A flow-based formulation for parallel machine scheduling using decision diagrams

Daniel Kowalczyk, Roel Leus
ORSTAT, Faculty of Economics and Business, KU Leuven, Leuven, Belgium; daniel.kowalczyk@kuleuven.be, roel.leus@kuleuven.be

Christopher Hojny
Combinatorial Optimization Group, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands; c.hojny@tue.nl

Stefan Røpke
Department of Technology, Management, and Economics, Technical University of Denmark, Lyngby, Denmark; ropke@dtu.dk

We present a new flow-based formulation for identical parallel machine scheduling with a regular objective function and without idle time. The formulation is constructed with the help of a decision diagram that represents all job sequences that respect specific ordering rules. These rules rely on a partition of the planning horizon into, generally non-uniform, periods and do not exclude all optimal solutions, but they constrain solutions to adhere to a canonical form. The new formulation has numerous variables and constraints, and hence we apply a Dantzig-Wolfe decomposition in order to compute the linear programming relaxation in reasonable time; the resulting lower bound is stronger than the bound from the classical time-indexed formulation. We develop a branch-and-price framework that solves three instances from the literature for the first time. We compare the new formulation with the time-indexed and arc-time-indexed formulation by means of a series of computational experiments.

Key words: parallel machine scheduling, weighted tardiness, column generation, decision diagrams

1. Introduction

We study a scheduling problem where a set $J = \{1, \dots, n\}$ of n jobs with processing time $p_j \in \mathbb{N} \setminus \{0\}$ for each $j \in J$ needs to be processed by a set $M = \{1, \dots, m\}$ of m identical parallel machines without pre-emption. The problem is to find an assignment of jobs to machines and a sequence of the jobs on each machine such that some objective function $\sum_{j \in J} f_j(C_j)$ is minimized, where C_j is the completion time of $j \in J$. We focus on parallel machine scheduling with a *regular* objective function, i.e., for which f_j is non-decreasing for all $j \in J$. Moreover, we require that there exists an optimal solution without idle time between the jobs on each machine. Scheduling with weighted completion-time objective $Pm||\sum w_j C_j$ is such a problem, where each job j has a weight w_j. For this objective the sequencing on each machine is easy (there exists a canonical sequence that can be followed), so that the difficulty only resides in finding an optimal division of the jobs over

the machines. Another common scheduling objective is the weighted tardiness, where each job j also has a due date d_j and the cost $f_j(C_j)$ associated with job j is w_jT_j with $T_j = \max\{0, C_j - d_j\}$. Since the one-machine case $1||\sum w_jT_j$ is strongly NP-hard (Lawler, 1977), unless P = NP there is no canonical order of jobs on a machine such that the weighted tardiness scheduling problem could reduce to partitioning jobs over the machines, and a different approach is needed. In the remainder of the paper, we only consider this weighted tardiness objective, but the developed method will be generic and other regular objective functions without idle time can be treated analogously.

The main goal of this paper is to introduce a new flow-based formulation for $Pm||\sum w_jT_j$. We first discuss some related work in Section 2. Our formulation is based on a time discretization of the planning horizon that was introduced by Baptiste and Sadykov (2009), which we summarize in Section 3. To the best of our knowledge, we are the first to apply a formulation with a coarser time discretization than the classical time-indexed formulation (TIF) to a parallel machine scheduling problem. The formulation itself is presented in Section 4, and is derived from a binary decision diagram (BDD) that represents all the possible job sequences on a machine. We show that the LP relaxation of this new integer linear formulation yields a stronger lower bound than the TIF.

Our new formulation has many variables and constraints, which renders the computation of the LP bound inefficient; we apply a Dantzig-Wolfe (DW) decomposition to resolve this issue. This reformulation is discussed in Section 5. We also need to overcome some convergence problems in the column generation (CG) phase, which is achieved using the stabilization technique of Wentges (1997) and by variable fixing by reduced cost as described in Pessoa et al. (2010). The running times of the CG phase for the new formulation are much lower than those for the arc-time-indexed formulation (ATIF), which was introduced independently by Sourd (2009), Pessoa et al. (2010), and Tanaka et al. (2009). At the same time, we find the quality of the lower bounds from the new formulation to be very similar to that of the ATIF in our experiments.

In Section 6 we develop a branch-and-price (B&P) algorithm to find optimal integer solutions, in which we use an aggressive strong branching strategy to establish optimality of primal solutions. We report on a series of computational experiments in Section 7, including a comparison with the current state-of-the-art procedure of Oliveira and Pessoa (2020). We conclude the paper in Section 8.

2. Related work

The most popular exact methods for single and parallel machine scheduling use Dynamic Programming (DP), Branch-and-Bound (B&B) including Mixed-Integer Programming (MIP) formulations that are solved by a solver, or a mix of those two techniques. The most popular MIP formulations in the literature are based on completion variables, (arc-) time-indexed variables, linear ordering variables, and positional and assignment variables. Below, we discuss formulations with time-indexed and arc-time-indexed variables. For an extensive introduction to other formulations, we refer to Queyranne and Schulz (1994).

2.1. Time-indexed formulation TIF

The TIF has been thorougly studied by, among others, Dyer and Wolsey (1990), Sousa and Wolsey (1992), and van den Akker et al. (1999b). With integer processing times p_j, a sufficiently large planning horizon T can be discretized into periods of unit length. Binary variables y_{jt} are defined for each job $j \in J$ and each period $t \in \{1, \dots, T\}$ to decide whether job j starts at the beginning of period t or not, where period t starts at time $t-1$ and ends at t. The model can be used to represent many different single and parallel machine scheduling problems (esp. with min-sum objective) by adjusting the cost parameters $\widetilde{c}_{jt}$.

$$\text{minimize} \sum_{j \in J} \sum_{t=1}^{T-p_j+1} \widetilde{c}_{jt} y_{jt} \tag{1a}$$

$$\text{subject to} \sum_{t=1}^{T-p_j+1} y_{jt} = 1 \qquad \forall j \in J \tag{1b}$$

$$\sum_{j \in J} \sum_{s=\max\{1,t-p_j+1\}}^{t} y_{js} \leq m \qquad \forall t \in \{1, \dots, T\} \tag{1c}$$

$$y_{jt} \in \{0,1\} \qquad \forall j \in J,\, t \in \{1, \dots, T\} \tag{1d}$$

With Constraints (1b) we ensure that every job starts exactly once, while Constraints (1c) impose that at most m jobs can be processed in any period. Extra constraints such as release times r_j for each job $j \in J$ can be easily modeled by deleting the variables for which $t \in \{1, \dots, r_j\}$. Solutions to Constraints (1c) and (1d) can be represented as a flow in a directed acyclic graph (DAG) where the nodes are associated to the starting period t of the jobs and the edges $(t, t+p_j)$ are associated to a job j that starts in period t and ends in period $t+p_j-1$. A unit flow from the root node (first period) to the terminal

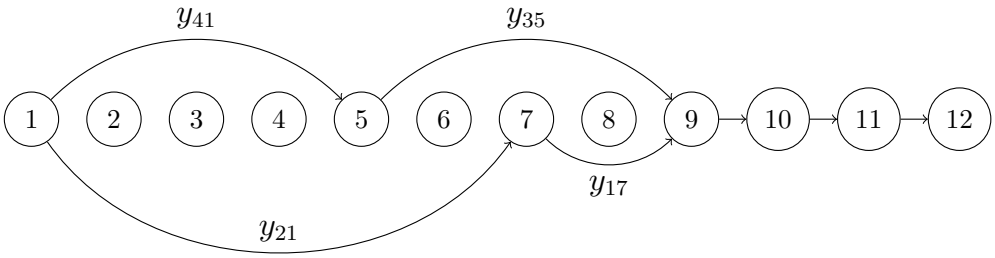


Figure 1 **An integral solution to the TIF represented as two paths in the DAG**

Table 1 **Job data for the example instance**

job j	p_j	d_j	w_j
1	2	4	6
2	6	6	3
3	4	8	2
4	4	8	5

node (last period) is called a *pseudo-schedule*. A flow satisfies the last two constraint sets ((1c) and (1d)) but not necessarily the assignment constraints (1b) and hence there can be pseudo-schedules where two or more edges associated to the same job are chosen. In Figure 1 we provide an optimal integral solution to an instance with $n = 4$ and $m = 2$, and with the job data given in Table 1. For problem $Pm||\sum w_j T_j$ the time horizon T can be chosen as $\lceil(\sum_{j\in J} p_j - p_{\max})/m\rceil + p_{\max}$ without losing all optimal solutions, where $p_{\max}$ is the maximum processing time (see, for instance, Pessoa et al., 2010). In this way we obtain $T = 11$ as a safe upper bound for the time horizon of the instance.

The TIF is known to have a strong LP bound (Dyer and Wolsey, 1990), but this strength comes at a cost: the length of the planning horizon is pseudo-polynomial in the size of the instance input, i.e., the number of constraints and variables depends on the number of jobs and on the processing times. Hence, the TIF is less applicable for instances with many jobs and large processing times, and one cannot always compute the LP relaxation in a reasonable amount of time. Many specialized techniques have been developed to overcome this issue; van den Akker et al. (2000), for instance, use CG to compute the LP relaxation of the TIF. Even using CG, solving the LP relaxation can be slow, because the CG phase can suffer from the heading-in effect (when the first iterations of the CG produce irrelevant columns and bad dual bounds because of the bad dual information), and extreme degeneracy (multiple optimal solutions in the dual and hence the solution of

the restricted master remains constant over several iterations). To cope with this problem, various approaches were proposed; we refer to Bigras et al. (2008), Pan and Shi (2007), Sadykov and Vanderbeck (2013), and Pessoa et al. (2018). We also note that the TIF can still leave a large duality gap and hence exact algorithms may need to explore a large B&B tree. Many polyhedral studies of the TIF were therefore performed, see for instance Crama and Spieksma (1996), Sousa and Wolsey (1992), and van den Akker et al. (1999b).

2.2. Arc-time-indexed formulation ATIF

The ATIF can be seen as an extended formulation of the TIF. The number of variables is a factor of n larger than in TIF. Let $x_{ij}^t \in \{0,1\}$ be variables for each pair of jobs $i, j \in J_+$, with $i \neq j$, $J_+ = \{0,1,\ldots,n\}$ and $p_0 = 0$, and each $t \in \{0,\ldots,T\}$. The variables x_{ij}^t indicate whether or not job i completes and job j starts at time t on some machine. Let $\overline{c}_{jt}$ be the cost of starting job j at time t (so $\overline{c}_{jt} = \widetilde{c}_{j,t+1}$).

$$\text{minimize} \sum_{i \in J_+} \sum_{j \in J \setminus \{i\}} \sum_{t=p_i}^{T-p_j} \overline{c}_{jt} x_{ij}^t \tag{2a}$$

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

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

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

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

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

$$x_{00}^t \in \mathbb{N} \qquad \forall t \in \{0,\ldots,T-1\} \tag{2g}$$

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

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

model a network flow of m units over a layered DAG. Since each flow over this network has the same source and destination node, we can decompose any integral solution into a

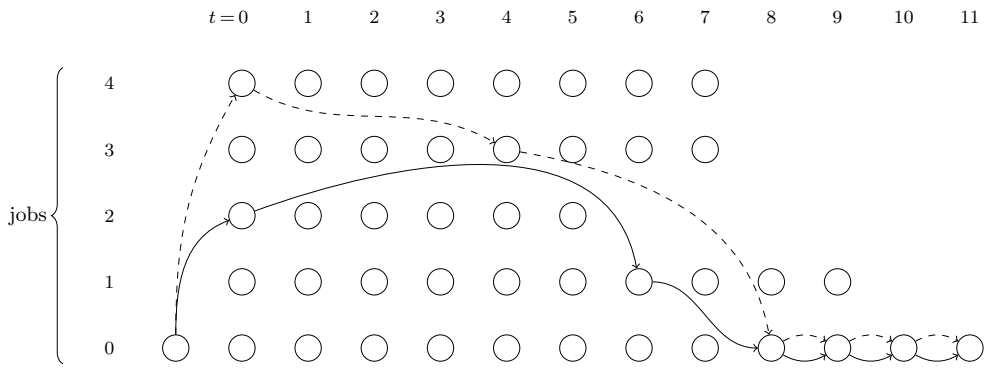


Figure 2 **Example of an integral solution of the ATIF represented as paths in the DAG, where each edge from job i to job j that arrives in column t corresponds to variable x_{ij}^t**

set of m paths that correspond to pseudo-schedules. Constraint (2d) models the possibility of idle time in the solution. With Constraint (2b) we impose that each job has to be visited by exactly one path and as a result, each job is assigned to exactly one machine. Sourd (2009), Tanaka et al. (2009), and Pessoa et al. (2010) proposed this formulation independently. Pessoa et al. (2010) develop a branch-cut-and-price algorithm to solve the ATIF, so as to handle the large number of variables, and point out that the ATIF is almost isomorphic to the arc-capacity formulation for the Vehicle Routing Problem and hence many inequalities for the latter formulation could be transposed. Pessoa et al. show that these valid inequalities can reduce the gap between a heuristic solution and the LP bound provided by the relaxation of the ATIF. They also show that the ATIF is stronger than the TIF, which mainly stems from the fact that direct repetitions of jobs are forbidden by excluding variables x_{jj}^t. One can easily project every solution $\bar{x}$ of the linear relaxation of ATIF onto a solution $\bar{y}$ of the linear relaxation of TIF by setting $\bar{y}_{jt} = \sum_{i \in J_+ \setminus \{j\}} \bar{x}_{ij}^{t-1}$ for $j \in J$ and $t \in \{1, \ldots, T - p_j + 1\}$. Moreover, simple dominance rules can be applied to the DAG by omitting the variables x_{ij}^t if permuting jobs i and j at time t decreases the overall cost. Oliveira and Pessoa (2020) continued and refined the work of Pessoa et al. (2010), and their procedure constitutes the current state-of-the-art benchmark for $Pm||\sum w_j T_j$.

