Algorithms over Arc-time Indexed Formulations for Single and Parallel Machine Scheduling Problems

Artur Pessoa, Eduardo Uchoa
{artur, uchoa}@producao.uff.br
Departamento de Engenharia de Produção
Universidade Federal Fluminense

Marcus Poggi de Aragão
poggi@inf.puc-rio.br
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Rosiane Rodrigues
rosiane@cos.ufrj.br
COPPE - Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
rosiane@dcc.ufam.edu.br
Departamento de Ciência da Computação
Universidade Federal do Amazonas

June, 2008

Abstract

This paper presents algorithms for single and parallel identical machine scheduling problems. While the overall algorithm can be viewed as a branch-cut-and-price over a very large extended formulation, a number of auxiliary techniques are necessary to make the column generation effective. Those techniques include a powerful fixing by reduced costs and a new proposed dual stabilization method. We tested the algorithms on instances of the classical weighted tardiness problem. All the 375 single machine instances from the OR-Library, with up to 100 jobs, are solved to optimality in reasonable computational times and with minimal branching, since the duality gaps are almost always reduced to zero in the root node. Many multi-machine instances with 40 and 50 jobs are also solved to optimality for the first time. The proposed algorithms and techniques are quite generic and can be used on several related scheduling problems.

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 in 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 viewing many classical scheduling problems as particular cases of the above defined problem. For example, one of the most widely studied such problems is the single machine weighted tardiness scheduling problem [19]. In this case each job has a due date d_j and a weight w_j, 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 [12] is referred in the scheduling literature as $1||\sum w_j T_j$. The variant with parallel identical machines is referred as $P||\sum w_j T_j$. 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.

Mathematical programming exact approaches for these scheduling problems use two distinct kinds of formulations [22]: (i) MIP formulations where 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, [8, 25, 29, 24]). The latter formulations are known to yield better bounds, but can not be directly applied to many instances due to their pseudo-polynomially large number of variables. Van den Akker, Hurkens and Savelsbergh [28] were the first to use column generation to solve the classical time indexed formulation introduced by [8]. Avella, Boccia and D'Auria [1] could find near-optimal solutions (gaps bellow 3%) for instances with n up to 400, using Lagrangean relaxation to approximate that bound. Recently, Pan and Shi [15] showed that the classical time indexed bound can be exactly computed by solving a cleverly crafted transportation problem. This result led to a branch-and-bound for the $1||\sum w_j T_j$ that consistently solved all the OR-Library instances with up to 100 jobs, a very significant improvement with respect to previous algorithms. Bigras, Gamache and Savard [5] proposed a column generation scheme that improves upon the one in [28] by the use of a temporal decomposition to improve convergence. The resulting branch-and-bound is competitive and could solve 117 out of the 125 instances with 100 jobs solved by [15]. Anyway, all exact algorithms over the time indexed formulation may need to explore large enumeration trees.

This paper presents an extended formulation for the single and parallel machine scheduling problems using an even larger number of variables, one for each pair of jobs and each possible completion time. This formulation is proved to yield strictly better lower bounds than the usual time indexed formulation. By applying a Dantzig-Wolfe decomposition, it is obtained a reformulation with an exponential number of columns, corresponding to paths in a suitable network. As done in [15, 5], we tested the approach on weighted tardiness instances. However, solving this reformulation using standard column generation techniques is not practical even for medium-sized instances of that problem. We attribute that to following reasons: (i) extreme degeneracy, in fact, when $m = 1$ it can happen that any optimal basis has just one variable with a positive value; (ii) extreme variable symmetry, in the sense that there are usually many alternative solutions with the same cost; and (iii) 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 very efficient procedure to fix variables in the original formulation by reduced costs.
- A dual stabilization procedure to further speedup column generation. This procedure differs from those found in the literature (like [7, 3]) by its simplicity, just one parameter has to be tuned, and by some interesting theoretical properties.
- The possible use of the Volume algorithm [2] in order to provide a fast hot start for the column generation.

After solving that reformulation, besides having a bound close to the optimal, the number of non-fixed variables is usually quite small. One can then proceed by separating cuts. We used the Extended Capacity Cuts, a powerful family of cuts first introduced in [26]. As the fixing procedure applied after each round of cuts is likely to further reduce instance size, at some point is usually advantageous to stop the column generation and continue the optimization using a branch-and-cut (still using the above mentioned cuts) over the few remaining variables. In almost all $1||\sum w_j T_j$ benchmark instances from the OR-Library, with $n \in \{40, 50, 100\}$, we found that the duality gaps were reduced to zero still in the root node.

The same algorithm was also tested on the $P||\sum w_j T_j$, a harder problem. We do not know any paper claiming optimal solutions on instances of significant size. Our branch-cut-and-price could solve instances derived from those in the OR-Library, with $m \in \{2, 4\}$ and $n \in \{40, 50\}$, consistently. However, the solution of several such multi-machine instances did require a significant amount of branching.

This article on scheduling can be viewed as part of a larger investigation on the use of pseudo-polynomially large extended formulations in the context of robust branch-cut-and-price algorithms. Previous experiences include the capacitated minimum spanning tree problem [27] and several vehicle routing problem variants [16, 17].

2 Formulation and Cuts

The classical time indexed formulation for the single machine scheduling problem [8] 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.

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

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

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

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 in 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.

