An Improved Branch-Cut-and-Price algorithm for Parallel Machine Scheduling Problems

Daniel Dias de Oliveira Neto
Universidade Federal Fluminense
Rua Passo da Pátria 156, 22210-240, Niterói, RJ
daniel.oliveira@id.uff.br

Artur Alves Pessoa
Universidade Federal Fluminense
Rua Passo da Pátria 156, 22210-240, Niterói, RJ
artur@producao.uff.br

RESUMO
Esse trabalho apresenta um algoritmo de Branch-Cut-and-Price para o problema de escalonamento de tarefas em máquinas paralelas minimizando uma função custo genérica baseada nos tempos de término. Uma nova família de desigualdades é proposta para fortalecer a formulação arco-tempo-indexada, em conjunto com um eficiente algoritmo de separação. Além disso, a projeção da formulação arco-tempo-indexada para a tempo-indexada é introduzida para se aproveitar da fixação de variáveis realizada no espaço extendido de variáveis. O algoritmo melhorado foi capaz de resolver 143 de 150 instâncias da literatura, sendo 9 resolvidas pela primeira vez. Em adição, o tempo de resolução de 134 instâncias que já haviam sido resolvidas previamente diminuiu em 84,1% na média.

PALAVRAS CHAVE. escalonamento de tarefas em máquinas paralelas, programação inteira, geração de colunas, branch-cut-and-price, formulações tempo-indexadas.

Tópicos: Programação Matemática

ABSTRACT
This work presents an improved Branch-Cut-and-Price algorithm for the identical parallel machine scheduling problem minimizing a generic function of the job completion times. A new family of cuts is proposed to strengthen the arc-time-indexed formulation, along with an efficient separation algorithm. Also, the projection of the arc-time-indexed into a time-indexed formulation is introduced to take advantage of the variable fixations performed in the larger variable space. The improved algorithm was capable of solving 143 out of the 150 literature instances, being 9 solved for the first time. Also, the running time for the 134 previously solved instances decreased by 84.1% on the average.

KEYWORDS. parallel machine scheduling, integer programming, column generation, branch-cut-and-price, time-indexed formulations.

Paper topics: Mathematical Programming
1. Introduction

In the scheduling problem denoted by $P||\sum f_j(C_j)$, a set $J$ of $n$ jobs have to be processed by a set of $m$ identical parallel machines. Each job has a processing time $p_j$, and is associated with a cost function $f_j(C_j)$ based on its completion time, $C_j$. Each machine can only process one job at a time and each job has to be processed by a single machine, with no pauses or preemption. The goal is to find a schedule that minimizes the sum of individual costs. A classical special case of this problem is the weighted tardiness variant ($P||\sum w_jT_j$), where each job has a weight $w_j$, a due date $d_j$, and the cost function is $f_j(C_j) = w_jT_j$, being $T_j$ the job tardiness, calculated as $\max\{0, C_j - d_j\}$.

Potts e Wassenhove [1985] proposed a branch-and-bound algorithm for the single machine total weighted tardiness problem ($1||\sum w_jT_j$) capable of solving instances of up to 40 jobs. Its lower bound was based on Lagrangian relaxation, solved by a multiplier adjustment method. The superiority of the algorithm at the time was highlighted by a review of 6 exact algorithms for the $1||\sum w_jT_j$ problem, presented by Abdul-Razaq et al. [1990], which included computational tests.

Tanaka et al. [2009] presented an efficient algorithm for solving the $1||\sum f_j(C_j)$ when machine idle times are not permitted. Their algorithm is based on the work of Ibaraki e Nakamura [1994], which used the Successive Sublimation Dynamic Programming (SSDP) to solve $1||\sum f_j(C_j)$. The SSDP, proposed by Ibaraki [1987], is based on a set of dynamic programming relaxations for a problem. It consists of solving a sequence of dynamic programming relaxations, stepping from the weaker to the stronger, tightening the gap in the process. Also, when stepping from one model to the next, states that can be proved not to be part of an optimal solution are eliminated to alleviate memory use and speed up computations.

The dynamic programming relaxations can be derived in many different ways. The ones used by Ibaraki e Nakamura [1994] were the state-space relaxations presented by Abdul-Razaq e Potts [1988]. It is interesting to note that the SSDP method uses dynamic programming in a different way than most exact algorithms, which usually employ the Lagrangian bounds on a Branch and Bound framework. As Tanaka et al. [2009] stated, their algorithm is based fully on dynamic programming.

Recently, Tanaka e Fujikuma [2012] extended Tanaka et al. [2009] to permit machine idle times. Following the same SSDP framework, Tanaka e Araki [2013] presented an algorithm for the single machine total weighted tardiness problem with sequence-dependent setup times, $1|s_{i,j}|\sum w_jT_j$.