An example of the network for the ATIF corresponding to the instance described in Table 1 is given in Figure 2. The paths in this solution correspond to schedules $(2, 1, 0, 0, 0)$ and $(4, 3, 0, 0, 0)$ on one machine, where each 0 stands for a unit of idle time. In terms of the variables x_{ij}^t of the ATIF, this solution corresponds with $x_{02}^0 = x_{04}^0 = x_{43}^4 = x_{21}^6 = x_{10}^8 = x_{30}^8 = 1$, $x_{00}^9 = x_{00}^{10} = x_{00}^{11} = 2$, and the other variables equal 0.

3. Discretization of the time horizon

To cope with the large number of variables in formulations based on a discretization of the time horizon as encountered by TIF and ATIF, one can resort to a coarser discretization of time. This is the underlying idea behind the models that were proposed by Baptiste and Sadykov (2009) and Boland et al. (2016) for single machine scheduling.

Boland et al. (2016) introduced the bucket-indexed formulation (BIF). Like the TIF, this formulation partitions the planning horizon into periods of equal lengths but the length of the periods in the BIF is a parameter, and can be as long as the processing time of the shortest job. The BIF is equivalent to the TIF if the length of the shortest job is one, but the number of variables can reduce significantly if the length is larger than 1. For a good comparison between the BIF and TIF for single machine scheduling, we refer to Boland et al. (2016).

Another formulation that uses a coarser time discretization is the *interval-indexed model*, which was introduced for single machine scheduling by Baptiste and Sadykov (2009), who partition the planning horizon into time intervals that are defined by a proper superset of the release dates, due dates, and deadlines of all jobs. Baptiste and Sadykov show that there exists an optimal schedule where the jobs assigned to a time interval are sequenced according to a modified weighted shortest-processing-time rule. In the remainder of this section, we describe the results from Baptiste and Sadykov (2009) in more detail. All definitions in this section are borrowed from this reference.

Let a *partition* $\mathcal{I}$ of order q of the time horizon be a set of time intervals $I_r = (e_{r-1}, e_r]$ with $e_0 = 0$, $e_q = T$, and $r \in Q = \{1, \ldots, q\}$. A partition is *based on due dates* if every d_j equals some e_r, i.e., $\{d_1, \ldots, d_n\} \subseteq \{e_r\}_{r \in Q}$. A job j is assigned to interval I_r if its completion time C_j is in I_r. A job j is *on time* in interval I_r if $d_j \geq e_r$ and *late* if $d_j \leq e_{r-1}$.

Next, we define for each interval I_r of $\mathcal{I}$ a permutation σ_r of $\{1, \ldots, n\}$, and let σ be the set of all permutations σ_r with $r \in Q$. We say that σ is an *appropriate* set of permutations for $\mathcal{I}$ if there exists an optimal schedule in which, for any interval $I_r \in \mathcal{I}$ and any two jobs $i, j \in J$ assigned to the same machine and the same interval I_r, job i is sequenced before job j when $\sigma_r(i) < \sigma_r(j)$. For the problem $Pm||\sum w_j C_j$, for example, there exists a partition of order 1 with $I_1 = (0, T]$, where the appropriate permutation σ_1 corresponds to *Smith's rule*, i.e., the jobs are sequenced in non-increasing order of the ratios $\frac{w_j}{p_j}$. Other problems with a similar priority rule are $Pm||\sum w_j U_j$ and $Pm||\sum w_j V_j$, where function U_j

indicates whether job j is late or not, while $V_j = \min\{p_j, \max\{0, C_j - d_j\}\}$ represents the portion of work of job j that is performed after its due date (see van den Akker et al., 1999a).

We say that a partition $\mathcal{I}$ is *appropriate* if it is possible to compute an appropriate set of permutations for the partition in polynomial time. Baptiste and Sadykov (2009) show how to find an appropriate partition for a large number of single machine problems, and the same approach can be applied to parallel machines because we need canonical sequences on each machine separately. We follow Baptiste and Sadykov (2009) for the construction of an appropriate partition of the time horizon. Since we consider $Pm||\sum w_j T_j$, our partition is based on the due dates d_j. Denote by σ a set of permutations for such a partition. We will make a distinction between *long* and *short* jobs of an interval I_r. A job j is long in interval I_r if $p_j \geq e_r - e_{r-1}$, and short if $p_j < e_r - e_{r-1}$. We demand that all long jobs of interval I_r appear first in σ_r, meaning that if $p_i \geq e_r - e_{r-1}$ and $p_j < e_r - e_{r-1}$ then $\sigma_r(i) < \sigma_r(j)$; note that at most one long job can effectively be assigned to I_r.

Baptiste and Sadykov (2009) devise the following set of rules. For each pair of short jobs i, j of interval I_r we require:

- if job i is late in I_r and job j is on time in I_r then $\sigma_r(i) < \sigma_r(j)$,
- if jobs i and j are on time and $p_i > p_j$ then $\sigma_r(i) < \sigma_r(j)$,
- if jobs i and j are late and $\frac{p_i}{w_i} < \frac{p_j}{w_j}$ then $\sigma_r(i) < \sigma_r(j)$,
- if jobs i and j are late, $\frac{p_i}{w_i} = \frac{p_j}{w_j}$, and $p_i > p_j$ then $\sigma_r(i) < \sigma_r(j)$.

For the short jobs, σ will satisfy an adaptation of the Weighted Shortest Processing Time (WSPT) and Longest Processing Time (LPT) rule: first all the late jobs are processed following WSPT (Smith's) rule and then all the on-time jobs are processed according to the LPT rule. All the late jobs with the same WSPT ratio are ordered according to the LPT rule (which is merely a tie-breaker). These rules for each partition are "almost" enough to be an appropriate set of permutations for a partition based on due dates: there is always an optimal solution that satisfies the rules in I_r for each $r \in \{1, \ldots, q\}$ except for maybe one job j, and this exception takes place only if the job j is late in I_r and is completed first in I_r. Based on this observation, Baptiste and Sadykov (2009) construct an algorithm that finds an appropriate partition for single machine problems. The procedure starts with a partition $\mathcal{I}$ for which $e_0 = 0$, $\{d_1, \ldots, d_n\} = \{e_1, \ldots, e_{q-1}\}$, and $e_q = T$, in other words, a partition based on due dates with the smallest number of intervals. If $T < d_n$ then we

stop at T and do not create further intervals; value e_{q-1} is then the largest due date less than T, and $e_q = T$. By subsequently dividing some intervals, a partition can be obtained that satisfies the conditions of Theorem 1.

THEOREM 1 **(Baptiste and Sadykov, 2009)**. *A partition* $\mathcal{I} = \{I_r\}_{r \in Q}$ *is* appropriate *if, for each* $r \in Q$ *and each pair of jobs* $i, j \in J$ *such that* $\sigma_r(i) < \sigma_r(j)$*, at least one of the following conditions holds:*

$$e_r \leq e_{r-1} + p_j \tag{4}$$

$$e_{r-1} \geq d_i + \left\lceil \frac{w_j p_i}{w_i} \right\rceil - p_i \tag{5}$$

Clement (2015) later showed that the algorithm provided in Baptiste and Sadykov (2009) provides an appropriate partition but not always a partition with a minimum number of intervals. Clement (2015) presents a method to construct an appropriate partition with a minimum number of intervals; we use his procedure in our implementation.

For the instance in Table 1, an appropriate partition is given by $I_1 = (0, 4]$, $I_2 = (4, 6]$, $I_3 = (6, 8]$, and $I_4 = (8, 11]$. The set of permutations σ is $\sigma_1 = (2, 3, 4, 1)$, $\sigma_2 = (2, 3, 4, 1)$, $\sigma_3 = (2, 3, 4, 1)$, and $\sigma_4 = (4, 2, 3, 1)$. Clearly, none of the intervals contains a "special" pair of jobs, i.e., a pair that does not satisfy the requirements of Theorem 1.

4. BDD-based formulation for parallel machine scheduling

Baptiste and Sadykov (2009) present a MIP formulation for single machine problems based on the ideas in the previous section, with binary variables for the assignment of jobs to intervals, which might also be generalized to parallel machines. In this work we follow a different approach: we will use the partition of the time horizon described by Baptiste and Sadykov to develop a new network-flow-based formulation. We will construct a binary decision diagram (BDD) over which at most m units of flow are pushed; each unit flow from the root node to the terminal node will represent a pseudo-schedule.

4.1. Introduction to BDDs

BDDs are data structures that allow one to represent and manipulate families of sets that can be linearly ordered. BDDs were introduced in Lee (1959) and Akers (1978) as DAGs that are obtained by reducing binary decision trees that represent a Boolean function. Recently, decision diagrams have also been used to solve discrete optimization problems.

Bergman et al. (2016), for instance, introduce a generic B&B algorithm for discrete optimization, where relaxed BDDs are used to compute relaxation bounds and restricted BDDs are used to find feasible solutions. Another relevant example is Cire and van Hoeve (2013), who use multi-valued decision diagrams to solve single machine scheduling problems. We refer to Castro et al. (2022) for a survey of recent advances in the use of decision diagrams for discrete optimization.

In the following definition, we follow the common references such as Castro et al. (2022) but we make some minor changes to suit the setting of our paper. A BDD B is a DAG (N, A) where the nodes are partitioned into (ordered) *layers*: $N = (N_1, N_2, \ldots, N_{\nu+1})$. The first and the last layer are the singletons $N_1 = \{\mathbf{r}\}$ an $N_{\nu+1} = \{\mathbf{1}\}$, respectively, with $\mathbf{r}$ the root node and $\mathbf{1}$ the terminal node. Every non-terminal node $i \in N$ has two outgoing edges: the *high edge*, which points to the *high child node* $\mathrm{hi}(i)$, and the *low edge*, pointing to the *low child node* $\mathrm{lo}(i)$. Both child nodes of $i \in N_k$ $(k = 1, \ldots, \nu)$ are located in strictly higher-indexed layers N_ℓ, so $\ell > k$. For our purposes, we will also associate with each node i within the same layer N_k $(k = 1, \ldots, \nu)$ a unique label $t \in \mathbb{N}$, and we call the combination (k, t) a *configuration*.