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

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

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

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

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

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

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 Figure 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.*

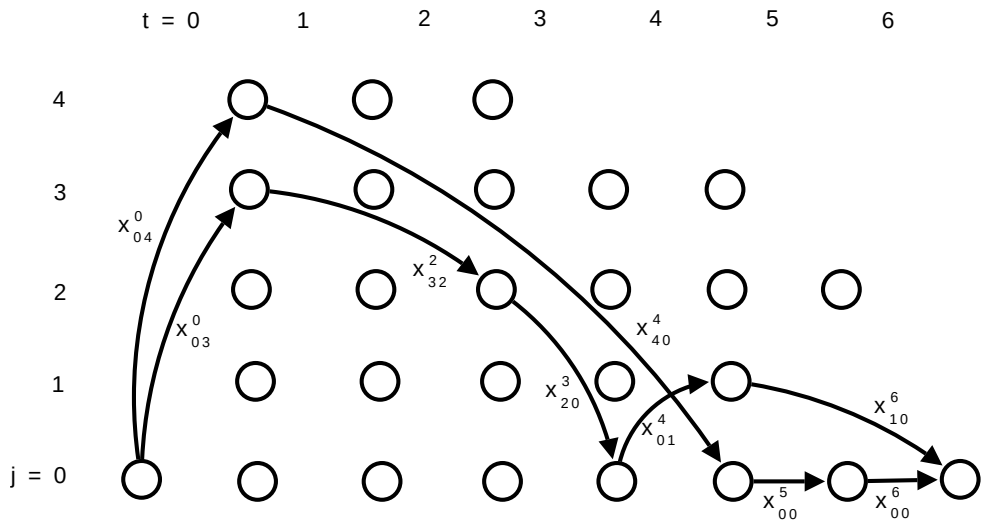

Figure 1: Example of an integral solution of the arc-time indexed formulation represented as paths in the layered network.

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, can not 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 improves 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 that:

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 $(2, 5, 2, 5)$ can not 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 never used in the literature, neither for the general scheduling problem, nor for its particular cases. Picard and Queyranne [18] proposed a three-index formulation for the $1||\sum w_j T_j$ over variables x_{ij}^k, meaning that job j follows job i and is the k-th job to be scheduled. That formulation contains $O(n^3)$ variables and $O(n^2)$ constraints, while the arc-time indexed formulation has $O(n^2 T)$ variables and $O(nT)$ constraints.

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.

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

S.t.

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

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

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

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

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

Formulation (4), containing both λ and x variables, is said to be in the explicit format [20]. Eliminating the x variables and relaxing the integrality, the Dantzig-Wolfe Master (DWM) LP is written as:

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

S.t.

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

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

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

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 [16, 17].

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, the same happening with the leaving variables. For a given set S, define aggregated variables v^t and z^t as follows:

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

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

The balance equation over those variables is:

$$\sum_{t=1}^{T} tv^t - \sum_{t=1}^{T} tz^t = p(S) \quad . \tag{10}$$

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 [6] recently showed that one can separate over $P(T, D)$ in pseudo-polynomial time. Strategies for choosing candidate sets S to perform that separation are discussed in [27, 16, 17].

3 Column Generation

The pricing subproblem 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:

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

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. For example, if $m = 1$, $n = 100$ and $p_{avg} = 50$, there are more than 25 million arcs in A.

Severe convergence problems can be observed when solving the DWM by standard column generation techniques, specially when $m = 1$. In this case, it is quite possible to have instances where any optimal basis has just one variable with a positive value. This extreme degeneracy is not limited to the final basis, the intermediate bases found during the column generation are also very degenerated. It is not unusual to observe situations where hundreds of expensive pricing iterations are necessary to escape from such degenerated points. We also conjecture that the poor convergence of the column generation in problems like $1||\sum w_j T_j$ is also related to a kind of variable symmetry, in the sense that there is usually many alternative pseudo-schedules with similar costs. This means that the linear programming polyhedra defined by the DWM may have several extreme points lying close to the hyperplane perpendicular to the objective function and touching the optimal face. Therefore, the part of the polyhedra defined by near-optimal extreme points may be "flat" and also have a complex topology, in such a way that the reduced costs give little guidance on how to reach the optimal face.

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 can not assume positive values in any integral solution that improves upon the current best known integral solution. This procedure can be applied at any point of 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 undirect 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.

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

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

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

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

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

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

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 the column generation, $L(\pi_{RM}) = m.Z_{sub} + Z_{RM}$. 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 can not 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:

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

The value of Z_{sub}^{at}, $a = (i,j)$, 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. [11] have, independently of us, proposed a similar fixing procedure for problems where the columns have a path structure. In their work, the computational experiments were performed in the Vehicle Routing Problem with Time Windows.

3.2 Dual Stabilization

Du Merle et al. [7] 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 very 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 along 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 already proposed since the seventies in the context of non-differentiable optimization, as surveyed in [13].

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 [14, 3] strongly recommend on 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 4 parameters to be chosen by dual variable. Moreover, one has to determine when and how the stabilizing functions and center should be updated along 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 4 additional variables (that are never removed) by constraint.

The newly proposed stabilization technique is very simple, there is a single scalar parameter α to be chosen (and this parameter is kept constant along the column generation) and it does not require any change (like additional artificial columns) in the restricted DWM. Let $\bar{\pi}$ be the current best known vector of dual multipliers, i.e., the vector providing the greater lower bound $L(\cdot)$ among all vectors ever evaluated. Let π_{RM} be an optimal dual solution (with value Z_{RM}) of the restricted DWM on some iteration of the column generation. Instead of solving the next pricing subproblem as usual using the π_{RM} values, we propose solving the pricing 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 Column Generation with the New Dual Stabilization

Input: parameter α, $0 < \alpha \leq 1$.