Pessoa et al. [2010] were the first to propose an exact algorithm for the same class of parallel machine scheduling problems addressed in this paper, but focusing on the weighted tardiness objective function. The explicit use of the arc-time-indexed formulation for a scheduling problem, where each variable is indexed by a pair of jobs and a time, was a novelty of their work. To handle the huge number of variables of such formulation, column generation was used along with a series of advanced techniques, composing a Branch-Cut-and-Price (BCP) algorithm. Some important highlights of their work were the use of a strong family of cuts, variable fixation by Lagrangean bounds, dual stabilization, and the use of a commercial MIP solver to finish the optimization when the number of remaining variables is sufficiently small. Throughout this paper, their algorithm is referred to as BCP-PMWT, where PMWT stands for parallel machine with weighted tardiness.

In this work, the following techniques were developed in order to improve the BCP-PMWT.

- A new family of cuts, called Overload Elimination Cuts (OECs), for the arc-time-indexed formulation, that improves the quality of the relaxation used by the BCP-PMWT. As the number of constraints potentially generated by this family is exponential, they are also added by demand.
• A genetic algorithm to separate OECs during the BCP execution.

• The use of job-time indexed, or simply time-indexed for shortness, formulations instead of arc-time-indexed to finish the optimization when the number of remaining variables is small enough. Experiments using four alternative formulations are presented. The time-indexed formulations are more compact, allowing for the commercial solver to expand the Branch-and-Bound tree much faster. However, the relaxations based on these formulation are weaker, reducing the algorithm ability to prune the tree.

• The addition of cuts to the time-indexed formulation used in the previous item. These cuts are derived from the arc-time-indexed formulation taking into account the fixed variables.

The improved method is called as BCP-PMWT-OTI, where OTI stands for overload, referring to the new family of cuts, and time-indexed, due to the formulation used to finish the optimization.

The rest of the paper is organized as follows. Section 2 reviews the main characteristics of the BCP-PMWT. Section 3 presents the proposed improvements that composes the BCP-PMWT-OTI. Section 4 details how computational experiments were performed and analyses its results. Section 5 summarizes our conclusions, highlighting our main results.

2. The Original Algorithm

In this section, we give an overview of the BCP-PMWT algorithm, including all details necessary to explain the proposed improvements. For further details, we refer to [Pessoa et al. 2010].

2.1. The Arc-Time-indexed Formulation

The formulation used in the BCP-PMWT is the arc-time-indexed formulation (ATIF), which uses a binary variable for each job pair \((i,j)\) and time period \(t\) to indicate that job \(i\) finishes and job \(j\) starts at time \(t\), on the same machine. Considering \(T\) as the latest time a job can finish in an optimal schedule, defining \(J_0 = J \cup \{0\}\) and \(p_0 = 0\), the formulation follows:

\[
\text{Minimize} \quad \sum_{i \in J_0} \sum_{j \in J \setminus \{i\}} \left( \sum_{t = p_i}^{T - p_j} f_j (t + p_j) x_{ij}^t \right) \tag{1a}
\]

\[
\text{Subject to} \quad \sum_{i \in J_0 \setminus \{j\}} \sum_{t = p_i}^{T - p_j} x_{ij}^t = 1 \quad (j \in J) \tag{1b}
\]

\[
\sum_{j \in J_0 \setminus \{i\}} \sum_{t = p_j}^{T - p_i} x_{ij}^t - \sum_{j \in J_0 \setminus \{i\}} \sum_{t = p_i + p_j}^{T} x_{ij}^{t + p_i} = 0 \quad (\forall i \in J; t = 0, \ldots, T - p_i) \tag{1c}
\]

\[
\sum_{j \in J_0} \sum_{t = p_j}^{T} x_{j0}^t - \sum_{j \in J_0} \sum_{t = p_j + 1}^{T} x_{0j}^{t + p_j} = 0 \quad (t = 0, \ldots, T - 1) \tag{1d}
\]

\[
\sum_{j \in J_0} x_{0j}^t = m \quad (t = 0, \ldots, T - 1) \tag{1e}
\]

\[
x_{ij}^t \in Z^+ \quad (\forall i \in J_0; \forall j \in J_0 \setminus \{i\}; t = p_i, \ldots, T - p_j) \tag{1f}
\]

\[
x_{00}^t \in Z^+ \quad (t = 0, \ldots, T - 1). \tag{1g}
\]