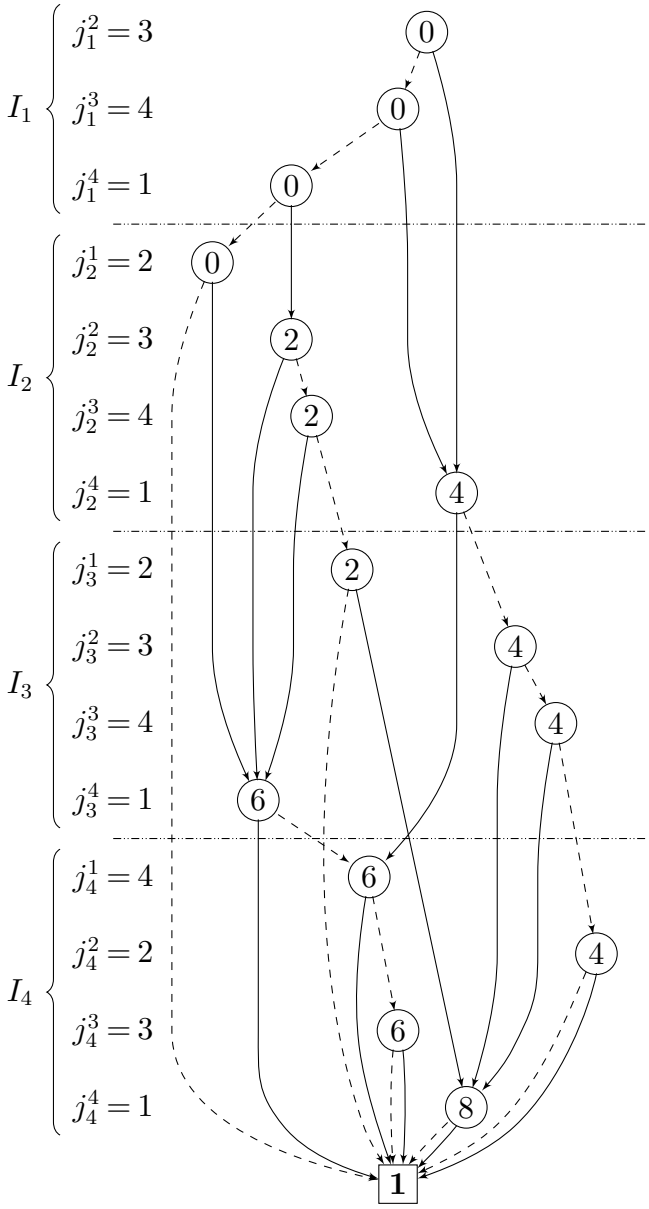
A BDD can be used to describe a family of subsets of a ground set V. Such a BDD will contain at most $|V| + 1$ layers, where each element $i \in V$ corresponds with at most one layer in $\{1, \ldots, \nu\}$; let $\mathrm{v}(i) \in V$ be the element in V corresponding with the layer of node $i \in N \setminus \{\mathbf{1}\}$. A path P from $\mathbf{r}$ to $\mathbf{1}$ in such a BDD B then induces a subset $S_P \subset V$, as follows. We start at the root node of B and iteratively add elements of V to S_P in the following way: if a is the current node on the path then if the next edge is the high edge of a then we add $\mathrm{v}(a)$ to S_P, otherwise not. Effectively, high edges represent selection and low edges exclusion of the element corresponding with the layer of their parent node.

One can construct a BDD associated to a family of subsets in different ways. In this work we use the efficient and generic recursive framework of Iwashita and Minato (2013). Below we show how to define a restricted family of pseudo-schedules, which is recursively constructed, based on the interval-indexed model of Baptiste and Sadykov (2009).

4.2. Constructing a BDD that contains all feasible sequences

Let $\mathcal{I} = \{I_1, \ldots, I_q\}$ be an appropriate partition of the time horizon and $\sigma = \{\sigma_1, \ldots, \sigma_q\}$ the set of permutations associated to $\mathcal{I}$. Each σ_r imposes an ordering $\prec_r$ of the jobs in interval I_r, and we write this as follows: $j_r^1 \prec_r \ldots \prec_r j_r^n$, meaning that if job j_r^1 is assigned to interval I_r then it is also the first job in interval I_r, otherwise the next job that can be assigned

Figure 3 **A BDD representing all sequences for the instance described in Table 1. Solid lines represent high edges, while dotted lines represent low edges.**



to interval I_r is job j_r^2, and so on. This leads to an ordering $\prec$ over the entire time horizon: $j_1^1 \prec j_1^2 \prec \ldots \prec j_1^n \prec j_2^1, \ldots \prec j_2^n \prec \ldots, j_q^1 \prec \ldots \prec j_q^n$. Obviously, for each $j \in J$ and $r \in Q$ there is only one $i \in \{1, \ldots, n\}$ such that $j_r^i = j$. For a feasible schedule we need to choose for each job $j \in J$ exactly one of its representations (in one of the intervals). We model this using a BDD to represent suitable subsets of the set $\{j_1^1, j_1^2, \ldots, j_1^n, j_2^1, \ldots, j_2^n, \ldots, j_q^1, \ldots, j_q^n\}$, while ensuring that each representation j_r^i is completed in the corresponding interval.

The BDD will follow the same ordering $\prec$. Figure 3 shows the BDD for the instance given in Table 1. The intervals I_1 to I_4 appear from top to bottom, and each layer of the BDD corresponds with one job occurrence j_r^i, appearing from top to bottom according to $\prec$. Each non-terminal node in the BDD corresponds with a configuration (j_r^i, t), which is a pair consisting of a representation j_r^i of a job j that can only be completed in interval I_r and the total processing time t of all the job representations that were chosen before j_r^i, so t is the starting time of job j_r^i. A node with configuration (j_r^i, t) appears in the layer with j_r^i and has t in the node itself. Note that a (job,time) configuration for a node is sufficient to uniquely identify the node, because a given job with a given starting time can only be assigned to one interval.

Each node (j_r^i, t) in the BDD apart from the terminal nodes has two child nodes. The high edge, representing inclusion of representation j_r^i, leads to $(j_{r'}^{i'}, t + p_j)$, where p_j is the processing time of the job $j = j_r^i$ and $j_{r'}^{i'}$ is the representation of the next job j' that is different from job j for which $t + p_j + p_{j'} \in I_{r'}$. If no such representation $j_{r'}^{i'}$ exists, the high edge points to the **1**-node. For the low edge (exclusion of representation j_r^i), the same holds but based on the value of t instead of $t + p_j$. An algorithmic description of the generation procedure of the BDD is provided in Algorithm 1 in Appendix A.

In conclusion, jobs are assigned to intervals based on their completion times, whereas the node labels contain the job starting times. In the instance, representation j_1^1 can never be chosen because job 2 cannot finish in interval I_1 since $p_2 = 6$ while $I_1 = (0, 4]$. As another illustration, the node $(j_2^2, 2) = (3, 2)$ decides upon the possible scheduling of job 3 at time 2, which would imply the completion time 6, which is the reason why this configuration is associated with interval $I_2 = (4, 6]$. The high edge emanating from this node implies that job 3 is effectively scheduled to start at time 2 and thus to end at time 6; the first job occurrence after j_2^2 according to the set of permutations described supra for this instance with starting time 6 and completing within its interval is then $j_3^4 = 1$. The low edge of $(3, 2)$ does not select job 3 and leads to $(4, 2)$, seeing that starting job 4 at time 2 still allows to end job 4 within I_2. It can also be seen that the low edge emanating from $(j_2^1, 0)$, corresponding with the non-selection of job 2 at starting time 0, immediately leads to the terminal node, because possible next jobs would end too early to complete in intervals I_2, I_3, or I_4, since we do not allow for idle time in the pseudo-schedules.

Similarly to the ATIF, every path from the root node to node **1** corresponds with a pseudo-schedule. In the example, the path from the root node leading up to $(j_2^1, 0)$ and then on to **1** along the low edge then corresponds with an empty machine schedule, which also constitutes a valid pseudo-schedule. In Appendix B we show two paths in the BDD of Figure 3 that together define an optimal two-machine solution to the example instance.

4.3. A new flow-based formulation for parallel machine scheduling

Let $B = (N, A)$ be the DAG that represents the constructed BDD. Each node v of the graph is associated to a configuration (j_r^i, t) and has two outgoing edges: high edge e_v^1 and low edge e_v^0. The high edge e_v^1 has a cost $c_{e_v^1} = w_j \max\{0, t + p_j - d_j\}$ with $j = j_r^i$, while the cost of low edge e_v^0 is 0. The set of all incoming edges of node v is given by $\delta^-(v)$. Let A^0 and A^1 be the set of all low and high edges of B, respectively. Let $p_B : A \to J$ be a map that projects each edge e in A onto the job associated to the head node of e.

A formulation for $Pm||\sum w_j T_j$ can now be constructed using a binary variable x_e for each $e \in A^1$ to indicate that the edge e is chosen, which means that job $j = p_B(e)$ is completed in interval I_r with completion time $C_j = t + p_j$, where the head node of e has configuration $(p_B(e), t)$. For each $e \in A^0$ we define a continuous variable x_e to allow all sequencing decisions to be represented by a flow from the root node of the BDD to the terminal node **1**. The formulation can now be stated as follows:

$$\text{minimize} \quad \sum_{e \in A^1} c_e x_e \tag{6a}$$

$$\text{subject to} \quad \sum_{e \in A^1 : p_B(e) = j} x_e = 1 \qquad \forall j \in J \tag{6b}$$

$$x_{e_v^1} + x_{e_v^0} = \sum_{e \in \delta^-(v)} x_e \qquad \forall v \in N \setminus \{\mathbf{r}, \mathbf{1}\} \tag{6c}$$

$$\sum_{e \in \delta^-(\mathbf{1})} x_e = m \tag{6d}$$

$$x_e \in \{0, 1\} \qquad \forall e \in A^1 \tag{6e}$$

$$x_e \geq 0 \qquad \forall e \in A^0 \tag{6f}$$

Equations (6c) and (6d) together with the redundant equation

$$x_{e_\mathbf{r}^0} + x_{e_\mathbf{r}^1} = m \tag{7}$$

can be interpreted as a network flow of m units through the BDD from the root node $\mathbf{r}$ to the terminal node $\mathbf{1}$. Constraints (6b) enforce that for each $j \in J$ we must choose exactly one edge $e \in A^1$ such that $p_B(e) = j$, meaning that we choose exactly one representation of each job across the intervals. In what follows, we call the new formulation (6) the BDD-based formulation BDDF. This flow-based formulation has quite some similarities with the recent work by van Hoeve (2022) on graph coloring with BDDs, where a path in the BDD corresponds with a subset of vertices of the graph to be colored. The ordering of the vertices (layers) in the BDD in van Hoeve's work is flexible, however (and it impacts the size of the graph), and also the objective (number of subsets) is different.

We now show that the formulation BDDF will yield better bounds than the TIF (1). We show that every solution $\overline{x}$ of the LP relaxation of BDDF can be transformed into a solution $\overline{y}$ of the LP relaxation of TIF. Consider for this the map $q_B : A^1 \to \{1, \ldots, T\}$ that projects each high edge of the BDD onto the starting period of its head node. For $j \in J$ and $t \in \{1, \ldots, T - p_j + 1\}$, let

$$\overline{y}_{jt} = \sum_{\substack{e \in A^1 : p_B(e) = j \\ q_B(e) = t}} \overline{x}_e.$$

Since $\overline{x}$ satisfies the assignment constraints (6b), it follows that $\overline{y}$ also satisfies the assignment constraints (1b), and accordingly Constraints (6c) and (6d) for $\overline{x}$ imply Constraint (1c) for $\overline{y}$.

We now show that the BDDF can be strictly better than the TIF for $Pm||\sum w_j T_j$. Consider the instance described in Table 1. An optimal solution for this instance has cost 4 with the job sequences $(1, 4, 3)$ and (2) on the two machines; this integral solution is also an optimal solution to the linear relaxation of the BDDF. An optimal solution $\overline{y}$ of the LP relaxation of the TIF is $\overline{y}_{11} = \overline{y}_{13} = \overline{y}_{31} = \overline{y}_{37} = 0.5$, $\overline{y}_{21} = \overline{y}_{45} = 1.0$, and all the other variables equal to 0. Clearly, this solution to the LP relaxation of the TIF (1) cannot be a solution of the LP relaxation of the BDDF (6), because the structure of the BDD diagram in Figure 3 implies that job 1 can not be assigned to the interval I_1 twice. With $\widetilde{c}_{11} = \widetilde{c}_{13} = \widetilde{c}_{21} = \widetilde{c}_{31} = \widetilde{c}_{45} = 0$ and $\widetilde{c}_{37} = 4$, the corresponding objective value equals 2. We thus obtain:

Proposition 1 *The BDDF dominates the TIF.*

The ATIF and the new BDDF are not comparable, however; the LP bound of ATIF can be either higher or lower than that of BDDF (see Appendix C for an illustration). The benefit of the BDDF is that within one interval each job can only occur once, even with extra intermediate jobs. The ATIF, on the other hand, avoids directly repeated jobs regardless their interval.