- 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} (after that, if desired, also remove some non-basic variables from it);
  - $\pi_{ST} \leftarrow \alpha.\pi_{RM} + (1-\alpha).\bar{\pi}$;
  - 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 $Z_{RM} - L(\bar{\pi}) < \epsilon$

We now have to prove that Algorithm 1 is correct, i.e., it terminates in a finite number of iterations with an optimal solution of the DWM.

Lemma 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}, 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

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

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

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

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

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

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

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

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

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

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

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

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 implies that a misprice is not a waste of time, quite to the contrary, it is a guarantee that $L(\pi_{ST})$ is a new best lower bound that significantly improves upon $L(\bar{\pi})$. More precisely, it states that a misprice must reduce the gap $Z_{RM} - L(\bar{\pi})$ (the algorithm terminates when this gap is sufficiently small, as discussed next) by at least a factor of $1/(1-\alpha)$. After a misprice, $\bar{\pi} \leftarrow \pi_{ST}$, therefore, although π_{RM} does not change (no column was added to the restricted DWM), the next pricing will be performed using a different π_{ST}.

Theorem 1 *The proposed stabilized column generation algorithm always finds an optimal basic feasible solution for the DWM in a finite number of iterations.*

Proof: Let Z^* be the optimum value for the DWM. Since the DWM has a finite number of basic feasible solutions, Z_{RM} can only assume a finite number of values. Let ϵ be a lower bound for the smallest difference between the value of a non-optimal basic feasible solution and Z^* (such lower bound can be actually computed, for example, by the formula shown in [4], page 375). Let g_1 be the gap $Z_{RM} - L(\bar{\pi})$ after the first restricted DWM LP is solved and the first $L(\bar{\pi})$ is evaluated (for example, if the restricted DWM is initialized with the columns from a known feasible solution with value Z_{INC} and $\bar{\pi}$ is initialized with zero, $g_1 \leq Z_{INC}$). By Lemma 1, the gap is ensured to be less than ϵ after $\lceil \log_{\frac{1}{1-\alpha}} \frac{g_1}{\epsilon} \rceil + 1$ misprices. Thus, $Z_{RM} = Z^*$ after this number of misprices. On the other hand, it can happen that the number of misprices never reaches the previous limit. In this case, the convergence to an optimal solution of the DWM must occur (if all generated columns are kept in the restricted DWM or if some lexicographic cost perturbation is used) after a sufficient number of columns with negative reduced costs are generated. ■

Remark that 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 positive. As the stabilized method does not perform pricings using the vector π_{RM}, it can not prove that optimality. However, since there are no columns with negative reduced cost with respect to π_{RM}, Lemma 1 assures that every subsequent pricing using the vector π_{ST} will reduce this gap by at least a constant factor. Therefore, the gap converges exponentially fast to zero, thus proving the optimality of the restricted DWM solution. Some additional remarks about the proposed stabilization:

- In order to obtain the desired stabilizing effect, the value of α should be small (we used a fixed value of 0.1 in our experiments), so the vector π_{ST} does not deviate much from the current stability center $\bar{\pi}$.

- 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 try the fixing with improving vectors π_{ST} (i.e., those that became the new center of stabilization). In practice this keeps the computational cost of the fixing very low, without a significant impact on the number of variables that are eliminated.

3.3 Hot Starting with the Volume Algorithm

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 advantage in hot starting $\bar{\pi}$ by performing a number of iterations of a Lagrangean multiplier adjustment method. We have chosen the Volume algorithm [2] for that purpose. Some remarks on the use of this algorithm:

- It is essential to also use the previously mentioned fixing by reduced costs while performing the multiplier adjustment iterations. Without the fixing procedure, not only each iteration of the Volume algorithm takes more time, the algorithm also converges into a significantly poorer dual solution.

- The parameters in the Volume algorithm were calibrated in order to obtain a fast convergence to a dual solution $\bar{\pi}$. As the fixing procedure is used, this solution is usually good. Using that $\bar{\pi}$ to hot start the stabilized column generation is quite effective, convergence to an optimal solution of the DWM is usually obtained in a few additional iterations.

- Even when the lower bound $L(\bar{\pi})$ obtained in the end of the Volume algorithm is very close to Z^*, in a branch-cut-and-price context it is still very important to perform the stabilized column generation. This is because the efficiency of the subsequent cut separation round requires the correct fractional solution of the DWM. Separating cuts using an approximated fractional primal solution can lead to the separation of many bad cuts (those that are not violated by the true DWM solution) and prevent the separation of the good ones. The approximated primal solutions provided by the Volume algorithm (at least when it is calibrated for fast convergence) are not good enough for accurate separation.

3.4 Switching to a Branch-and-Cut

The overall algorithm proposed is a branch-cut-and-price, when no more violated cuts are found or when the last rounds of separation show signs of tailing-off (each cut round only improves the node bound marginally), a branch (over the arc-time variables x) is performed. In principle, column and cut generation should be also performed in the child nodes. However, if the fixing by column generation have reduced a lot the number of x variables, we found computational advantage in finishing the solution of those nodes using a branch-and-cut over Formulation (2), restricted to the non-fixed variables, and including all Extended Capacity Cuts that are active in the current restricted DWM. This means that the column generation is not performed anymore. Cut generation, using the same separation routines, is still performed.