The objective function (1a) is defined as the sum of the completion costs for all jobs. Constraints (1b) impose that every job \(j\) starts exactly once. Constraints (1c) and (1d) are flow conservation constraints. Constraint (1e) requires that \(m\) flows, one per machine schedule, start on time 0. Constraints (1f) and (1g) enforce integrality.
Each ATIF solution can be seen as a set of paths in a graph $G = (V, A)$, where $V = \{ (i, t) \mid i \in J_0, t \in \{0, 1, 2, \ldots, T-1\} \} \cup \{ (0, T) \}$ and each arc $a^t = ((i, t - p_i), (j, t)) \in A$ ($a = (i, j)$) corresponds to a $x^t_{ij}$ variable. Figure 1 represents a possible solution, for an instance with 5 jobs and 2 machines, as such graph, where $p_1 = p_3 = 5$, $p_2 = 4$, $p_4 = 3$, and $p_5 = 2$.

### 2.2. ATIF Reformulation

Following the idea represented in figure 1, a pseudo-schedule is defined as a path from $(0, 0)$ to $(0, T)$ in $G$, possibly repeating jobs. Each pseudo-schedule represents the part of the schedule to be processed on one machine in a fractional solution to (1a)-(1e) (with $x^t_{ij} \geq 0; \forall i,j,t$).

Let $P$ be the set of all possible pseudo-schedules. For every pseudo-schedule $p \in P$, define a variable $\lambda_p$ and a set of constants $\{q^p_{ij} \mid a^t \in A\}$ to indicate if an arc $a^t$ appears in $p$. Define $f_0(t)$ as zero for any $t$. The Dantzig-Wolfe Master (DWM) LP is written as:

\[
\text{Minimize } \sum_{p \in P} \left( \sum_{(i,j) \in A} q^p_{ij} f_j(t + p_i) \right) \lambda_p \\
\text{Subject to } \sum_{p \in P} \left( \sum_{(j,i) \in A} q^p_{ji} \right) \lambda_p = 1 \quad (\forall i \in J) \\
\sum_{p \in P} \left( \sum_{(0,j) \in A} q^p_{0j} \right) \lambda_p = m \quad (\forall p \in P), \\
\lambda_p \geq 0 \quad (\forall p \in P).
\]

Since the number of $\lambda$ variables is exponential on $n$, in order to solve DWM, these variables are generated on demand. For that, given an optimal solution to (2) restricted to a subset of the $\lambda$ variables, and its dual, the pricing subproblem consists of finding the variable $\lambda_p$ with minimum reduced cost. If such a reduced cost is negative, then $\lambda_p$ is added to the restricted master problem and the process continues. Otherwise, the current solution is also optimal for the complete DWM.

To efficiently compute the optimal $\lambda_p$ variable, its reduced cost is expressed as the sum of the reduced costs of the arcs of $p$, which are defined in the following.

Suppose that, at a given instant, there are $r+1$ constraints in the DWM. Let $\pi_0$ be the dual variable of constraint (2c), $\pi_i$ be the dual variable of constraints (2b) for $i \in J$, and $\pi_l, n < l \leq r$, be the dual variable of any additional constraint. Being $\alpha^t_{ai}$ the coefficient of variable $x^t_{ij}$ in constraint...
l, the reduced cost of arc $a^l$ is defined using the $\alpha$ as:

$$\bar{c}_a^l = f_j(t + p_j) - \sum_{l=0}^r \alpha_a^l \pi_l.$$  (3)

2.3. Pricing and Fixing by Reduced Costs

The pricing subproblem in the BCP-PMWT algorithm consists of finding the pseudo-schedule $p$ with the minimum reduced cost for its corresponding variable $\lambda_p$. This can be done by finding the shortest path from $(0, 0)$ to $(0, T)$ in the previously defined graph $G$, setting the length of each arc $a^l$ to $\bar{c}_a^l$.

Let $\bar{c}^*$ be the reduced cost of a shortest path from $(0, 0)$ to $(0, T)$ in $G$, when the current objective value for the DWM is $LB$ and the best known upper bound on the optimal solution cost is $UB$. If the shortest path from $(0, 0)$ to $(0, T)$ that uses a given arc $a^l$ has a reduced cost $\hat{c}_a^l$ such that $LB + (m - 1)\bar{c}^* + \hat{c}_a^l > UB - 1$, then one can conclude that no solution better than $UB$ may use this arc. Thus, it can be fixed to zero.