5. Solving the LP

5.1. Dantzig-Wolfe decomposition

The BDDF (Formulation (6)) can have many variables and constraints, which makes a direct application restrictive. Following van den Akker et al. (2000) and Pessoa et al. (2010), we apply a DW decomposition to the LP relaxation of the BDDF (i.e., when all variables x_e are non-negative reals). This will reduce the number of constraints from $|N|+n-1$ to $n+1$, where $|N|$ is the number of nodes in the BDD. We keep the assignment constraints (6b) and the bounding constraint (6d) in the formulation, but we recognize that the extreme points of the polytope formed by the flow constraints (6c) are the paths in the BDD from the root node to the terminal $\mathbf{1}$. Denote the set of all these paths by $\mathcal{P}$, and let z_e^p be a parameter that is one if edge e belongs to path p and zero otherwise. We introduce a new variable λ_p for each $p \in \mathcal{P}$, with which the LP relaxation of BDDF can be re-stated as follows:

$$\text{minimize} \quad \sum_{e \in A^1} c_e x_e \tag{8a}$$

$$\text{subject to} \quad \sum_{e \in A^1 : p_B(e) = j} x_e = 1 \qquad \forall j \in J \tag{8b}$$

$$\sum_{p \in \mathcal{P}} z_e^p \lambda_p = x_e \qquad \forall e \in A \tag{8c}$$

$$\sum_{e \in \delta^-(\mathbf{1})} x_e = m \tag{8d}$$

$$x_e \geq 0 \qquad \forall e \in A \tag{8e}$$

$$\lambda_p \geq 0 \qquad \forall p \in \mathcal{P} \tag{8f}$$

Eliminating the variables x_e from the model, we obtain:

$$\text{minimize} \quad \sum_{p \in \mathcal{P}} \left(\sum_{e \in A^1} c_e z_e^p \right) \lambda_p \tag{9a}$$

$$\text{subject to} \sum_{p\in\mathcal{P}} \left(\sum_{e\in A^1: p_B(e)=j} z_e^p \right) \lambda_p = 1 \qquad \forall j \in J \tag{9b}$$

$$\sum_{p\in\mathcal{P}} \left(\sum_{e\in\delta^-(\mathbf{1})} z_e^p \right) \lambda_p = \sum_{p\in\mathcal{P}} \lambda_p = m \tag{9c}$$

$$\lambda_p \geq 0 \qquad \forall p \in \mathcal{P} \tag{9d}$$

5.2. Column generation

We solve the decomposed model (9) with CG, which implies the iterative solution of a *restricted master problem* (RMP), which contains a limited set of columns, and a *pricing problem*, which checks whether there exists a column with negative reduced cost. If we assign dual variables π_j for $j \in J$ with Constraints (9b) and dual variable π_0 with Constraint (9c), the dual of the LP (9) is given by:

$$\text{maximize} \sum_{j\in J} \pi_j + m\pi_0 \tag{10a}$$

$$\text{subject to} \sum_{j\in J} \left(\sum_{e\in A^1: p_B(e)=j} z_e^p \right) \pi_j + \pi_0 \leq \sum_{e\in A^1} c_e z_e^p \qquad \forall p \in \mathcal{P} \tag{10b}$$

$$\pi_j \in \mathbb{R} \qquad \forall j \in J \tag{10c}$$

$$\pi_0 \in \mathbb{R} \tag{10d}$$

At each iteration of the CG algorithm we check if one of the constraints (10b) is violated, meaning that the reduced cost of the associated column is negative. Recall that a column is a path from the root node to the terminal **1** in the BDD that was presented in Section 3. The pricing problem is then to verify whether given current dual prices $\overline{\pi}$, a path $p \in \mathcal{P}$ exists such that

$$\sum_{e\in A^1} c_e z_e^p - \sum_{j\in J} \left(\sum_{e\in A^1: p_B(e)=j} z_e^p \right) \overline{\pi}_j - \overline{\pi}_0 < 0. \tag{11}$$

Inequality (11) can be rewritten as

$$\left(\sum_{e\in A^1\cap p} \overline{c}_e \right) - \overline{\pi}_0 < 0, \tag{12}$$

where $\overline{c}_e$ is equal to $c_e - \overline{\pi}_{p_B(e)}$, which is the reduced cost of high edge e.

A CG algorithm typically needs fewer iterations if one considers constraints that are strongly violated and hence we will identify paths with the lowest reduced cost. In this way

the pricing problem becomes a shortest path problem in the BDD, where the length of the high edges e is $\overline{c}_e$, and the length of the low edges is zero. Since the graph is acyclic and has only two outgoing arcs per node, the running time of a labeling algorithm for pricing will be linear in the number of nodes in the BDD (Ahuja et al., 1993).

5.3. Labeling algorithm

In our initial computational experiments we noticed that the pricing algorithm often generates paths from the root node to the terminal for which the associated pseudo-schedules contain jobs that are repeated in consecutive positions. The consequence of this is that the lower bound will tend to be weaker than the bound from the ATIF (but of course still stronger than the bound from the TIF). In Figure 3, for example, the path corresponding to the pseudo-schedule $(j_1^4, j_2^2, j_4^3) = (1, 3, 3)$ is allowed, where the path includes the low edges emanating from $(j_3^4, 6) = (1, 6)$ and $(j_4^1, 6) = (4, 6)$. In Section 4.2 we mentioned that we avoid two consecutive high edges for the same job, but this does not yet avoid repeated jobs via intermediate low edges.

In principle, one can impose the condition that no job can be visited by a path more than once: we can compute the intersection of the family of pseudo-schedules and the family of paths where each job is visited at most once (see Minato (1993) for a generic intersection operation on BDDs). In this way, we obtain a BDD that contains exactly all possible schedules. It would be overly time-consuming to construct such a BDD, however, and the pricing problem would also become much harder to solve because of the number of nodes in the resulting BDD.

It is easier to restrict the pseudo-schedules such that all pairs of consecutive jobs are different. Note that two jobs assigned to the same time interval will always be different, so if two successive tasks in a pseudo-schedule are the same then they are assigned to different intervals. Thus, one might say that the BDDF only "remembers" what happens in the same interval. This implicit memory mechanism is the most fundamental reason why the flow-based formulation is stronger than the TIF. We therefore devise a labeling algorithm for pricing that takes into account that two consecutive jobs in the pseudo-schedule cannot be the same. This restriction will have a significant impact on the quality of the lower bound, and the running time of the algorithm will still be linear in the number of nodes in the BDD.

To avoid that two consecutive jobs in the pseudo-schedule are the same, we need to know the previous job in the optimal path to avoid that the same job is scheduled consecutively. We therefore maintain a bucket with two entries at each node in the BDD: each entry contains a distance label and the identification of the previously selected job to achieve that distance label. The first entry is the lowest cost to reach the node, and the second entry is the lowest cost while not passing via the same predecessor job as the first entry. This modified labeling algorithm can be implemented in a forward or backward fashion. We have observed in preliminary experiments that the forward labeling algorithm is more time-consuming than the backward variant because the number of label updates is higher in that case. In the forward labeling algorithm, we may have to update the labels more often because the in-degree (the number of incoming edges) of each node can be higher than the out-degree (the number of outgoing edges), which is at most two. Hence, in our computational experiments, we use the backward labeling algorithm to find paths with minimal reduced cost. The forward labeling algorithm is used to remove nodes from the BDD (N, A) by reduced cost fixing. A more detailed description of the forward and backward algorithm is provided in Appendix D.

By avoiding identical consecutive jobs, BDDF dominates ATIF in principle because a job is only selected once at most in each interval, while ATIF cannot guarantee this. This dominance only holds for a basic implementation of the ATIF, however. In a preprocessing step, extra variables can be removed from ATIF based on pairwise interchange (as described in Proposition 2 (and 3) of Pessoa et al. (2010)). The logic behind these pairwise interchanges is embedded into the BDDF only in the ordering within each interval, while the ATIF benefits from this across the entire time horizon.

5.4. Stabilization

The convergence of the CG algorithm can be slow because of primal degeneracy. This problem can be circumvented by applying stabilization methods for CG. We apply a smoothing method that was developed by Wentges (1997), in which we correct the optimal solution of the dual problem of the RMP based on information from the previous iterations before plugging it into the pricing problem. For details of this technique and of stabilization in general, we refer to Pessoa et al. (2018).

5.5. Reduced cost fixing

In many B&P algorithms, the CG pricing problem consists of computing feasible paths in a network. Irnich et al. (2010) and Pessoa et al. (2010) explain how to remove some arcs from the underlying network (fixing variables to zero) without compromising optimality, and thus improve the convergence of the CG. In our case, we can fix the flow on edge $e \in A^1$ to 0 (so remove the high edge from the BDD) if

$$LB + (m-1)\bar{c} + \chi_e \geq UB, \tag{13}$$

where χ_e is the best reduced cost of a path from the root node to the terminal node $\mathbf{1}$ that traverses the edge e, $\bar{c}$ is the reduced cost of the shortest path from the root node to node $\mathbf{1}$, LB is the current lower bound of the RMP of (9), and UB is the best known upper bound of the optimal cost. In this way, we only remove arcs that will not improve the current best solution. The computation of χ_e uses the forward and backward distance labels discussed in Section 5.3.

Fixing edges by reduced cost not only has a beneficial effect on the running time of the pricing algorithm, but can also speed up the B&B procedure for solving the integer formulation by making the BDD smaller (as a pre-solving step) and via better lower bounds (see Irnich et al., 2010; Pessoa et al., 2010). Computationally, we find that applying variable fixing each time a number of CG iterations has past, performs better than only doing this at the end of the CG. Concretely, we execute variable fixing at the beginning of the CG, at the end of the CG phase, and after every 50 CG iterations.

6. Branch and price

In order to find optimal integer solutions for the BDDF, we embed the CG into a B&B search tree, leading to a B&P procedure. The addition of branching constraints to the LP has the undesirable effect of increasing the size of the formulation as we progress; a more efficient way to branch is to change the bounds on variables, and Linderoth and Savelsbergh (1999) explain how *generalized upper bound* (GUB) constraints of the form $\sum_{i \in N} x_i = 1$, for some set N of binaries, can be used. The branching scheme *GUB dichotomy* is based on a subset $N' \subsetneq N$ for which the solution of the LP relaxation at a node satisfies $0 < \sum_{i \in N'} x_i < 1$, where we enforce constraint $\sum_{i \in N'} x_i = 0$ in one child node and constraint $\sum_{i \in N \setminus N'} x_i = 0$ in the other child node. A clear advantage of branching over

GUB constraints instead of branching over individual variables is that the tree can be more balanced.

If there exists a logical ordering of the variables in set N then N is sometimes called a *special ordered set* (SOS). For the assignment constraints (6b) in the BDDF, which require the selection of one high edge for each job, we can order the edges in $A_j = \{e \in A^1 : p_B(e) = j\}$ in non-decreasing order of the starting time $q_B(e)$ for each $j \in J$, so each A_j clearly is an SOS. Following Linderoth and Savelsbergh (1999), *SOS branching* can be based on the subset

$$A'_j = \left\{ e \in A_j \,|\, q_B(e) \leq \sum_{\ell \in A_j} q_B(\ell) x^*_\ell \right\},$$

where x^* is a solution of the linear relaxation and A'_j serves as the set N'. In other words, we partition the high edges for job j based on how the implied starting time for j compares to the "average" starting time in the LP solution. Alternatively, branching can also be based on the weighted tardiness value c_e.

There can still remain multiple branching choices, namely for every $j \in J$ we can branch if the corresponding high edges in A_j are not integral. We first consider the subset of those jobs with non-zero average tardiness; branching on such edges often allows to quickly prune nodes in the search tree. If no job with fractional edges and non-zero tardiness exists then we branch on starting times. We apply strong branching to make good branching decisions. We take a small set of branching candidates (at most 50) and evaluate the child nodes heuristically by performing a small number ($2m$) of CG iterations. This first phase produces a ranking, and in this order we fully evaluate the child nodes. If for a number of consecutive full evaluations of the child nodes we do not find better bounds, we terminate the full evaluations and branch on the best candidate. Concretely, we maintain a counter of the number of non-improving fully evaluated nodes; we terminate when this counter exceeds 3, but when there is an improvement the counter is divided by 2, rounded down.

7. Computational experiments

7.1. Implementation details and instances

All algorithms have been implemented in the C++ programming language and compiled with gcc version 11.2.0 with full optimization pack -O3. We have used and adjusted the implementation of Iwashita and Minato (2013) that can be found on Github[1] to construct

[1] https://github.com/kunisura/TdZdd

the BDDs. All computational experiments were performed on one core of a server with Intel Xeon E5–4610 at 2.4GHz processors and 64 GB of RAM under a Linux OS. All LPs are solved with Gurobi 9.1.2 using default settings and only one core. The source code of the procedures can be retrieved from the Github repository by Kowalczyk et al. (2024).

We use the same instances from the OR-library as Pessoa et al. (2010) and Oliveira and Pessoa (2020). These instances were generated for the single machine problem with weighted tardiness objective in Potts and Van Wassenhove (1985). There are 125 instances for each $n \in \{40, 50, 100\}$. The processing time p_j for each $j \in \{1, \ldots, n\}$ was generated from the discrete uniform distribution on the integers in $[1, 100]$ and the weight w_j was generated similarly from $[1, 10]$. It was observed that the difficulty of $1||\sum w_j T_j$ depends on two parameters, namely the relative range of due dates RDD, and the tardiness factor TF. The due dates are generated from the discrete uniform distribution on $[P(1-TF-RDD/2), P(1-TF+RDD/2)]$, where $P = \sum_{j \in J} p_j$ and TF, $RDD \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. For each $n \in \{40, 50, 100\}$ and each pair (RDD, TF), five instances were constructed. In order to obtain reasonable instances for parallel machine scheduling, Pessoa et al. (2010) transformed the instances of Potts and Van Wassenhove (1985) by dividing the due dates by the number of machines m. The processing times p_j and weights w_j are kept the same for each $j \in J$. For each pair (RDD, TF) they only retain the first instance; thus there are 25 instances for each $n \in \{40, 50, 100\}$ and each $m \in \{2, 4\}$.