The threshold (maximum number of non-fixed variables) for switching to a branch-and-cut was set to 50,000 when $m = 1$ and to 100,000 when $m > 1$.

4 Computational Experiments

Following [15, 5], we performed experiments on the set of 375 instances of the classical $1||\sum w_jT_j$ problem available at the OR-Library. This set was generated by Potts and Wassenhove [21] and contains 125 instances for each $n \in \{40, 50, 100\}$. In fact, for each such n, they created 5 similar instances (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 by 5 similar instances. Those 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 also perform experiments on the $P||\sum w_jT_j$ problem, we derived 100 new instances from those OR-Library instances. For $m \in \{2, 4\}, n \in \{40, 50\}$, we pick the first $1||\sum w_jT_j$ 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 of having idle times between consecutive jobs on the same machine in these tardiness problems. So, we assume that idle times may appear only in the end of the schedule. For the $1||\sum w_jT_j$ problem, T is defined as $\sum_{j=1}^{n} p_j$. For the $P||\sum w_jT_j$ 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.

All our experiments were performed in a notebook with processor Intel Core Duo (but using a single core) with a clock of 1.66GHz and 2GB of RAM. The linear program solver was CPLEX 11.

4.1 Primal Heuristics

Since 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 devised and implemented heuristics for the $1||\sum w_jT_j$ and $P||\sum w_jT_j$ problems, combining known ideas from the literature [10] with a number of new ideas. The description of those heuristics is available in [23], here we only give information about their performance:

- The heuristics were able to find the optimal solutions for all the 375 OR-Library instances of the $1||\sum w_jT_j$ problem. This means that in all such instances the exact algorithm started with an optimal value of Z_{INC}. The time spent by the heuristics is a small fraction (always less than 1%) of the time spent by the exact algorithm.

- The heuristics were able to find the optimal solutions for 93 out of the 100 instances of the $P||\sum w_jT_j$ problem used in our tests. As the exact algorithm could not solve instances wt50-2m-31 and wt50-4m-56, we do not know if their heuristical solutions are optimal or

not. The heuristical solutions for instances wt40-2m-81, wt40-2m-116, wt40-4m-81, wt50-2m-116, and wt50-4m-91 were shown to be suboptimal. In this multi-machine case, the time spent by the heuristics is also a small fraction (always less than 1%) of the time spent by the exact algorithm.

4.2 Computational Results on $1||\sum w_j T_j$ instances

The first experiment aims at showing the importance of the different techniques introduced in Section 3 on solving the DWM problem. Table 1 contains averages over 125 instances, for each $n \in \{40, 50, 100\}$, of five different methods. Method A is the standard column generation, 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 the 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 [9, 27, 16]. Anyway, in this case we report the average time in seconds (*Time (s)*), average number of iterations (*Iter*) and the average integrality gap (*Gap %*). 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 Section 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 the standard column generation, without stabilization, but with the fixing by reduced cost described in Section 3.1 performed at every 20 iterations. Column *R.Arcs* gives the average number of remaining (non-fixed) arcs in the end of the column generation. Remark that the average integrality gap is also reduced, since the fixing may cut a fractional DWM solution. Method D is the stabilized column generation combined with the 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 Finally, Method E is the enhancement of Method D obtained by hot-starting $\bar{\pi}$ using the Volume algorithm (see Section 3.3). Note that the fixing by reduced costs is also called from inside the Volume algorithm. In this case, column *Iter* gives the average number of iterations of the overall method, including Volume iterations and stabilized column generation iterations. The use of the Volume algorithm not only reduces the average times, it also reduces the average number of remaining arcs and the average gaps substantially. It seems that the sequence of improving multipliers $\bar{\pi}$ provided along this algorithm are more effective for the fixing procedure. Methods A, B and C can take a lot of time on some instances with $n = 100$, so we could not compute their statistics over those instances.

Table 2 is a comparison of our complete BCP algorithm with the best algorithm presented by Pan and Shi [15], over all the 375 OR-Library instances. Their best algorithm is a branch-and-bound using not only the time indexed bound (computed in a sophisticated way as a transportation problem), but also other combinatorial bounds. We compare the average time to solve the instance, the maximum time to solve the instance, the average number of nodes in the search tree, the maximum number of nodes and the average integrality gap. The times reported in [15] were obtained using a Pentium IV 2.8 GHz processor machine. It can be seen that our algorithm is a little faster. However, the outstanding difference lies in the root gaps, and therefore, in the number of nodes of the search tree. In fact, as can be seen by also looking at Table 1, the integrality gaps obtained by solving the linear relaxation of the arc-time indexed formulation

Table 1: Comparison of different methods for solving the DWM problem.

	Method	*Time (s)*	*Iter*	*Gap %*	*St. Chgs*	*Misprices*	*R.Arcs*
$n = 40$	A	366.8	819.2	0.0399	–	–	1532996
	B	54.9	116.5	0.0399	83.1	38.3	1532996
	C	44.7	322.0	0.0291	–	–	30
	D	14.5	86.8	0.0224	72.4	32.6	151
	E	12.1	71.7	0.0040	1.9	0.9	2.3
$n = 50$	A	1545.1	1766.2	0.0660	–	–	3007354
	B	146.8	160.1	0.0660	95.4	37.4	3007354
	C	137.8	586.0	0.0557	–	–	241
	D	35.0	118.7	0.0568	85.4	33.2	562
	E	27.6	103.2	0.0249	25.5	5.1	180
$n = 100$	D	673.2	337.6	0.0241	146.2	39.5	5246
	E	386.8	266.8	0.0229	23.4	17.7	4855

are already much smaller than those from the time indexed formulation. The addition of the Extended Capacity Cuts reduces this gap to zero in almost all instances. The branching was performed in only six instances [1], all of them with $n = 100$.

