Library instances are solved with little (if any) additional effort. This is not true for the $P||\sum w_j T_j$ problem. The lower bounds obtained by the arc-time-indexed formulation are not much stronger than the one obtained by the time-indexed formulation. However, the Extended Capacity Cuts proved to be effective in reducing the integrality gap of the arc-time-indexed formulation (these cuts cannot be

Table 9 Comparison of different bounding methods

n	m	Time-indexed relax.		Arc-time relax.		Arc-time + cuts	
		Avg. Gap %	Max Gap %	Avg. Gap %	Max Gap %	Avg. Gap %	Max Gap %
40	2	1.533	21.016	1.243	20.840	0.042	0.818
	4	0.544	4.787	0.406	3.390	0.200	0.872
50	2	0.535	4.074	0.487	4.074	0.090	1.035
	4	0.529	5.614	0.489	5.614	0.274	5.088
100	1	0.540	13.430	0.023	0.535	0.000	0.000
	2	1.801	26.087	0.688	12.775	0.422	8.590
	4	0.505	7.937	0.493	7.936	0.362	5.738

Table 10 Comparison of different methods for solving the DWM problem on instances with $n = 100$

	Method	Time (s)	Iter	St. Chgs	Misprices	R.Arcs
$m = 2$	A	>7,645.0	4005.4	–	–	4850961
	B	1,687.7	855.0	240.7	25.2	4850961
	C	>2,666.6	27163.0	–	–	466103
	D	588.8	1122.8	241.1	20.8	465998
$m = 4$	A	2,163.0	1664.6	–	–	3391936
	B	887.4	666.0	234.0	27.4	3391936
	C	824.7	1592.9	–	–	388508
	D	395.4	678.7	237.5	25.8	413816

introduced in the time-indexed formulation). This is crucial for the performance of the BCP algorithm: several OR-Library instances with only 40 jobs could not be solved without the cuts.

The last table aims at illustrating the importance of fixing by reduced costs and of the stabilization presented in Sect. 3. Table 10 contains statistics about the solution of the first DWM, corresponding to the arc-time-indexed formulation, on the OR-Library instances with $n = 100$. Method A is the standard column generation process, the restricted DWM is initialized with $2n$ artificial columns corresponding to the pseudo-schedules $(1, \ldots, i-1, i+1, \ldots, n)$ and $(1, \ldots, i-1, i, i, i+1, \ldots, n)$, for all $i \in J$, and a sufficiently large cost. This initialization is significantly better than using artificial variables forming an identity matrix or using the columns from the best known integral solution. We do not think that column generation on Method A is implemented in a naive way; we did our best to make it work using the standard techniques used with success in previous works [6,13,27]. In this case we report the average time in seconds (*Time (s)*) and the average number of iterations (*Iter*). Since no fixing of variables by reduced costs is performed, column *R.Arcs* gives the average original number of variables (arcs) in the arc-time-indexed formulation. Method B is the stabilized column generation described in Sect. 3.2, initialized with $\bar{\pi} = 0$. In this

case, we also report the average number of changes of the stability center (*St. Chgs*) and the average number of misprices (*Misprices*). Method C is standard column generation, without stabilization, but with fixing by reduced cost as described in Sect. 3.1 performed at every 20 iterations. Column *R.Arcs* gives the average number of remaining (non-fixed) arcs at the end of column generation. Method D is stabilized column generation process combined with fixing by reduced costs. The fixing procedure is only called after an improvement in $\bar{\pi}$, but still keeping a minimum interval of 20 iterations between two consecutive fixings. Methods A and C can take a great deal of time on some instances with $m = 2$; we stopped those instances at 14,400 s.

5 Conclusions

This paper presents a set of algorithms, bundled into a branch-cut-and-price code, for several important multi-machine scheduling problems. Very good experimental results are shown. Several instances of the $P||\sum w_j T_j$ problem of reasonable size can now be solved to optimality. The paper also stresses the need for sophisticated techniques in order to handle the large sized arc-time-indexed formulation.

Another contribution lies in bridging parallel machine scheduling and vehicle routing problems. In fact, the arc-time-indexed formulation used in this paper is so akin to the VRP capacity-indexed formulation presented in [13] that an existing code that separates Extended Capacity Cuts could be used with minimal changes. It is quite possible that further improvements in cuts for the VRP would immediately lead to better scheduling codes. This analogy can be pushed even farther: the capacity-indexed formulation for the heterogeneous fleet VRP presented in [14] could be adapted to solve scheduling problems with heterogeneous parallel machines.

Acknowledgments AP, EU and MPA received support from CNPq grants 309004/09-8, 304533/02-5, and 300475/93-4, respectively. EU also received support from FAPERJ grant E-26/100.504/2007. RR was supported by grant PICDT/CAPES.