2.4. Extended Capacity Cuts

One family of cuts is separated to further strengthen the continuous relaxation of the ATIF. Let $S \subseteq J$ be a set of jobs, define $p(S) = \sum_{j \in S} p_j$ as the total processing time of $S$, $\delta^-(S) = \{(i, j)^t \in A : i \notin S, j \in S\}$ and $\delta^+(S) = \{(i, j)^t \in A : i \in S, j \notin S\}$. Equation (4) below is valid.

$$\sum_{a^l \in \delta^+(S)} tx_a^l - \sum_{a^l \in \delta^-(S)} tx_a^l = p(S)$$

(4)

The Rounded Homogeneous Extended Capacity Cuts (RHECCs), inequality (5), is obtained by multiplying (4) by a value $r \in (0, 1)$ and applying integer rounding.

$$\sum_{a^l \in \delta^+(S)} \lceil rt \rceil x_a^l - \sum_{a^l \in \delta^-(S)} \lfloor rt \rfloor x_a^l \geq \lceil r p(S) \rceil$$

(5)

To separate RHECCs, a specific heuristic procedure is used.

2.5. Branch-cut-and-price

The algorithm consists of a Branch-cut-and-price where the continuous relaxation of the ATIF strengthened by RHECCs is solved in each node, using column generation stabilized by the technique of Wentges [1997].

After every 5 column generation iterations, variable fixing by reduced costs is performed. This helps to reduce the number of arcs in $A$, therefore speeding the pricing and improving convergence. The relaxation bounds may also be improved.

When the number of arcs in $A$ after solving the root node is below 200,000, additionally to the BCP, the current reduced ATIF is fed to a commercial MIP solver (CPLEX 11.1).

3. The Improved Algorithm

In this section we propose improvements to the BCP-PMWT algorithm, resulting in the algorithm referred to as the BCP-PMWT-OTI. We present a new family of cuts, along with its separation algorithm. Next, we propose that the existing procedure of feeding a residual model to a MIP solver uses a time-indexed formulation. We then study different time-indexed formulations to be used in such procedure, and show how they can be strengthened by information gathered from the arc-time-indexed formulation.
3.1. Overload Elimination Cuts

Define the aggregated variables \(v^t\) and \(u^t\), for a given set \(S \subseteq J\), as follows:

\[
v^t = \sum_{a^t \in \delta^+(S)} x^t_a \quad (t = 1, \ldots, T),
\]

\[
u^t = \sum_{a^t \in \delta^-(S)} x^t_a \quad (t = 0, \ldots, T - 1).
\]

For \(m \geq 2\), a subset of jobs \(S \subseteq J\), and \(t \in \left\{ 1, \ldots, \left\lfloor \frac{p(S) - 1}{m} \right\rfloor + 1 \right\}\), we have the following inequality, which we call the overload elimination constraint (OEC):

\[
\sum_{q=t}^{t_1} v^q + \sum_{q=t_1+1}^{T} 2v^q - \sum_{q=\max\{t_1, T-p(S)+m(t-1)+1\}}^{T-1} u^q \geq 2,
\]

\(t_1 = p(S) - t - (m - 2)(t - 1)\).

**Theorem 1.** The OEC is valid for the ATIF.

**Proof.** To be included in coming paper.

3.1.1. Separation

For the separation of OECs, we implemented a genetic algorithm. Here, the term solution and individual will be used interchangeably to denote a cut, being it violated or not. Each OEC is fully described by the elements of \(S\) and the \(t\) value. Given a subset \(S\) of jobs, the \(t\) value that leads to the best cut can always be found by testing all possibilities. Thus, the separation procedure described in the following focus on defining which subsets will be tested.

Define \(\bar{G} = (\bar{V}, \bar{E})\) as an (undirected) support graph for the fractional solution \(\bar{x}\), such as \(\bar{V} = J\) and \(\bar{E} = \{(i, j) : \exists t | \bar{x}^t_{ij} > 0\text{ or }\bar{x}^t_{ji} > 0\}\). Also, define \(\bar{C}\) as the average completion time for jobs \(j \in J\), calculated as

\[
\bar{C}_j = \sum_{(i,t) | \bar{x}^t_{ji} > 0} t \bar{x}^t_{ji},
\]

\(\bar{F}(k)\) as the set of jobs with the \(k\) smallest values of \(\bar{C}\), \(j(k)\) as the job \(j\) with the \(k\)-th smallest \(\bar{C}_j\), \(\bar{G}(k)\) as the subgraph of \(\bar{G}\) induced by \(\bar{F}(k)\), and \(\bar{S}(k)\) as the connected component of \(\bar{G}(k)\) that includes \(j(k)\). In a fractional schedule, the set \(\bar{F}(k)\) can be interpreted as the first \(k\) jobs to finish, and \(j(k)\) as the \(k\)-th job to finish.