Table 2: Comparison of the complete BCP algorithm with the best algorithm by Pan and Shi [15].

n	*Alg.*	*Avg. Time (s)*	*Max Time (s)*	*Avg. Nd*	*Max Nd*	*Avg. Root Gap %*
40	[15]	69.0	235	141	293	0.68
	BCP	12.1	43.6	1	1	0
50	[15]	142.8	232	416	5623	0.74
	BCP	28.1	123.8	1	1	0
100	[15]	1811	32400	18877	> 909844	0.52
	BCP	648.5	8508	2.03	42	0.0013

Table 3 presents detailed results of the BCP over a sample of 25 instances, those with numbers ending with the digit 1 or 6, with $n = 100$. Bigras, Gamache and Savard [5] also chose to present detailed results on these 25 instances. It should be noted that 4 of these instances are trivial, in the sense that they have a solution of value 0 that can be easily found by the heuristic in less than a milisecond (deciding if a $1||\sum w_j T_j$ instance has a solution of value 0 or not can be done in polynomial time [19]). Since such solution must be optimal, the BCP is not run on such cases. The first set of columns give information about the Volume algorithm: the lower bound obtained, the number of iterations, the running time, and the number of remaining (non-fixed) arcs in the end of the algorithm. The optimality of 12 of these instances can be proved only by the Volume lower bound. The next set of columns gives information about the stabilized column generation used to actually find an optimal solution of the DWM. Again, we report the DWM lower bound, the number of column generation iterations, the column generation time and the number of remaining arcs. The third set of columns gives information about the work

[1] An additional experiment has shown that a more aggressive separation of Extended Capacity Cuts would solve all the 375 instances without ever branching, but this more than doubles the running time of the BCP for some instances with $n = 100$.

performed in the root node of the BCP after cuts start to be added. We report the final root node bound, the number of cut rounds, the time spent by this part of the code and the number of remaining arcs. More 10 instances in this sample have their optimality proven still in the root node. The last two columns are the total number of nodes in the search tree, including nodes where column generation is not performed (see Section 3.4) and the total running time.

Table 3: Detailed results of the complete BCP algorithm over a sample of 25 OR-Library instances with $n = 100$.

	Volume				1st.LP				Remaining Root Node				Total		
Inst	*LB*	*Iter*	*Time*	*R.Arcs*	*LB*	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Opt*
1	5988	1	77.3	0	–	–	–	–	–	–	–	–	1	77.3	5988
6	58258	19	93.2	0	–	–	–	–	–	–	–	–	1	93.2	58258
11	181649	28	136.1	0	–	–	–	–	–	–	–	–	1	136.1	181649
16	407703	160	279.9	0	–	–	–	–	–	–	–	–	1	279.9	407703
21	898925	125	212.7	0	–	–	–	–	–	–	–	–	1	212.7	898925
26	8	1	69.4	0	–	–	–	–	–	–	–	–	1	69.4	8
31	24202	20	90.2	0	–	–	–	–	–	–	–	–	1	90.2	24202
36	108293	93	209.3	0	–	–	–	–	–	–	–	–	1	209.3	108293
41	462117	401	922.7	13093	462123	227	120.1	10596	462324	1	24.3	0	1	1067.1	462324
46	829771	340	498.0	1583	829773	115	67.5	1375	829828	1	14.6	0	1	580.1	829828
51	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
56	9046	20	76.6	0	–	–	–	–	–	–	–	–	1	76.6	9046
61	86793	227	480.4	0	–	–	–	–	–	–	–	–	1	480.4	86793
66	243637	555	1164.7	33525	243644	382	119.3	30002	243822	9	1088.1	20807	30	2807.0	243872
71	640799	351	735.1	1178	640802	113	103.2	775	640816	1	16.0	0	1	854.3	640816
76	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
81	1400	30	80.2	0	–	–	–	–	–	–	–	–	1	80.2	1400
86	66850	186	463.5	0	–	–	–	–	–	–	–	–	1	463.5	66850
91	248284	401	1027.7	77260	248293	428	152.6	64425	248699	4	3089.8	0	1	4270.1	248699
96	495358	376	918.7	18853	495362	275	104.7	15994	495516	2	50.0	0	1	1085.6	495516
101	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
106	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
111	158962	454	1554.2	29457	158968	337	143.6	25652	159123	2	1369.2	0	1	3067.0	159123
116	370435	445	1288.2	40265	370451	354	176.4	34754	370614	2	1250.3	0	1	2714.9	370614
121	471166	392	957.4	4024	471175	235	144.2	2626	471214	1	15.6	0	1	1117.2	471214