In our experiments we also use a primal heuristic. The algorithm creates an initial solution by iteratively assigning jobs (in descending order of their due dates) to a machine with the smallest workload. The heuristic then proceeds with a rudimentary iterated local search mechanism that changes the position of jobs or groups of jobs, or changes their machine allocation.

The local search consists of 1000 iterations. Within each iteration, several moves are tried in a fastest-descent fashion; these moves are the insertion or swap of one or a block of a limited number of jobs (blocks of maximum three jobs); these moves are similar to those in Kramer and Subramanian (2019). All moves are applied in a random order, and we stay within the same iteration as long as an improving move can be found; the evaluation of the effect of a move is efficiently done following Kramer and Subramanian (2019). Subsequently, we move from one iteration to the next by perturbing the current solution by means of a random selection of moves (out of the same set of moves; the number is drawn randomly between 1 and 8).

Table 2 Size of the graph for TIF, ATIF, BDDF$_r$, and BDDF

		TIF			ATIF			BDDF$_r$			BDDF
n	m	*avg size*	*max size*	*avg red*	*avg size*	*max size*	*avg red*	*avg size*	*max size*	*avg red*	*avg red*
40	2	40,429.9	48,853	78.1%	790,466.9	954,703	88.7%	126,406.3	171,440	86.1%	85.6%
40	4	21,178.7	25,133	87.4%	395,336.0	467,850	91.2%	71,374.4	88,404	90.1%	89.4%
50	2	63,730.8	73,205	80.4%	1,564,008.6	1,796,148	90.4%	199,107.5	257,220	84.5%	84.2%
50	4	33,066.8	37,605	85.7%	781,463.4	887,636	91.2%	111,098.7	131,976	88.7%	88.2%
100	2	257,888.0	294,297	76.6%	12,770,654.5	14,572,902	90.5%	813,603.8	1,049,008	83.7%	83.4%
100	4	131,364.0	149,197	83.2%	6,379,927.2	7,243,901	91.9%	448,032.2	536,716	85.1%	84.8%

7.2. Comparison of the LP bounds

In this section, we will present computational results of CG for the LP bound computation of the TIF (1), the ATIF (2), and our new formulation BDDF (6). We have implemented a CG algorithm for each of these three formulations, with the same enhancements such as stabilization and reduced cost fixing for all three models. We also incorporate the pairwise-interchange-based preprocessing rules discussed at the end of Section 5.3 in the ATIF; a similar interchange argument is implicitly embedded in the BDDF only within each interval, while this can benefit the ATIF over the entire time horizon. In this section, "BDDF" refers to the formulation with a standard labeling algorithm in the CG phase, which can generate consecutive repeated jobs, while "BDDF$_r$" stands for CG with the labeling refinement described in Section 5.3 that avoids identical jobs in consecutive positions in a pseudo-schedule.

The graphs that represent the ATIF and the BDDF are much larger than those for the TIF. The number of edges for ATIF is $O(n^2T)$, while this number is $O(nT)$ for the TIF. Table 2 provides some empirical evidence for this by comparing the average (*avg size*) and maximum (*max size*) number of edges in all formulations. The graphs for BDDF$_r$ and BDDF are obviously the same, so those columns are not duplicated. We see that the number of edges in the graph that represents the BDDF falls in between the numbers for the other two formulations. The column *avg red* presents the average percentage of edges that were removed via reduced cost fixing by the end of the CG procedure. We observe that the ATIF benefits the most from this variable fixing, with around 90% of the high edges removed for all instance classes, followed by the BDDF, and finally the TIF has the lowest average reduction, but this still amounts to 76.6% at least across the instance classes. Model BDDF$_r$ is a bit more restrictive than BDDF and benefits slightly more from variable fixing, but the differences are not very large.

Table 3 **Computation time (in seconds) and number of instances solved at the root for the LP relaxation of TIF and ATIF**

		TIF			ATIF		
n	m	*avg time*	*max time*	*# opt*	*avg time*	*max time*	*# opt*
40	2	0.59	1.16	6	2.16	4.09	7
40	4	0.31	0.67	8	0.82	1.50	10
50	2	1.12	3.13	6	5.23	12.34	7
50	4	0.63	1.42	7	1.90	3.69	8
100	2	14.43	35.17	4	90.42	183.96	4
100	4	7.29	16.22	6	26.71	51.46	6

Table 4 **Computation time (in seconds) and number of instances solved at the root for the LP relaxation of BDDF$_r$ and BDDF**

		BDDF$_r$			BDDF		
n	m	*avg time*	*max time*	*# opt*	*avg time*	*max time*	*# opt*
40	2	1.39	2.79	9	1.80	3.89	8
40	4	0.67	1.38	9	0.87	1.77	8
50	2	2.57	3.84	8	3.25	5.83	8
50	4	1.29	2.62	8	1.70	3.52	8
100	2	27.26	52.14	6	36.18	60.49	6
100	4	12.01	20.46	6	15.58	25.21	6

In Tables 3 and 4 we report the runtimes of the CG algorithms for TIF, ATIF, BDDF$_r$, and BDDF. The columns *avg time*, *max time*, and *# opt* contain the average and the maximum CPU time of the algorithms (in seconds), and the number of instances solved at the root node (out of 25), respectively. An instance is said to be solved at the root node when the linear relaxation can confirm optimality of the primal heuristic described in Section 7.1; the same initial heuristic is used for all tested models in this Section 7.2. We find that the average running time of the CG for computing the lower bound with BDDF is significantly less than with ATIF, and also that the time needed for TIF, in turn, is a lot lower than with BDDF. These observations are completely in line with the size of the graphs in which the pricing procedures are executed, which was reported in Table 2. Avoiding consecutive identical jobs via labeling is beneficial: BDDF$_r$ is consistently faster than BDDF.

The gap between the starting solution and the LP bound of the different formulations is given in Table 5 (we report both the average (*avg*) as well as the maximum (*max*) gap). The pattern that arises here is not as clear-cut as in the previous two tables: we see from Table 5 that, despite the smaller graphs and the lower runtimes than the ATIF, the BDDF still yields LP bounds that are quite tight, and very close on average to the ones produced

Table 5 **Average and maxium gap from the starting solution for the formulations TIF, ATIF, BDDF$_r$, and BDDF**

		TIF		ATIF		BDDF$_r$		BDDF	
n	m	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*
40	2	1.78%	26.61%	1.73%	26.61%	1.49%	26.61%	1.53%	27.05%
40	4	0.56%	5.03%	0.46%	3.68%	0.49%	4.44%	0.53%	4.83%
50	2	0.62%	4.25%	0.57%	4.25%	0.58%	4.25%	0.59%	4.25%
50	4	0.54%	5.95%	0.51%	5.95%	0.51%	5.95%	0.52%	5.95%
100	2	2.27%	35.29%	1.62%	21.05%	0.79%	14.79%	0.83%	15.82%
100	4	0.54%	8.62%	0.52%	8.62%	0.53%	8.62%	0.53%	8.62%

by ATIF. We conclude that while the BDDF is a formulation that is positioned between the TIF and the ATIF in terms of runtimes and graph size for CG, the LP bounds produced by the BDDF are of rather similar quality as the ATIF, which makes the formulation promising for finding optimal integer solutions.

7.3. Comparison of exact procedures

In this section, we present the computational results of the overall B&P algorithm based on the BDDF formulation. We compare our algorithm with the currently most competitive procedure in the literature, which is the one by Oliveira and Pessoa (2020). In what follows, we refer to our new B&P procedure based on the BDDF simply as "BDDF," and to the algorithm devised by Oliveira and Pessoa (2020) and which is based on ATIF as "ATIF." We incorporate the labeling refinement described in Section 5.3 (which was previously referred to as BDDF$_r$) into BDDF. Oliveira and Pessoa (2020) feed the solutions found by the heuristic of Kramer and Subramanian (2019) into their procedure as initial primal bounds, while BDDF computes an initial solution using the local search procedure mentioned in Section 7.1.

The processors in our hardware are clearly slower than those used by Oliveira and Pessoa (2020): the CPU in Oliveira and Pessoa (2020) has around 30% higher clock speed (according to benchmarking websites[2]). Hence, we transform the results of our algorithm accordingly, namely we multiply our results by factor 0.7. The time limit per instance is set to 7200 seconds for our computations (without rescaling). It should be clear that this correction factor can only constitute a rough approximation of the differences due to processor speed, operating system, etc., and therefore the results in this section mainly serve

[2] See, for instance, `https://www.cpubenchmark.net/` for a comparison of the CPUs.

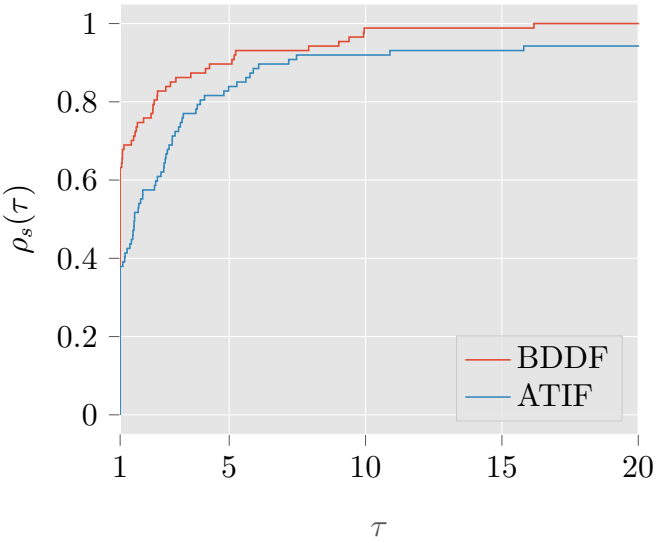
to provide a rough overall idea of how competitive our new method is, while all conclusions from the head-to-head comparison between the two algorithms are only indicative.

As a tool for comparing the computational performance of the two procedures, we will use *performance profiles*, which were proposed as a tool for benchmarking optimization software in Dolan and Moré (2002). The idea is to compare the methods by the ratio of each method's runtime to the best runtime, per instance. Let $r_{p,s}$ be this ratio for method s on instance p, and let $\rho_s(\tau)$ be the probability for method s that a performance ratio $r_{p,s}$ for a given instance p is within a factor $\tau \in \mathbb{R}$ of the best possible ratio. The function ρ_s is then a performance profile, which can be seen as the (cumulative) distribution function for the performance ratio over all tested instances. In other words, considering τ as the time needed by an algorithm normalized with respect to the best algorithm, for each value of τ a performance profile curve reports the fraction of the data set for which the algorithm is at most τ times slower than the best algorithm. For a more detailed description of performance profile curves we refer to Dolan and Moré (2002).

In Figure 4 we plot the performance profiles for BDDF and ATIF for all integer $\tau = 1, 2, 3, \ldots$ based on all the instances that were solved by both methods (so this excludes instances where one of the two methods found a guaranteed optimal solution, but the other one did not). Oliveira and Pessoa (2020) only provide detailed computational results for instances that were not "trivial," i.e., not solved in the root node by merely calculating the LP relaxation of the formulation without additional cuts. BDDF comes out rather favorably in this plot: from Figure 4 we can deduce that BDDF is the fastest algorithm for approximately 65% of the instances, while this is the case for only almost 40% for ATIF (this can be read for the entry $\tau = 1$ on the horizontal axis, which is where the plot starts). Algorithm BDDF can solve approximately 90% of the instances within a computing time not exceeding five times the time for ATIF. Since the details for the trivial instances are not presented in Oliveira and Pessoa (2020), we do not fully see the effect of the faster CG phase of BDDF here.

In Figure 5 we present performance profiles per instance class, i.e., per combination (n, m). Clearly, BDDF outperforms ATIF for all instance sets with $m = 4$ machines (the three plots on the right side). Conversely, for instances with two machines (the left plots) ATIF wins the comparison, although the difference in performance is slightly less pronounced than for the case with four machines.