The genetic algorithm starts by generating an initial population of \(n\) solutions. Then, a number of crossover and selection operations are performed iteratively until the stopping criterion is met. For every new individual, being generated either by the initial population or the crossover operation, local search is performed to improved it. The algorithm parameters are the size of the population to be carried from one generation to the next (\(n\)Population), the crossover rate (\(crossoverRate\)), which is the percentage of the \(n\)Population to be used as the number of individuals generated at each iteration by crossover, and the criterion to stop the algorithm (\(stopCriterion\)).

As did Uchoa et al. [2006] for separating RHECCs, we require that the set \(S\) of every generated cut induces a connected subgraph in \(\bar{G}\). However, we allow that induced subgraphs become disconnected during the local search. For shortness, in this section, we refer to sets with this property as connected \(S\) sets.

The crossover operation consists in deriving a new solution (say child) from two solutions (say father and mother) randomly chosen from the population. To generate the child, we first select
all elements that are in both parents. If this intersection is empty, we include one random element from each parent, along with the smallest subset of nodes that yields a connected $S$. Then, each element contained in the parents is included in $S$ at random, with probability $0.5$. If the resulting $S$ is not connected, we select a random subcomponent of $S$ and discard the remaining nodes.

The local search operator, consists of performing every single element insertion and deletion on $S$ until no movement yields a better cut. For every change, the violation is calculated for every valid value of $t$ and we pick the one generating the greater violation, which is considered as minus the constraint slack if it is not violated.

The selection operation consists of eliminating the worst individuals so that the remaining population has exactly $nPopulation$ individuals. However, a pool of violated cuts (including the ones found during a local search) is kept apart from the current population. No cut is removed from this pool. No mutation operator is used. Based on preliminary tests we set $nPopulation$ to 20, $crossoverRate$ to 100%, and $stopCriterion$ to the execution of 100 generations.

### 3.2. Triangle Clique Cuts

Following the approach of Pessoa et al. [2009], the triangle clique cuts are used here to further strengthen the ATIF:

Given a set $S \subset J$, with exactly three elements, let $G = (\mathcal{V}, \mathcal{E})$ be the compatibility graph where each vertex in $\mathcal{V}$ represents an arc $a^t = (i, j)^t \in A$, $i, j \in S$ and each edge $e = (a^t_1, a^t_2)$ belongs to $\mathcal{E}$ if and only if $a^t_1$ and $a^t_2$ are compatible. For each $i, j, k \in S$, $e \in \mathcal{E}$ if $e = ((i, j)^t_1, (j, k)^t_2)$ and $t_2 = t_1 + p_j$. For any independent set $I \subset \mathcal{V}$, a triangle clique constraint (TCC) is defined by the valid inequality

$$\sum_{a^t \in I} x^t_a \leq 1.$$  \hspace{1cm} (10)

Given $\bar{x}$, the separation algorithm consists in building the compatibility graph $G$ for every triple of jobs $(i, j, k)$, and finding the maximum-weight independent set. As noted by Pessoa et al. [2009], $G$ is a set of chains, and by preliminary weight computation we observed that chains of four or more elements don’t occur frequently in the subgraph of $G$ induced by the variables with non-zero values. So, in order to use a faster and simpler separation algorithm, we only consider the first three vertices of each chain and search for a violated TCC by enumerating each possible independent set.

### 3.3. Switching to a MIP Solver

Dyer e Wolsey [1990] proposed a MIP formulation for scheduling problems using one binary variable $y^t_j$ for each job-time to indicate that a job $j$ completes at time $t$. A time period $t$ is defined as spawning from instant $t - 1$ to instant $t$. This formulation is referred to as the time-indexed formulation (TIF). For the multi machines case, the TIF follows:

Minimize

$$\sum_{j \in J} \sum_{t = p_j}^T f_j(t) y^t_j$$  \hspace{1cm} (11a)

Subject to

$$\sum_{t = p_j}^T y^t_j = 1 \quad (j \in J)$$  \hspace{1cm} (11b)

$$\sum_{j \in J} \min\{t + p_j - 1, T\} \sum_{s = \max\{p_j, t\}}^{t + p_j} y^s_j \leq m \quad (t = 1, \ldots, T)$$  \hspace{1cm} (11c)

$$y^t_j \in \{0, 1\} \quad (j \in J; \ t = p_j, \ldots, T) \hspace{1cm} (11d)$$

The objective function (11a) is defined as the sum of the completion cost for all jobs. Constraints (11b) ensure that each job completes once. Constraints (11c) limit the number of jobs on execution to at most $m$ at any time period $t$. Constraints (11d) enforce integrality.
As performed in the BCP-PMWT, when the solved root node is still fractional, as an alternative to performing branching, we will also feed the residual model to a MIP solver. The model is called residual since some variables may have been fixed to zero by the variable fixation procedure. But instead of feeding the residual ATIF, we project it to the TIF and feed this formulation. In the next section, alternative TIF’s to be used in this procedure are described.