4.3 Computational Results on $P||\sum w_j T_j$ instances

Table 4 gives statistics on the integrality gaps provided by different methods over each set of 25 instances, for $m \in \{2, 4\}, n \in \{40, 50\}$. The first set of columns gives results obtained by solving the linear relaxation of the time indexed formulation (1) using CPLEX: average integrality gap (*Avg. Gap %*), maximum integrality gap (*Max Gap %*) and average time in seconds (*Time (s)*). The second set of columns shows results obtained by solving the DWM corresponding to the arc-time indexed formulation, the third the results of the same formulation after the addition of some rounds of Extended Capacity Cuts. It can be seen that, unlike in the single machine instances, the lower bounds obtained by the arc-time indexed formulation were not much stronger than the one by the time indexed formulation. By the way, in instance wt40-2m-81, both methods provided a bound that was more than 20% below the optimal. However, the Extended Capacity Cuts proved to be very effective in reducing the integrality gap of the arc-time indexed formulation (these cuts can not be introduced in the time indexed formulation). This is crucial for the effectivity of the BCP algorithm, most instances from this set can not be solved without the cuts. We also remark that the column showing the time that CPLEX used for solving the time indexed formulation was only included in Table 4 for the sake of curiosity. It is not fair to compare it with the time to solve the arc-time indexed formulation using our column generation algorithm, since a column generation similar to the one in [5] would probably be more efficient in obtaining the time indexed bound.

Finally, tables 5 to 8 give detailed results of the complete BCP algorithm over all multi-machine instances. The columns in these tables are analogous to those in Table 5; excepting that, since the Volume algorithm is not used, the corresponding columns do not exist. It can be seen that the BCP algorithm could solve 98 out of 100 instances to optimality. The solved instance that required more time was wt50-4m-6, it took 48385 seconds.

Table 4: Comparison of different bounding methods for multi-machine instances.

		Time indexed Relaxation			Arc-Time Relaxation			Arc-Time + Cuts		
n	m	*Avg. Gap %*	*Max Gap %*	*Time (s)*	*Avg. Gap %*	*Max Gap %*	*Time (s)*	*Avg. Gap %*	*Max Gap %*	*Time (s)*
40	2	1.533	21.016	85.6	1.243	20.840	32.2	0.053	0.853	295.9
	4	0.544	4.787	32.2	0.406	3.390	14.1	0.105	0.841	63.9
50	2	0.535	4.074	182.2	0.487	4.074	88.3	0.078	1.051	2298.7
	4	0.529	5.614	79.5	0.489	5.614	36.8	0.266	5.088	262.5

Table 5: Detailed results of the complete BCP algorithm over the instances with $m = 2$ and $n = 40$.

	1st.LP				Remaining Root Node				Total		
Inst	*LB*	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Opt*
1	584	89	18.0	156948	606	7	325.4	0	1	343.4	606
6	3875	141	25.5	82838	3886	3	119.6	0	1	145.1	3886
11	9592	189	34.5	66999	9617	2	94.6	0	1	129.1	9617
16	38279	292	45.2	59225	38351	2	515.8	59225	3	561.0	38356
21	41048	384	37.1	0	–	–	–	–	1	37.1	41048
26	87	48	12.7	0	–	–	–	–	1	12.7	87
31	3758	172	34.0	106263	3812	5	452.0	0	1	486.0	3812
36	10662	303	44.4	52812	10700	2	1113.6	52812	5	1193.6	10713
41	30802	387	46.4	0	–	–	–	–	1	46.4	30802
46	34146	430	29.8	0	–	–	–	–	1	29.8	34146
51	–	–	–	–	–	–	–	–	–	0	0
56	1272	80	16.5	107098	1279	2	72.3	0	1	88.8	1279
61	11311	269	45.3	72238	11390	2	1754.2	72238	327	9097.3	11488
66	35130	323	51.9	75499	35196	2	1503.6	75499	196	6451.1	35279
71	47935	423	42.9	42430	47952	2	19.5	0	1	62.4	47952
76	–	–	–	–	–	–	–	–	–	0	0
81	452	150	20.6	71423	571	2	947.2	0	1	967.8	571
86	5996	302	40.4	47829	6041	2	253.5	47829	6	298.0	6048
91	26075	388	56.6	0	–	–	–	–	1	56.6	26075
96	66110	358	50.9	46481	66116	2	2.9	0	1	53.8	66116
101	–	–	–	–	–	–	–	–	–	0	0
106	–	–	–	–	–	–	–	–	–	0	0
111	17898	292	50.0	51884	17936	2	46.5	0	1	96.5	17936
116	25786	317	50.4	54574	25870	2	173.7	0	1	224.1	25870
121	64507	390	50.9	48152	64516	2	3.0	0	1	53.9	64516

Table 6: Detailed results of the complete BCP algorithm over the instances with $m = 4$ and $n = 40$.