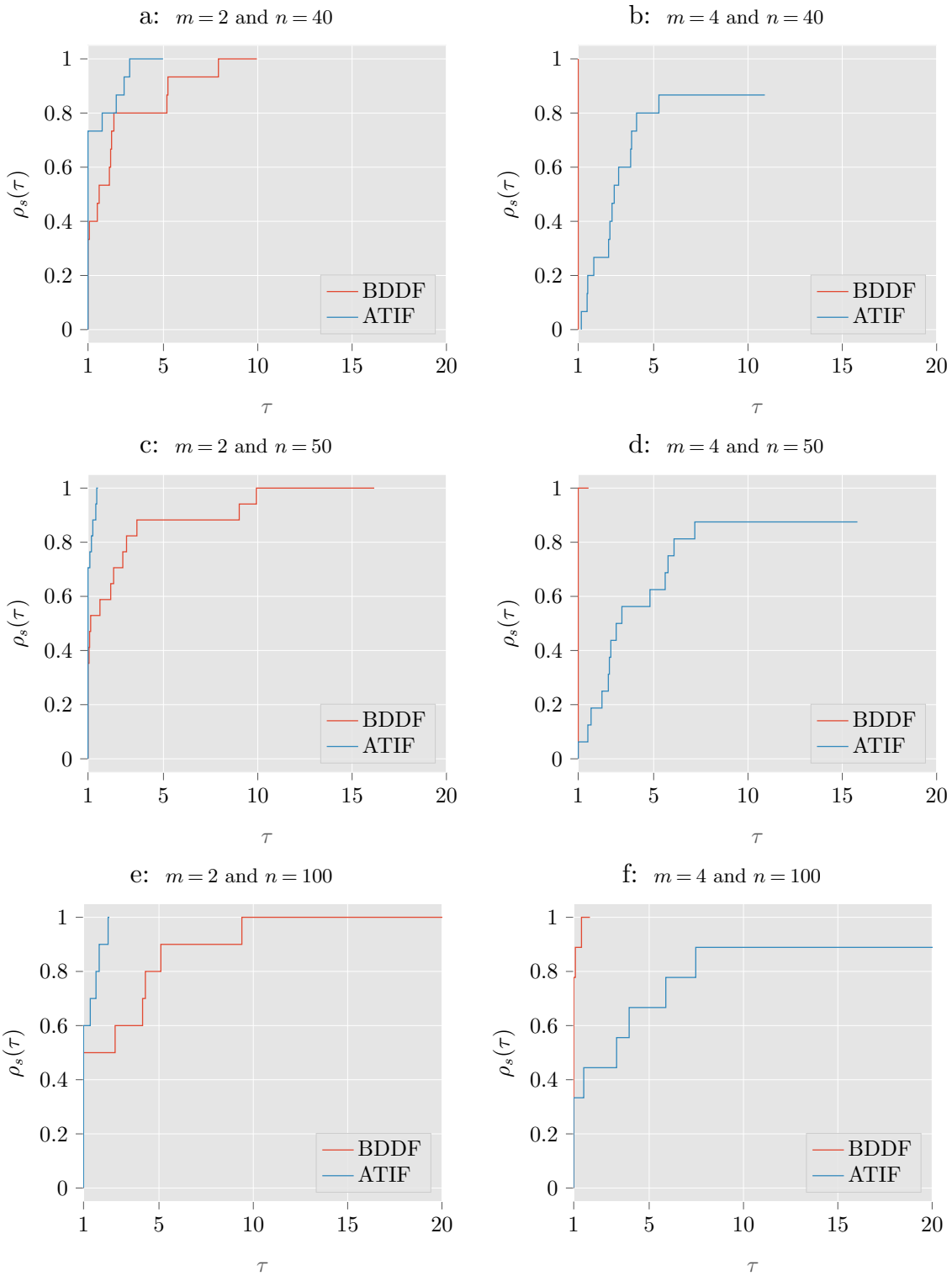
Figure 4 **Performance profiles over all the instances solved to optimality by both algorithms ($s =$ BDDF, ATIF)**



In Table 6 we summarize the results of the two algorithms. Columns *avg time* contain the average running time over all solved non-trivial instances (in seconds), under *solved* we report the number of solved instances (out of 25), and in the column *avg time all* we display the average time over all solved instances (only for BDDF; in seconds). Algorithm ATIF solves more instances to optimality; the runtime limit imposed is not clear, however: some of the instances have taken more than one day to run in Oliveira and Pessoa (2020). Consequently, a perfect comparison between the two methods is not possible based on this table. Nevertheless, the overall pattern that was observed in Figure 5 also occurs here: the runtimes of ATIF are significantly higher than BDDF for $m = 4$, while the differences are not that clear-cut for $m = 2$; only for $n = 50$ ATIF really dominates BDDF for $m = 2$. We conjecture that the cuts that are used by Oliveira and Pessoa (2020) are particularly helpful in tightening the formulation especially for instances with few machines. Overall, for BDDF our strong branching mechanism can close the gap relatively quickly for $n = 40$ and 50, while this is not the case anymore for instances with 100 jobs.

In Appendix E we present the detailed computational results of the B&P procedure based on the formulation BDDF for every instance. We can also report the optimal solution of three previously unsolved instances in the instance class with $n = 100$ and $m = 4$, namely those with ID number 16, 31, and 56. The only remaining unsolved instance in the data set is the one with ID number 91 for $n = 100$ and $m = 2$. The instances in the rows with $UB = LB = 0$ are trivial instances, for which the starting solution of the primal heuristic

Figure 5 **Performance profiles per instance class ($s =$ BDDF, ATIF)**



already finds an optimal solution. This happens for the same ID values (namely 51, 76, 101, and 106) across the six tables due to the specific values of RDD and TF for those

Table 6 Summary of the results for the exact procedures

		ATIF		BDDF		
n	m	*avg time*	*solved*	*avg time*	*solved*	*avg time all*
40	2	55.51	25	40.45	25	26.29
40	4	30.16	25	7.60	25	5.00
50	2	18.65	25	56.27	25	40.85
50	4	272.59	25	25.31	25	17.43
100	2	852.23	24	756.88	16	520.94
100	4	7,396.23	22	847.54	19	642.13

instances. We also observe that many instances are solved by one algorithm but not the other for $n = 100$: three instances are solved by BDDF and not ATIF for $m = 4$, and the number of instances solved by ATIF and not BDDF is eight for $m = 2$ and six for $m = 4$. We conclude that the behavior of the two algorithms (in terms of the difficulty experienced for solving instances that differ in their due date settings) is very different, and even somewhat complementary.

7.4. Effect of other algorithmic choices

In this subsection, we first investigate the impact of different initial solutions on the performance and efficiency of our procedure. Our results are summarized in Table 7, where we compare the standard implementation of the BDDF-based B&P ("standard," using the local-search-based primal heuristic described in Section 7.1) with two other choices: in "KS bounds" we plug in the primal bounds reported by Kramer and Subramanian (2019) (similarly to Oliveira and Pessoa's (2020) procedure described in Section 7.3), while "simple" is the variant where our local search is only allowed to execute one instead of 1000 iterations. The columns labeled *solved* report the number of instances solved within the runtime limit, while *avg time* in Table 7 is the average runtime over all 25 instances. In this case, we have not rescaled the runtimes, which implies that for "standard" the first four *avg time* entries in the table (with all 25 instances solved) equal the BDDF entries *avg time all* of Table 6 divided by 0.7.

Clearly, the setting "KS bounds" outperforms the other implementations, and this especially for $m = 4$, with lower average runtimes and more instances solved; the differences for $m = 2$ are a bit less clear-cut. The "simple" variant does worse than the other two implementations, with fewer instances solved and higher runtimes. We conclude that a stronger initial primal bound improves the performance of the B&P procedure; the absence of a strong heuristic or informed starting point hampers the algorithm's ability to explore the

Table 7 **Summary of the results of BDDF with different initial solutions and with additional pairwise-interchange rule (without rescaling of time)**

		KS bounds		standard		simple		interchange	
n	m	*avg time*	*solved*	*avg time*	*solved*	*avg time*	*solved*	*avg time*	*solved*
40	2	44.92	25	37.55	25	423.36	24	31.98	25
40	4	6.80	25	7.14	25	441.94	24	7.02	25
50	2	55.42	25	58.35	25	877.09	23	50.01	25
50	4	17.06	25	24.90	25	324.14	25	14.69	25
100	2	3,143.52	16	3,073.28	16	5,309.61	8	2,751.04	17
100	4	1,600.81	23	2,411.63	19	5,218.42	8	1,728.98	22

solution space efficiently. Better bounds will allow to eliminate unpromising regions from the search space in the B&B process from the outset, and will also have a direct effect on the variable fixing: the better the bound, the more variables can be fixed, which propagates throughout the BDD, reducing the size of the BDD and rendering the pricing problem easier, and probably even strengthening the LP bound in the root node.

Finally, we also briefly report on our attempt to include a pairwise-interchange dominance rule into the procedure (similar to the way in which Pessoa et al., 2010, remove variables from ATIF). For consecutive high edges, this is easily incorporated into the construction of the BDD (see Section 4.2) by skipping job representations in the choice of the head node of a high edge. The setting "interchange" in Table 7 contains the results for the implementation "KS bounds" augmented with this dominance rule. We observe that this extension has a little effect on the size of the generated graph, but does not consistently reduce the overall running time of the algorithm (compared to the columns "KS bounds").

8. Conclusion and further research

In this work, we have introduced a new formulation for $Pm||\sum w_jT_j$ based on binary decision diagrams, which are built using a time discretization from Baptiste and Sadykov (2009). We show theoretically and experimentally that this formulation is stronger than the classical time-indexed formulation, and show experimentally that this formulation is sometimes weaker and sometimes stronger than the arc-time-indexed formulation. The computation time of the LP lower bound of the new formulation with column generation is lower than for the bound computation of the arc-time-indexed model. The reason for this is mainly the size of the graphs that represent the different formulations. We have also developed a branch-and-price procedure based on the new formulation; this procedure can solve many instances faster than before, thanks to strong branching together with

the improved running time of the column generation. Compared with the state-of-the-art procedure of Oliveira and Pessoa (2020), our new procedure seems to perform better especially with a larger number of machines.

As a prime avenue for further research, one can examine several techniques from the rich vehicle routing literature to construct a branch-cut-and-price algorithm for the new flow-based formulation. Further closing the gap without branching but rather by introducing cuts seems to be a logical next step for rendering the resulting algorithm more competitive. Pessoa et al. (2010) considered such a plan of attack for the arc-time-indexed formulation for single and parallel machine scheduling, and derived robust cuts (which do not destroy the structure of the pricing problem; see also Poggi de Aragão and Uchoa, 2003). A similar approach for scheduling on one machine was followed by van den Akker et al. (2000) for the time-indexed formulation. Potentially, separation for the new formulation could be faster than for the arc-time-indexed model due to its lower number of variables.

A different interesting alternative for continuing this work is to develop a variant of the enumeration algorithm devised in Baldacci et al. (2008) for vehicle routing. The algorithm would iterate over all the paths from the root node to the terminal node in the decision diagram with a reduced cost that is less than the duality gap. One can then construct a set-partitioning formulation containing all these paths and hand the resulting formulation to a general MIP solver. In this case, it may be possible to add non-robust cuts to the formulation and to do the pricing by inspection if the number of retained schedules is low enough.

Another attractive option for further work is to examine the extension of our labeling algorithm that prevents identical consecutive jobs to keep track of (and avoid) the previous K jobs. Such a more aggressive refinement would require more detailed bookkeeping at each node in the BDD, but might also have a considerable impact on the quality of the bounds. Cycle elimination with $K = 1$ has already been incorporated routinely in earlier routing research (see, for example, Section 8 of Houck et al., 1978), and cycle elimination with $K \geq 2$ has also already been studied; the work by Irnich and Villeneuve (2006), Fukasawa et al. (2006), and Baldacci et al. (2011) can serve as a valuable starting point for such extensions in the context of our model. The incorporation of pairwise-interchange dominance rules into the labelling algorithm is another option that deserves further investigation (such a rule was tested for consecutive high edges in the creation of the BDD, but not for paths

with intermediate low edges). Finally, one might also try to leverage van Hoeve's (2022) constraint separation to stepwise reduce infeasible columns from the formulation.

As a final opportunity for further work, one can try to extend the new flow-based formulation to parallel machine scheduling problems with other constraints and objective functions. It would be interesting to examine, for example, whether the formulation can be adapted to the parallel machine scheduling problem with earliness-tardiness objective, and whether idle time can be incorporated.

References

Ahuja, R.K., T.L. Magnanti, J.B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall.

Akers, S.B. 1978. Binary decision diagrams. *IEEE Transactions on Computers* **100** 509–516.

Baldacci, R., N. Christofides, A. Mingozzi. 2008. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming* **115** 351–385.

Baldacci, R., A. Mingozzi, R. Roberti. 2011. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research* **59** 1269–1283.

Baptiste, P., R. Sadykov. 2009. On scheduling a single machine to minimize a piecewise linear objective function: A compact MIP formulation. *Naval Research Logistics* **56** 487–502.