When projecting the ATIF, the bounds provided by the RHECCs, TCCs and OECs are lost, since such cuts are not translated to the TIF. Also, for a TIF variable \(y^t_j\) to be fixed to zero, every variable \(x^t_{ij}\) \((i \in J_0)\) needs to be fixed to zero, which weakens the fixation. To mitigate this, in Subsection 3.5 we describe a procedure to enhance the TIF linear relaxation bounds, based on the performed ATIF variable fixations.

3.4. Alternative Time-Indexed Formulations

When investigating different formulations for a problem, the usual purpose is to find the one that yields the best bounds. Here, we explore different ways of describing the exactly same polyhedron and discuss how they may influence the MIP solver. We believe two characteristics of a formulation can improve the MIP solver performance.

First is how sparse the constraint matrix is. The sparser, the faster are the bound computations. Second is the impact of fixing one variable over the linear relaxation bound. The more balanced this impact is, the better the MIP solver performs when selecting the branching variables.

When modeling scheduling problems, there are basically two characteristics to be constrained, one is that every job has to be processed, and the other is that no more than \(m\) machines are active at any moment. Two ways of modeling the first characteristic are:

1. Use binary variables \(y^t_j\) indicating that job \(j\) has finished at time \(t\), enforcing for every job \(j\) that \(\sum_{t=p_j}^{T} y^t_j = 1\).

2. Use binary variables \(z^t_j\) indicating that job \(j\) has finished until time \(t\), enforcing for every job \(j\) that \(z^{p_j-1}_j = 0, z^T_j = 1, \text{ and } z^{t-1}_j \leq z^t_j (t = p_j, \ldots, T)\).

Two ways of modeling the second characteristic are:

1. Formulate the problem as a network flow, where \(m\) flow units leave the source, and flow is conserved.

2. Formulate the problem by constraining the number of active machines at any given moment.

By combining these alternatives, four different formulations are yielded. The first two formulations, (12) and (11), use binary variables \(y^t_j\), and differ in how the second characteristic is enforced. The first does it by a network flow (12c, 12d) and the second does it by resource constraints (11c).

\[
\text{Minimize} \quad \sum_{j \in J} \sum_{t=p_j}^{T} f_j(t) y^t_j \quad (12a)
\]

\[
\text{Subject to} \quad \sum_{t=p_j}^{T} y^t_j = 1 \quad (j \in J) \quad (12b)
\]

\[
\sum_{j \in J} y^p_j = m \quad (12c)
\]

\[
\sum_{i \in J | t \geq p_i} y_i^t \geq \sum_{j \in J} y_j^{t+p_j} \quad (t = 1, \ldots, T) \quad (12d)
\]

\[
y_j^t \in \{0, 1\} \quad (j \in J; \; t = p_j, \ldots, T) \quad (12e)
\]
The next two formulations, \((13)\) and \((14)\), use binary variables \(z_j^t\), and again, differ in how the second characteristic is enforced. The first does it by a network flow \((13b)\)\((13c)\) and the second does it by resource constraints \((14b)\).

The main difference of these two formulations is how fixing one variable influences the other variables. For example, by setting \(z_j^{t_1} = 1\), every \(z_j^t\) with \(t > t_1\) are also set to 1, and, by setting \(z_j^{t_1} = 0\), every \(z_j^t\) with \(t < t_1\) are also set to 0. This leads to a more effective branching, since fixing some variable often changes the relaxed solution substantially for both children nodes.

\[
\text{Minimize } \sum_{j \in J} \sum_{t=p_j}^{T} f_j(t) \left( z_j^t - z_j^{t-1} \right) \tag{13a}
\]

\[
\text{Subject to } \sum_{j \in J} \left( z_j^{p_j} - z_j^{p_j-1} \right) = m \tag{13b}
\]

\[
\sum_{j \in J, t \geq p_j} \left( z_j^t - z_j^{t-1} \right) \geq \sum_{j \in J} \left( z_j^{t+p_j} - z_j^{t+p_j-1} \right) \quad (t = 1, \ldots, T) \tag{13c}
\]

\[
z_j^{t-1} \leq z_j^t \quad (j \in J; \ t = p_j, \ldots, T) \tag{13d}
\]

\[
z_j^{p_j-1} = 0 \quad (j \in J) \tag{13e}
\]

\[
z_j^t \in \{0, 1\} \quad (j \in J; \ t = p_j, \ldots, T - 1) \tag{13f}
\]

\[
z_j^T = 1 \quad (j \in J) \tag{13g}
\]