	1st.LP				Remaining Root Node				Total		
Inst	*LB*	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Opt*
1	438	65	9.8	73227	439	2	25.1	0	1	34.9	439
6	2372	108	12.7	30946	2374	2	6.7	0	1	19.4	2374
11	5735	157	18.1	30100	5737	2	4.4	0	1	22.5	5737
16	21484	184	19.0	26703	21493	2	1.6	0	1	20.6	21493
21	22793	248	18.4	0	–	–	–	–	1	18.4	22793
26	88	37	6.3	0	–	–	–	–	1	6.3	88
31	2496	106	15.4	51528	2511	2	600.5	51528	22	2702.4	2525
36	6355	214	21.4	37543	6366	2	121.7	37543	2875	15727.4	6420
41	17634	185	20.9	38062	17642	2	91.0	38062	129	673.3	17685
46	19124	186	12.8	0	–	–	–	–	1	12.8	19124
51	–	–	–	–	–	–	–	–	–	0	0
56	798	64	9.4	76075	826	2	486.0	0	1	495.4	826
61	7316	170	19.8	32934	7321	2	68.9	32934	490	1711.7	7357
66	20247	227	21.8	25041	20251	2	0.8	0	1	22.6	20251
71	26740	179	15.4	0	–	–	–	–	1	15.4	26740
76	–	–	–	–	–	–	–	–	–	0	0
81	550	106	10.6	27399	564	2	16.2	0	1	26.8	564
86	4719	178	17.3	22971	4725	2	0.6	0	1	17.9	4725
91	15557	211	23.8	25525	15569	2	8.2	0	1	32.0	15569
96	36266	159	18.6	0	–	–	–	–	1	18.6	36266
101	–	–	–	–	–	–	–	–	–	0	0
106	–	–	–	–	–	–	–	–	–	0	0
111	11212	147	19.3	35245	11225	2	71.0	35245	126	442.9	11263
116	15539	175	20.7	28340	15545	2	59.2	28340	93	164.6	15566
121	35739	198	20.6	29012	35742	2	35.8	29012	19	68.2	35751

Table 7: Detailed results of the complete BCP algorithm over the instances with $m = 2$ and $n = 50$.

	1st.LP				Remaining Root Node				Total		
Inst	*LB*	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Opt*
1	1232	110	46.1	359070	1268	6	634.1	0	1	680.2	1268
6	14261	263	96.1	149062	14269	17	34893.7	142741	4	34989.8	14272
11	23000	391	107	97813	23028	6	488.2	0	1	595.2	23028
16	46011	506	111	81432	46072	5	157.5	0	1	268.5	46072
21	111067	711	109.7	72348	111069	2	13.1	0	1	122.8	111069
26	26	61	25.7	0	–	–	–	–	1	25.7	26
31	5289	265	90.5	229566	5348	15	7806.9	197307	–	–	$\leq$ 5378
36	18895	401	115.3	95689	18956	10	656.9	0	1	772.2	18956
41	37968	466	118.2	96739	38058	5	186.7	0	1	304.9	38058
46	82087	596	107	79936	82105	6	100.1	0	1	207.1	82105
51	–	–	–	–	–	–	–	–	–	0	0
56	730	114	38.1	264965	759	11	1002.3	232073	1	1040.4	761
61	13589	403	125.1	104020	13628	20	4908.9	99990	74	12323.7	13682
66	40904	498	100.7	54008	40907	2	4.0	0	1	104.7	40907
71	78532	637	133.8	0	–	–	–	–	1	133.8	78532
76	–	–	–	–	–	–	–	–	–	0	0
81	538	115	37.9	184060	542	1	103.0	0	1	140.9	542
86	12277	520	137.2	106811	12425	15	2666.3	99296	1258	30742.6	12557
91	47294	573	155.2	94353	47328	17	1083.4	94353	20	1425	47349
96	92803	603	152.6	77182	92822	4	51.7	0	1	204.3	92822
101	–	–	–	–	–	–	–	–	–	0	0
106	–	–	–	–	–	–	–	–	–	0	0
111	15544	514	128.3	74027	15564	4	44.8	0	1	173.2	15564
116	19524	522	132.2	85783	19574	22	2665.9	85783	151	9877	19608
121	41696	725	138.9	0	–	–	–	–	1	138.9	41696

Table 8: Detailed results of the complete BCP algorithm over the instances with $m = 4$ and $n = 50$.

	1st.LP				Remaining Root Node				Total		
Inst	*LB*	*Iter*	*Time*	*R.Arcs*	*LB*	*CutR*	*Time*	*R.Arcs*	*Nd*	*Time*	*Opt*
1	777	85	27.0	238283	785	2	62.5	0	1	89.5	785
6	8298	150	42.9	87158	8309	12	2680.2	87158	95	48385.6	8317
11	12871	195	40.8	52183	12875	10	372.4	52183	8	484.4	12879
16	25375	277	45.2	34528	25376	2	1.0	0	1	46.2	25376
21	59440	352	46.1	0	–	–	–	–	1	46.1	59440
26	54	52	13.8	0	–	–	–	–	1	13.8	54
31	3061	163	39.6	0	–	–	–	–	1	39.6	3061
36	10794	321	53.8	38681	10796	2	1.4	0	1	55.2	10796
41	21783	312	57.3	50836	21787	10	136.2	50836	69	845.1	21806
46	44452	304	46.1	38591	44455	2	1.6	0	1	47.7	44455
51	–	–	–	–	–	–	–	–	–	0	0
56	538	85	18.7	182659	541	5	2206.0	181333	–	–	$\leq$ 570
61	7850	300	59.3	57025	7858	10	305.3	57025	2720	43645.9	7898
66	23138	329	45.9	0	–	–	–	–	1	45.9	23138
71	42625	283	49.6	50117	42629	9	186.9	50117	27	328.1	42645
76	–	–	–	–	–	–	–	–	–	0	0
81	478	84	17.8	125105	495	3	93.4	0	1	111.2	495
86	8330	242	47.1	41757	8335	10	81.8	41757	58	274	8369
91	26546	339	61.0	40882	26551	3	2.8	0	1	63.8	26551
96	50312	258	51.9	44559	50317	10	117.0	44559	43	246.9	50326
101	–	–	–	–	–	–	–	–	–	0	0
106	–	–	–	–	–	–	–	–	–	0	0
111	10049	267	52.1	41325	10053	8	71.9	41325	36	139.7	10069
116	11520	321	54.8	46912	11523	9	152.3	46912	3315	18277.5	11552
121	23769	281	48.2	45560	23775	9	90.5	45560	179	898.8	23792