Bergman, D., A.A. Cire, W.J. van Hoeve, J.N. Hooker. 2016. Discrete optimization with decision diagrams. *INFORMS Journal on Computing* **28** 47–66.

Bigras, L.P., M. Gamache, G. Savard. 2008. Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing* **20** 133–142.

Boland, N., R. Clement, H. Waterer. 2016. A bucket indexed formulation for nonpreemptive single machine scheduling problems. *INFORMS Journal on Computing* **28** 14–30.

Castro, M.P., A.A. Cire, J.C. Beck. 2022. Decision diagrams for discrete optimization: A survey of recent advances. *INFORMS Journal on Computing* **34** 2271–2295.

Cire, A.A., W-J. van Hoeve. 2013. Multivalued decision diagrams for sequencing problems. *Operations Research* **61** 1411–1428.

Clement, R. 2015. Mixed integer linear programming models for machine scheduling. Ph.D. thesis, The University of Newcastle, Australia.

Crama, Y., F.C.R. Spieksma. 1996. Scheduling jobs of equal length: complexity, facets and computational results. *Mathematical Programming* **72** 207–227.

Dolan, E.D., J.J. Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical Programming* **91** 201–213.

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

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

Houck, D.J.Jr., J.C. Picard, M. Queyranne, R.R. Vemuganti. 1978. Traveling salesman problem as a constrained shortest path problem: Theory and computational experience. Tech. Rep. EP-R-78-28, Ecole Polytechnique de Montréal.

Irnich, S., G. Desaulniers, J. Desrosiers, A. Hadjar. 2010. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing* **22** 297–313.

Irnich, S., D. Villeneuve. 2006. The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing* **18** 391–406.

Iwashita, H., S.I. Minato. 2013. Efficient top-down ZDD construction techniques using recursive specifications. Tech. Rep. TCS-TR-A-13-69, Hokkaido University, Graduate School of Information Science and Technology.

Kowalczyk, D., R. Leus, C. Hojny, S. Røpke. 2024. A flow-based formulation for parallel machine scheduling using decision diagrams. doi:10.1287/ijoc.2022.0301.cd. Available for download at `https://github.com/INFORMSJoC/2022.0301`.

Kramer, A., A. Subramanian. 2019. A unified heuristic and an annotated bibliography for a large class of earliness–tardiness scheduling problems. *Journal of Scheduling* **22** 21–57.

Lawler, E.L. 1977. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics* **1** 331–342.

Lee, C.Y. 1959. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal* **38** 985–999.

Linderoth, J.T., M.W.P. Savelsbergh. 1999. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing* **11** 173–187.

Minato, S.I. 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. *Proceedings of the 30th International Design Automation Conference*. DAC '93, ACM, New York, NY, USA, 272–277.

Oliveira, D., A. Pessoa. 2020. An improved branch-cut-and-price algorithm for parallel machine scheduling problems. *INFORMS Journal on Computing* **32** 90–100.

Pan, Y., L. Shi. 2007. 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.

Pessoa, A., R Sadykov, E. Uchoa, F. Vanderbeck. 2018. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing* **30** 339–360.

Pessoa, A., E. Uchoa, M. Poggi de Aragão, R. Rodrigues. 2010. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. *Mathematical Programming Computation* **2** 259–290.

Poggi de Aragão, M., E. Uchoa. 2003. Integer program reformulation for robust branch-and-cut-and-price algorithms. *Mathematical Programming in Rio: a Conference in Honour of Nelson Maculan*. 56–61.

Potts, C.N., L.N. Van Wassenhove. 1985. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research* **33** 363–377.

Queyranne, M., A.S Schulz. 1994. Polyhedral approaches to machine scheduling. Tech. Rep. 408/1994, Technical University of Berlin.

Sadykov, R., F. Vanderbeck. 2013. Column generation for extended formulations. *EURO Journal on Computational Optimization* **1** 81–115.

Sourd, F. 2009. New exact algorithms for one-machine earliness-tardiness scheduling. *INFORMS Journal on Computing* **21** 167–175.

Sousa, J.P., L.A. Wolsey. 1992. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming* **54** 353–367.

Tanaka, S., S. Fujikuma, M. Araki. 2009. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling* **12** 575–593.

van den Akker, J.M., J.A. Hoogeveen, S.L. van de Velde. 1999a. Parallel machine scheduling by column generation. *Operations Research* **47** 862–872.

van den Akker, J.M., C.A.J. Hurkens, M.W.P. Savelsbergh. 2000. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing* **12** 111–124.

van den Akker, J.M., C.P.M. Van Hoesel, M.W.P. Savelsbergh. 1999b. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming* **85** 541–572.

van Hoeve, W.J. 2022. Graph coloring with decision diagrams. *Mathematical Programming* **192** 631–674.

Wentges, P. 1997. Weighted Dantzig–Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research* **4** 151–162.

Acknowledgments

This work was partially funded by the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 754462.

Appendix A: Generation of the BDD (Section 4.2)

Algorithm 1 provides a recursive specification of the decision diagram that contains all the sequences for a given instance of $Pm||\sum w_jT_j$, as described in Section 4. The root of the BDD has configuration $(j_1^1, 0)$ (or

$(j_r^i, 0)$ for the first j_r^i with $p_{j_r^i} \in I_r$), and the function CHILD takes as input a configuration (j,t) of a node and $b \in \{0,1\}$ and outputs the configuration of the b-child of (j,t), where 0 and 1 refer to the low and high edge, respectively.

Algorithm 1: Recursive specification of the BDD

Function $CHILD((j_r^i, t), b)$

- **if** $b = 1$ **then**
  - $t' \leftarrow t + p_{j_r^i}$;
- **else**
  - $t' \leftarrow t$
- $j_{r'}^{i'} \leftarrow MINJOB(j_r^i, t')$;
- **if** $j_{r'}^{i'} = nq + 1$ **then**
  - **return** terminal node **1** ;
- **return** $(j_{r'}^{i'}, t')$;

Function $MINJOB(j_r^i, t)$

- **if** $\min\{j_{r'}^{i'} \succ j_r^i | t + p_{j_{r'}^i} \in I_{r'}\}$ *exists* **then**
  - **return** $\min\{j_{r'}^{i'} \succ j_r^i | t + p_{j_{r'}^{i'}} \in I_{r'}\}$;
- **return** $nq + 1$;

Appendix B: Illustration of a two-machine schedule for the example instance of Section 4.2

The BDD for the example instance depicted in Figure 3 is shown again in Figure 6, where we now include two paths in bold, which represent the machine schedules $(2,3)$ (in blue) and $(1,4)$ (in red). This constitutes an optimal schedule, which is different from the optimal solution described in Section 4.3. Job 3 can indeed be freely scheduled as the last job on any of the two machines; in either case its starting time is 6.

Appendix C: The ATIF and the BDDF are not comparable

We provide an example instance that shows that the polyhedron that represents the solution space of the linear relaxation of the formulation ATIF is not included in the polyhedron of the BDDF. Table 8 contains the job data for the instance with $n = 7$ jobs, and we work with $m = 2$ machines. The lower bound provided by the relaxation of the BDDF is $117.333\ldots$, while the bound for ATIF is $116.6777\ldots$ In our experiments discussed in Section 7.2 we encountered a number of instances where the LP relaxation of the ATIF provides a tighter bound than the BDDF; for brevity, we do not include such an instance here.

Figure 6 **The BDD of Figure 3 for the example instance, with two paths representing two machine schedules**

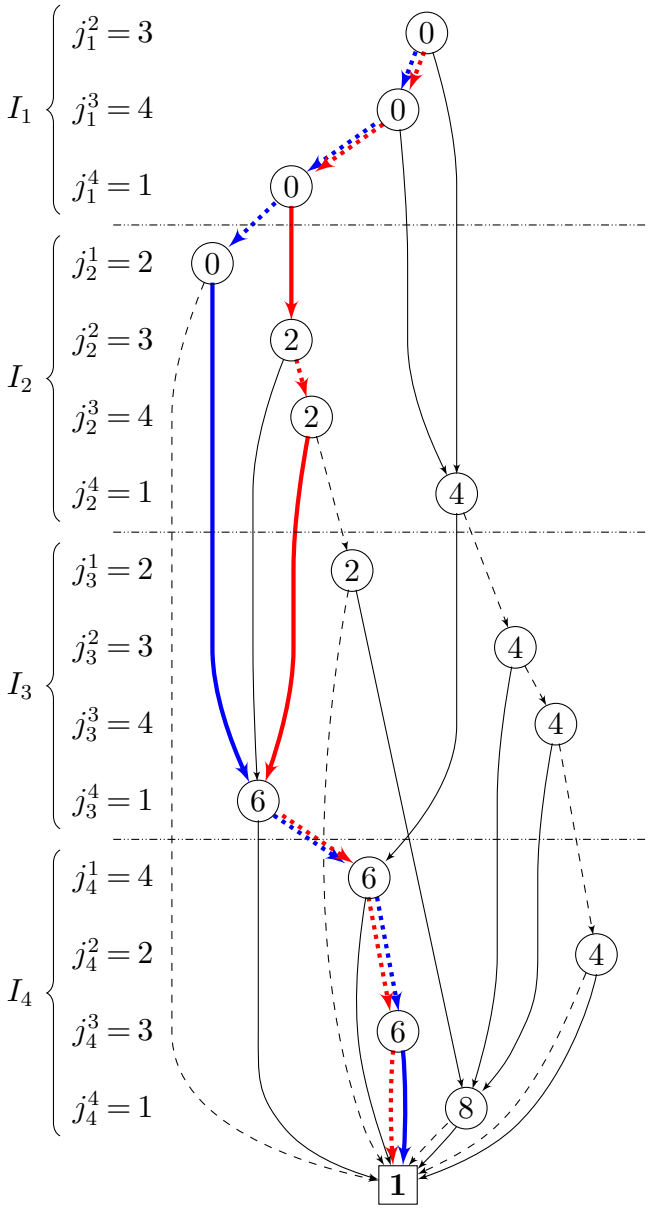


Table 8 **Job data for the example instance**

job j	p_j	d_j	w_j
1	92	197	5
2	30	114	6
3	47	86	6
4	19	155	1
5	65	136	5
6	78	158	6
7	82	95	1

Appendix D: Labeling algorithm (Section 5.3)

For each node v in the BDD (N, A) we define a bucket F_v that stores distance labels, representing lengths of partial paths that end in v. Unlike traditional labeling algorithms for shortest-path problems with resource constraints, we store only two labels in the bucket F_v for each node $v \in N$, namely the label $L_v^1 = (\bar{c}_v^1, v, pred_v^1)$ for a partial path P^1 leading to v with the best reduced cost $\bar{c}_v^1$, and the label $L_v^2 = (\bar{c}_v^2, v, pred_v^2)$ for a partial path P^2 with best reduced cost such that $j_{w^1} \neq j_{w^2}$, where $pred_v^1 = nil$ or contains a pointer to the label of the predecessor configuration $w^1 = (j_{w^1}, t_{w^1})$ of v in path P^1, and similarly $pred_v^2 = nil$ or a pointer to the label of predecessor $w^2 = (j_{w^2}, t_{w^2})$ in path P^2. The corresponding forward recursion in Algorithm 2 finds a path $P \in \mathcal{P}$ with minimum reduced cost such that all pairs of consecutive jobs on the machine are different. After the labeling algorithm, the bucket $F_{\mathbf{1}}$ associated with the terminal node **1** will hold two labels with paths from the root to the terminal node **1** with the smallest reduced cost and for which the associated pseudo-schedule is such that consecutive jobs are different. We can retrieve these pseudo-schedules by a simple backtracking algorithm via the pointers associated to the labels.

As mentioned in Section 5.3, the recursion can alternatively be conducted backwards. In this case the labels will have the structure $(\widetilde{c}_v, v, prev_v)$, with $prev_v = nil$ or a pointer to the previously chosen node (successor node) as part of the partial path, and two labels are stored in a bucket B_v for each node v, one for the best partial path, and one for a path with the best length but with different successor than the first label. The backwards labeling algorithm is described in pseudo-code in Algorithm 3.

Appendix E: Detailed computational results for the B&P procedure based on BDDF

In the following tables (Table 9 to 14) we include the detailed computational results of our B&P procedure based on BDDF for each tested instance. An instance is solved to guaranteed optimality if and only if $LB = UB$. The information provided in the different columns is as follows:

- $\#id$ = instance ID number
- UB = best found upper bound (best solution)
- LB *root* = lower bound in the root node of the B&B search tree
- LB = best found (global) lower bound
- $\#iter$ = total number of CG iterations across all nodes in the search tree
- $\#iter$ *root* =number of CG iterations in the root node
- $\#nodes$ = number of nodes in the B&B search tree (excluding the root node)
- *time LP* = CPU time (in seconds) for solving the LPs in all nodes
- *time LP root* = CPU time (in seconds) for solving the LP in the root node
- *time* = total CPU time (in seconds)

The CPU times were transformed to be comparable with those obtained by the machine used by Oliveira and Pessoa (2020), where 7200 seconds on our computer equates with approximately 5040 seconds on the computer of Oliveira and Pessoa.