References

1. Avella, P., Boccia, M., D'Auria, B.: Near-optimal solutions of large scale single-machine scheduling problems. ORSA J. Comput. **17**, 183–191 (2005)
2. Ben Amor, H., Frangioni, A., Desrosiers, J.: On the choice of explicit stabilizing terms in column generation. Discrete Appl. Math. **157**, 1167–1184 (2009)
3. Bigras, L., Gamache, M., Savard, G.: Time-indexed formulations and the total weighted tardiness problem. INFORMS J. Comput. **1**, 133–142 (2008)
4. Dash, S., Fukasawa, R., Gunluk, O.: On the generalized master knapsack polyhedron. In: Proceedings of the 12th IPCO Conference, Lecture Notes in Computer Science, vol. 4513, pp. 197–209 (2007)
5. Dyer, M., Wolsey, L.: Formulating the single machine sequencing problem with release dates as a mixed integer program. Discrete Appl. Math. **26**, 255–270 (1990)
6. Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragão, M., Reis, M., Uchoa, E., Werneck, R.F.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Math. Program. **106**, 491–511 (2006)
7. Irnich, S., Desaulniers, G., Desrosiers, J., Hadjar, A.: Path-reduced costs for eliminating arcs in routing and scheduling. INFORMS J. Comput. **22**, 297–313 (2010)
8. Lawler, E.: A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. Ann. Discrete Math. **1**, 331–342 (1977)

9. Lemaréchal, C.: Lagrangean relaxation. In: Juenger, M., Naddef, D. (eds.) Computational Combinatorial Optimization, pp. 115–160. Springer, Berlin (2001)
10. du Merle, O., Villeneuve, D., Desrosiers, J., Hansen, P.: Stabilized column generation. Discrete Math. **194**, 229–237 (1999)
11. Oukil, A., Ben Amor, H., Desrosiers, J., El Gueddari, H.: Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems. Comput. Oper. Res. **34**, 817–834 (2007)
12. Pan, Y., Shi, L.: On the equivalence of the max- min transportation lower bound and the time-indexed lower bound for single machine scheduling problems. Math. Program. **110**, 543–559 (2007)
13. Pessoa, A., Poggi de Aragão, M., Uchoa, E. : Robust branch-cut-and-price algorithms for vehicle routing problems. In: Golden, B., Raghavan, S., Wasil, E. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges, pp. 297–326. Springer, Berlin (2008)
14. Pessoa, A., Uchoa, E., Poggi de Aragão, M.: A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. Networks **54**, 167–177 (2009)
15. Pessoa, A., Uchoa, E., Poggi de Aragão, M., Rodrigues, R.: Algorithms over arc-time indexed formulations for single and parallel machine scheduling problems. Tech. Rep. RPEP Vol.8 no.8, Universidade Federal Fluminense, Engenharia de Produção, Niterói, Brazil (2008)
16. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, USA (2002)
17. Poggi de Aragão, M., Uchoa, E. : Integer program reformulation for robust branch-and-cut-and-price. In: Wolsey, L. (ed.) Annals of Mathematical Programming in Rio, pp. 56–61. Búzios, Brazil (2003)
18. Potts, C., Wassenhove, L.: A branch-and-bound algorithm for the total weighted tardiness problem. Oper. Res. **32**, 363–377 (1985)
19. Queyranne, M., Schulz, A.: Polyhedral approaches to machine scheduling. Tech. Rep. 408, University of Berlin (1997)
20. Rodrigues, R., Pessoa, A., Uchoa, E., Poggi de Aragão, M.: Heuristics for multi-machine weighted tardiness problems. Tech. Rep. RPEP Vol.8 no.11, Universidade Federal Fluminense, Engenharia de Produção, Niterói, Brazil (2008)
21. Sadykov, R.: Integer programming-based decomposition approaches for solving machine scheduling problems. Ph.D. thesis, Universite Catholique de Louvain (2006)
22. Souayah, N., Kacem, I., Haouari, M., Chu, C.: Scheduling on parallel identical machines to minimise the total weighted tardiness. Int. J. Adv. Oper. Manage. **1**(1), 30–69 (2009)
23. Sourd, F.: New exact algorithms for one-machine earliness-tardiness scheduling. INFORMS J. Comput. **21**, 167–175 (2009)
24. Sousa, J., Wolsey, L.: A time indexed formulation of non-preemptive single machine scheduling problems. Math. Program. **54**, 353–367 (1990)
25. Tanaka, S., Araki, M.: A branch and bound algorithm with lagrangian relaxation to minimize total tardiness on identical parallel machines. Int. J. Prod. Econ. **113**, 446–458 (2008)
26. Tanaka, S., Fujikuma, S., Araki, M.: An exact algorithm for single-machine scheduling without machine idle time. J. Sched. **12**, 575–593 (2009)
27. Uchoa, E., Fukasawa, R., Lysgaard, J., Pessoa, A., Poggi de Aragão, M., Andrade, D.: Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. Math. Program. **122**, 443–472 (2008)
28. Van der Akker, J., Hurkens, C., Savelsbergh, M.: Time-indexed formulations for machine scheduling problems: column generation. INFORMS J. Comput. **12**(2), 111–124 (2000)
29. Van der Akker, J., Van Hoesel, C., Savelsbergh, M.: A polyhedral approach to single-machine scheduling problems. Math. Program. **85**, 541–572 (1999)
30. Wentges, P.: Weighted dantzig-wolfe decomposition for linear mixed-integer programming. Int. Trans. Oper. Res. **4**, 151–162 (1997)