5 Conclusions

This paper presented a set of algorithms, bundled into a branch-cut-and-price code, for some important scheduling problems. Very good experimental results are shown. The $1||\sum w_j T_j$ instances that just a few years ago were considered as untractable can now be solved quickly and almost without any branching. Instances of the $P||\sum w_j T_j$ problem of reasonable size can also be solved. The paper original contributions include:

- An arc-time indexed formulation for scheduling problems that is shown to be significantly stronger than the time indexed formulation, specially for the single-machine case.
- A procedure to fix variables by reduced costs that can be applied on any column generation or Lagrangean relaxation algorithm where the subproblem has a path structure. As already mentioned, a similar procedure was independently proposed in [11]. It is experimentally shown that this procedure is crucial for working with the arc-time indexed formulation.
- A very simple, effective and theoretically sound dual stabilization procedure. This procedure is general and can be used on any column generation algorithm.

Acknowledgements. AP, EU and MPA received support from CNPq grants 301175/06-3, 304533/02-5, and 300475/93-4, respectively. RR was supported by grant PICDT/CAPES.

References

[1] P. Avella, M. Boccia, and B. D'Auria. Near-optimal solutions of large scale single-machine scheduling problems. *ORSA Journal on Computing*, 17:183–191, 2005.

[2] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient algorithm. *Mathematical Programming*, 87:385–399, 1999.

[3] H. Ben Amor, A. Frangioni, and J. Desrosiers. On the choice of explicit stabilizing terms in column generation. Technical Report G-2007-109, GERAD, Montreal, December 2007.

[4] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[5] L. Bigras, M. Gamache, and G. Savard. Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing*, 1:133–142, 2008.

[6] S. Dash, R. Fukasawa, and O. Gunluk. On the generalized master knapsack polyhedron. In *Proceedings of the 12th IPCO Conference*, volume 4513 of *Lecture Notes in Computer Science*, pages 197–209, 2007.

[7] du Merle, O. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

[8] M. Dyer and L. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26:255–270, 1990.

[9] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106:491–511, 2006.

[10] A. Grosso, F. Della Croce, and R. Tadei. An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, 32:68–72, 2004.

[11] S. Irnich, G. Desaulniers, J. Desrosiers, and A. Hadjar. Path reduced costs for eliminating arcs. Technical Report G-2007-83, GERAD, Montreal, November 2007.

[12] E. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Research Letters*, 1:207–208, 1977.

[13] C. Lemaréchal. Lagrangean relaxation. In M. Juenger and D. Naddef, editors, *Computational Combinatorial Optimization*, pages 115–160. Springer, 2001.

[14] A. Oukil, H. Ben Amor, , J. Desrosiers, and H. El Gueddari. Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems. *Computers & Operations Research*, 34:817–834, 2007.

[15] Y. Pan and L. Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single machine scheduling problems. *Mathematical Programming*, 110:543–559, 2007.

[16] A. Pessoa, M. Poggi de Aragão, and E. Uchoa. Robust branch-cut-and-price algorithms for vehicle routing problems. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008. In press.

[17] A. Pessoa, E. Uchoa, and M. Poggi de Aragão. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. *Networks*, 2008. To appear.

[18] J. Picard and M. Queyranne. The time-dependant traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26:86–110, 1978.

[19] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice-Hall, 2002.

[20] M. Poggi de Aragão and E. Uchoa. Integer program reformulation for robust branch-and-cut-and-price. In L. Wolsey, editor, *Annals of Mathematical Programming in Rio*, pages 56–61, Búzios, Brazil, 2003.

[21] C. Potts and L. Wassenhove. A branch-and-bound algorithm for the total weighted tardiness problem. *Operations Research*, 32:363–377, 1985.

[22] M. Queyranne and A. Schulz. Polyhedral approaches to machine scheduling. Technical Report 408, University of Berlin, 1997.

[23] R. Rodrigues, A. Pessoa, E. Uchoa, and M. Poggi de Aragão. Heuristics for single and multi-machine weighted tardiness problems. Technical Report RPEP Vol.8 no.8, Universidade Federal Fluminense, Engenharia de Produção, Niterói, Brazil, 2008.

[24] R. Sadykov. *Integer Programming-based Decomposition Approaches for Solving Machine Scheduling Problems*. PhD thesis, Universite Catholique de Louvain, 2006.

[25] J. Sousa and L. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, 54:353–367, 1990.

[26] E. Uchoa. Robust branch-and-cut-and-price for the CMST problem and extended capacity cuts. Presentation in the MIP 2005 Workshop, Minneapolis, 2005. Available at http://www.ima.umn.edu/matter/W7.25-29.05/activities/Uchoa-Eduardo/cmst-ecc-IMA.pdf.

[27] E. Uchoa, R. Fukasawa, J. Lysgaard, A. Pessoa, M. Poggi de Aragão, and D. Andrade. Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Mathematical Programming*, 122:443–472, 2008.

[28] J. Van der Akker, C. Hurkens, and M. Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[29] J. Van der Akker, C. Van Hoesel, and M. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, 85:541–572, 1999.