\[
\text{Minimize } \sum_{j \in J} \sum_{t=p_j}^{T} f_j(t) \left( z_j^t - z_j^{t-1} \right) \tag{14a}
\]

\[
\text{Subject to } \sum_{j \in J} \left( z_j^{p_j(t+p_j-1,T)} - z_j^{t-1} \right) \leq m \quad (t = 1, \ldots, T) \tag{14b}
\]

\[
z_j^{t-1} \leq z_j^t \quad (j \in J; \ t = p_j, \ldots, T) \tag{14c}
\]

\[
z_j^{p_j(t-1)} = 0 \quad (j \in J) \tag{14d}
\]

\[
z_j^t \in \{0, 1\} \quad (j \in J; \ t = p_j, \ldots, T - 1) \tag{14e}
\]

\[
z_j^T = 1 \quad (j \in J) \tag{14f}
\]

### 3.5. TIF cuts by projecting the ATIF Polytope

We devise a procedure to generate cuts from the projection of the ATIF to the TIF. These cuts take further advantage of the variable fixing performed by BCP-PMWT over the ATIF variables.

We define a fractional solution of the TIF as \(y^*\) and the set of remaining arcs \(A_t\) from the ATIF, for each time period \(t\), as below.

\[
A_t = \{(i,j)^t \in A \mid x_{ij}^t\text{ is not fixed}\} \tag{15}
\]

From the network flow time-indexed formulation \((12)\), inequality \((12d)\) can be interpreted as the association of some variable \(y_{ij}^{t+p_j}\) equal to one to some variable \(y_i^t\) also equal to one, meaning that some job \(j\) is the successor of job some job \(i\). Since, by the ATIF (and the variable fixing), not all such successions are allowed, the following inequalities must be valid for the TIF,

\[
\sum_{i:j \in S(i,j)^t \in A_t} y_{ij}^t \geq \sum_{j \in S} y_{j}^{t+p_j} \quad (S \subseteq J; \ t = 1, \ldots, T). \tag{16}
\]
For a given time period $t$, exact separation of (16) can be done by finding the minimum cut on a digraph built as follows. We lay one source node ($s$) and one sink node ($t$), and $n$ nodes $\{1, 2, ..., n\}$ and other $n$ nodes $\{1', 2', ..., n'\}$. For each variable $y^s_i \neq 0$, we draw an arc connecting the source node to the node $i$ with capacity $c_{si} = y^s_i$. For each variable $y^{s+t+p}_j \neq 0$, we draw an arc connecting the node $j'$ to the sink with capacity $c_{j't} = y^{s+t+p}_j$. Finally, for each non-fixed variable $x_{ij}$ we draw an arc from $i$ to $j'$ with capacity $c_{ij'} = \infty$. After calculating the minimum cut, the set $S$ will be all nodes $i'$ in the sink side of the cut. Figure 2 shows how the graph is constructed.

This procedure can be viewed as a way to check if a solution in the $y$ variables space is feasible in the $x$ variables space. For shortness, we refer to the cuts proposed here as projected cuts.

4. Experimental Results

In this section, we detail how computational tests were performed and comment its results. First, we evaluate the effectiveness of the Triangle Clique and Overload Elimination Cuts on improving the ATIF bounds. Next, we feed the 40 and 50 jobs instances residual model to a MIP solver, using all four time-indexed formulations, in order to evaluate which performs better. Last, we analyze the results of the full BCP-PWMT-OTI algorithm.

The $P||\sum w_j T_j$ instances were generated by transforming the $1||\sum w_j T_j$ instances of [Potts and Wassenhove, 1985], found on the OR-Library. As described by [Pessoa et al., 2010], for $n \in \{40, 50, 100\}$, and $m \in \{2, 4\}$, the first instance for each group $(n, m)$ was picked (instance numbers ending with 1 or 6) and had its due dates $d_j$ divided by $m$.

For the primal bounds, we used the solutions computed by the heuristic procedure of [Kramer and Subramanian, 2015]. For better evaluating our improvements to the BCP algorithm, we highlight where those values are better than the ones of [Rodrigues et al., 2008], used in the BCP-PMWT. Tests are run only on instances not proven optimal by the ATIF linear relaxation, i.e., the first linear relaxation solution is less than the primal bound.

The LP/MIP solver used was the IBM ILOG CPLEX 12.5. All tests ran on a Intel Core i7-3770 PC with a 3.4Ghz clock (using one thread), 12GB of RAM, and the Linux operating system.

4.1. Solving the Root Node

Table 1 summarizes by instance set $(m,n)$ the results of both algorithms in solving the root node. The integrality gap is calculated as $(UB - Root LB) / UB$. By root lower bound, we refer to the objective value of the LP after all rounds of cut separation. A cut separation round consist of separating the RHECCs, TCCs and OECs, at once, for a fractional solution. At most 50 cuts from each family are inserted in each round. It can be seen that the increase in lower bound compensates the time expense of separating OEC cuts. Especially for instances with $n = 100$ and $m = 2$.

4.2. Evaluating the best alternative Time-Indexed Formulation

We ran tests for each of the four time-indexed formulations presented in Section 3.4 in order to compare them. To save time, we constrained the tests to 40 and 50 jobs instances. But we believe that any conclusions drawn can be assumed true for bigger instances. For all formulations, we used the residual model after the root node of the BCP-PWMT-OTI.
Table 1: Root relaxation and cut separation results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>2</td>
<td>0.525%</td>
<td>0.235%</td>
<td>78.0</td>
<td>51.9</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>0.456%</td>
<td>0.448%</td>
<td>23.4</td>
<td>18.8</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>0.379%</td>
<td>0.276%</td>
<td>256.8</td>
<td>193.8</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>0.571%</td>
<td>0.583%</td>
<td>67.8</td>
<td>29.9</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>0.878%</td>
<td>0.114%</td>
<td>6297.0</td>
<td>3398.8</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>0.494%</td>
<td>0.322%</td>
<td>984.0</td>
<td>481.6</td>
</tr>
</tbody>
</table>

Table 2: Comparison of Alternative Time-Indexed Formulations – Summary

<table>
<thead>
<tr>
<th>n</th>
<th>Fy</th>
<th>Ry</th>
<th>Fz</th>
<th>Rz</th>
<th>Average LP Time (s)</th>
<th>Average MIP Time (s)</th>
<th># Solved</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.72</td>
<td>0.84</td>
<td>7.17</td>
<td>0.97</td>
<td>63.17</td>
<td>351.97</td>
<td>12</td>
</tr>
<tr>
<td>50</td>
<td>1.77</td>
<td>1.98</td>
<td>47.08</td>
<td>2.43</td>
<td>53.46</td>
<td>150.26</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 2 summarizes the results by giving the average time for solving the linear relaxation plus all cut generations (Average LP Time), the average time the MIP solver took to solve the TIF formulation strengthened by projected cuts (Average MIP Time), as well as the number of instances solved in up to 3,600 seconds (# Solved). In it, we refer to formulations (12), (11), (13) and (14) as Fy, Ry, Fz and Rz respectively. The averages consider only the 10 instances solved by all four formulation. It can be observed that the best choice is formulation Rz, even though solving the linear relaxation sometimes may be faster for a different TIF, the branching performed by the MIP solver is more effective for Rz.

4.3. The Full Improved Algorithm

Table 3 summarizes the results for all instance sets (m,n). All 40 and 50 jobs instances were already solved, but the better lower bounds and the TIF improved running times. Three instances of the (100,2) set and five instances of the (100,4) set were solved for the first time. Also, there was an improvement of running time. Each set (m,n) have 25 instances, the instances proven optimal by the first ATIF LP relaxation are omitted from the table, but are accounted in the number of Solved instances.

5. Conclusions

This work proposed a set of improvements to the BCP algorithm of [Pessoa et al., 2010], referred to as BCP-PMWT throughout the text. The resulting algorithm, referred to as BCP-PMWT-OTI, included a new family of cuts, the Overload Elimination Cuts, with an efficient algorithm for

Table 3: Full Results - Summary

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>BCP-PMWT # Solved</th>
<th>BCP-PMWT Avg. Time</th>
<th>BCP-PMWT-OTI # Solved</th>
<th>BCP-PMWT-OTI Avg. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>50</td>
<td>357.9</td>
<td>50</td>
<td>48.1</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>5734.9</td>
<td>50</td>
<td>241.9</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>18</td>
<td>22523.8</td>
<td>21</td>
<td>7058.5</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>16</td>
<td>37667.7</td>
<td>22</td>
<td>5672.0</td>
<td></td>
</tr>
</tbody>
</table>
separation. Also, we projected the Arc-Time-Indexed formulation into the Time-Indexed formulation for solving the residual model with a MIP solver, which we consider to be the main contribution of this work. For the instances that could be solved both by feeding the ATIF and the TIF residual model to the MIP solver, the average MIP solution time decreases 92.7% by using the latter, even though its bounds are worse.

For the $P$||$\sum w_j T_j$ instances that could be solved by both algorithms, the proposed improvements resulted in an average solution time decrease of 84.1%, solving 9 instances for the first time, leaving 7 instances still unsolved.

References