Algorithm 2: Forward labeling algorithm for pricing

Data: BDD (N, A), optimal solution π of the dual program

Result: Pseudo-schedule s with minimum negative reduced cost

$L_v^1 \leftarrow (\infty, v, nil)$ and $L_v^2 \leftarrow (\infty, v, nil)$, $\forall v \in N \setminus \{\mathbf{r}\}$;

$L_{\mathbf{r}}^1 \leftarrow (-\pi_0, \mathbf{r}, nil)$ and $L_{\mathbf{r}}^2 \leftarrow (\infty, \mathbf{r}, nil)$;

for $v \in N \setminus \{\mathbf{1}\}$, *with* $v = (j_v, t_v)$, *in breadth-first order* **do**

Let $lo(v)$ be the low child of v and $hi(v)$ be the high child of v ;

$L_v \leftarrow$ best label in F_v for which the predecessor job j_w is different from j_v ;

Extend label L_v to a label in $F_{hi(v)}$ if the reduced cost of the new label is smaller than the current reduced cost of the label, and the labels are constructed in such a way that the predecessors point to nodes associated with different jobs ;

Extend the labels L_v^1 and L_v^2 to labels in $F_{lo(v)}$ if the reduced cost of the new labels is smaller than the reduced cost of the current labels in $F_{lo(v)}$;

Algorithm 3: Backward labeling algorithm for pricing

Data: BDD (N, A), optimal solution π of the dual program

Result: Pseudo-schedule s with minimum negative reduced cost

$L_v^1 \leftarrow (\infty, v, nil)$ and $L_v^2 \leftarrow (\infty, v, nil)$, $\forall v \in N \setminus \{\mathbf{1}\}$;

$L_{\mathbf{1}}^1 \leftarrow (-\pi_0, \mathbf{1}, nil)$ and $L_{\mathbf{1}}^2 \leftarrow (\infty, \mathbf{1}, nil)$;

for $v \in N \setminus \{\mathbf{1}\}$, *with* $v = (j_v, t_v)$, *in reversed breadth-first order* **do**

First extend the labels of $B_{lo(v)}$ to labels in B_v ;

Let L_v be a candidate label of B_v that is extended from one of labels of $B_{hi(v)}$ such that j_v is different from the job associated to the pointer of label of one of the labels in $hi(v)$;

If the reduced cost of label L_v is smaller than the current reduced costs of the labels in B_v then update the labels in B_v appropriately ;

Table 9 B&P results per instance for $m = 2$ and $n = 40$

#id	*UB*	*LB root*	*LB*	*#iter*	*#iter root*	*#nodes*	*time LP*	*time LP root*	*time*
1	606	584	606	484	201	2	3.16	0.65	4.40
6	3,886	3,875	3,886	345	220	2	1.71	0.72	3.83
11	9,617	9,592	9,617	303	237	1	1.13	0.66	2.04
16	38,356	38,277	38,356	4,293	336	10	26.77	1.07	69.35
21	41,048	41,048	41,048	477	477	0	1.22	1.22	1.23
26	87	87	87	151	151	0	0.49	0.49	0.49
31	3,812	3,758	3,812	976	225	3	10.07	1.05	13.76
36	10,713	10,660	10,713	7,232	329	29	31.65	1.42	64.10
41	30,802	30,798	30,802	429	413	1	2.15	1.81	4.22
46	34,146	34,146	34,146	565	565	0	1.49	1.49	1.50
51	0	0	0	1	1	0	0.04	0.04	0.04
56	1,279	1,272	1,279	307	211	1	1.56	0.71	1.96
61	11,488	11,302	11,488	8,546	338	18	52.81	1.29	110.45
66	35,279	35,130	35,279	13,731	364	35	89.80	2.19	229.61
71	47,952	47,935	47,952	739	480	2	5.12	1.93	14.62
76	0	0	0	1	1	0	0.02	0.02	0.03
81	571	451	571	1,785	288	8	19.95	1.69	27.62
86	6,048	5,996	6,048	1,216	407	3	6.05	2.49	11.43
91	26,075	26,075	26,075	482	482	0	2.37	2.37	2.38
96	66,116	66,110	66,116	662	513	1	2.97	2.30	4.61
101	0	0	0	1	1	0	0.02	0.02	0.03
106	0	0	0	1	1	0	0.03	0.03	0.03
111	17,936	17,897	17,936	2,069	400	7	11.66	2.14	31.09
116	25,870	25,764	25,870	4,007	366	8	21.32	2.05	51.32
121	64,516	64,507	64,516	573	460	2	3.30	2.26	7.00

Table 10 **B&P results per instance for $m = 4$ and $n = 40$**

#id	*UB*	*LB root*	*LB*	*#iter*	*#iter root*	*#nodes*	*time LP*	*time LP root*	*time*
1	439	438	439	186	140	1	0.25	0.17	0.36
6	2,374	2,372	2,374	220	194	1	0.67	0.41	1.18
11	5,737	5,735	5,737	208	189	1	0.42	0.27	0.78
16	21,493	21,484	21,493	1,442	227	17	6.04	0.97	13.41
21	22,793	22,793	22,793	318	318	0	0.46	0.46	0.46
26	88	88	88	113	113	0	0.21	0.21	0.21
31	2,525	2,496	2,525	342	157	1	1.32	0.41	2.05
36	6,420	6,355	6,420	1,044	214	4	5.29	0.43	10.20
41	17,685	17,633	17,685	615	193	2	4.31	1.07	8.97
46	19,124	19,124	19,124	268	268	0	0.75	0.75	0.76
51	0	0	0	1	1	0	0.01	0.01	0.01
56	826	798	826	588	121	5	2.75	0.23	4.42
61	7,357	7,315	7,357	2,738	214	37	15.30	1.28	39.04
66	20,251	20,247	20,251	317	303	1	1.39	1.21	1.97
71	26,740	26,740	26,740	381	381	0	0.99	0.99	1.00
76	0	0	0	1	1	0	0.01	0.01	0.01
81	564	540	564	578	171	8	2.04	0.35	3.79
86	4,725	4,719	4,725	279	216	2	1.42	0.89	2.28
91	15,569	15,557	15,569	392	254	3	2.30	0.82	4.83
96	36,266	36,266	36,266	330	330	0	0.77	0.77	0.77
101	0	0	0	1	1	0	0.01	0.01	0.02
106	0	0	0	1	1	0	0.01	0.01	0.01
111	11,263	11,212	11,263	926	146	5	4.39	0.90	8.08
116	15,566	15,539	15,566	1,113	252	3	4.99	0.74	9.00
121	35,751	35,739	35,751	897	301	6	4.96	0.98	11.29

Table 11 **B&P results per instance for $m = 2$ and $n = 50$**

#id	*UB*	*LB root*	*LB*	*#iter*	*#iter root*	*#nodes*	*time LP*	*time LP root*	*time*
1	1,268	1,232	1,268	935	220	3	15.37	1.49	19.68
6	14,272	14,261	14,272	627	445	2	5.51	2.42	10.55
11	23,028	23,000	23,028	685	427	1	4.59	1.84	8.69
16	46,072	46,011	46,072	8,638	501	20	50.85	2.67	159.61
21	111,069	111,067	111,069	721	626	1	4.11	3.50	7.76
26	26	26	26	244	244	0	1.51	1.51	1.52
31	5,378	5,289	5,378	1,692	367	6	34.63	2.21	49.74
36	18,956	18,891	18,956	1,574	441	4	12.40	2.93	27.69
41	38,058	37,952	38,058	2,584	524	5	23.06	3.57	62.93
46	82,105	82,084	82,105	3,617	600	4	18.41	3.62	51.26
51	0	0	0	1	1	0	0.04	0.04	0.04
56	761	730	761	622	242	2	8.54	1.72	11.76
61	13,682	13,582	13,682	3,238	402	5	27.69	2.94	49.80
66	40,907	40,907	40,907	781	781	0	4.11	4.11	4.12
71	78,532	78,532	78,532	554	554	0	3.28	3.28	3.29
76	0	0	0	1	1	0	0.05	0.05	0.06
81	542	538	542	575	434	1	5.24	3.59	6.30
86	12,557	12,267	12,557	22,169	547	44	99.21	4.31	287.87
91	47,349	47,293	47,349	3,846	475	10	28.22	3.36	86.36
96	92,822	92,801	92,822	1,258	595	5	9.33	4.01	26.22
101	0	0	0	1	1	0	0.04	0.04	0.05
106	0	0	0	1	1	0	0.05	0.05	0.06
111	15,564	15,543	15,564	1,009	517	3	5.06	3.83	8.47
116	19,608	19,520	19,608	10,005	495	19	48.65	3.99	133.97
121	41,696	41,696	41,696	800	800	0	3.36	3.36	3.36

Table 12 **B&P results per instance for $m = 4$ and $n = 50$**

#id	*UB*	*LB root*	*LB*	*#iter*	*#iter root*	*#nodes*	*time LP*	*time LP root*	*time*
1	785	777	785	342	173	3	2.15	0.64	4.04
6	8,317	8,298	8,317	481	298	2	2.51	0.73	4.69
11	12,879	12,871	12,879	404	292	2	3.73	1.13	8.02
16	25,376	25,376	25,376	338	338	0	1.51	1.51	1.51
21	59,440	59,440	59,440	450	450	0	1.39	1.39	1.39
26	54	54	54	182	182	0	0.48	0.48	0.48
31	3,061	3,061	3,061	243	243	0	0.74	0.74	0.75
36	10,796	10,794	10,796	288	274	1	1.64	1.27	2.48
41	21,806	21,783	21,806	1,865	370	4	13.01	1.67	28.24
46	44,455	44,452	44,455	448	344	2	4.09	2.77	6.52
51	0	0	0	1	1	0	0.01	0.01	0.01
56	570	538	570	689	180	4	3.57	0.45	5.42
61	7,898	7,850	7,898	8,417	263	37	74.56	2.80	146.32
66	23,138	23,138	23,138	495	495	0	2.81	2.81	2.83
71	42,645	42,625	42,645	2,652	361	9	23.32	2.22	56.51
76	0	0	0	1	1	0	0.02	0.02	0.03
81	495	478	495	282	219	1	1.47	0.77	1.66
86	8,369	8,328	8,369	1,444	321	3	11.76	2.65	24.22
91	26,551	26,546	26,551	463	360	1	4.48	2.90	7.37
96	50,326	50,312	50,326	1,006	321	5	14.07	2.24	36.07
101	0	0	0	1	1	0	0.02	0.02	0.02
106	0	0	0	1	1	0	0.02	0.02	0.02
111	10,069	10,047	10,069	955	313	7	12.21	2.90	29.11
116	11,552	11,519	11,552	1,655	268	7	18.29	2.82	43.54
121	23,792	23,768	23,792	1,371	293	4	11.39	1.97	24.50

Table 13 **B&P results per instance for $m = 2$ and $n = 100$**

#id	*UB*	*LB root*	*LB*	*#iter*	*#iter root*	*#nodes*	*time LP*	*time LP root*	*time*
1	3,339	3,314	3,339	2,999	549	5	104.77	11.60	156.18
6	30,665	30,644	30,665	9,784	840	13	403.82	21.12	855.91
11	93,894	93,894	93,894	1,134	1,134	0	17.40	17.40	17.42
16	209,100	209,057	209,100	16,470	1,414	8	206.91	31.48	816.28
21	457,836	457,814	457,836	16,402	1,873	10	248.53	28.16	1,096.78
26	92	92	92	391	391	0	8.43	8.43	8.45
31	12,729	12,725	12,729	940	724	1	27.71	19.02	43.98
36	56,671	56,574	56,671	15,383	1,056	17	495.50	29.59	1,435.73
41	237,964	237,770	237,906	40,421	1,395	24	834.62	36.01	5,086.75
46	422,831	422,804	422,831	41,848	1,612	25	666.84	36.06	3,511.15
51	0	0	0	1	1	0	0.14	0.14	0.16
56	5,047	4,983	5,047	2,309	683	4	57.09	15.15	91.75
61	45,573	45,411	45,473	56,982	1,033	34	1,275.14	19.90	5,021.28
66	126,513	126,405	126,430	71,940	1,126	45	1,003.64	25.51	5,070.91
71	327,305	327,300	327,305	2,650	1,905	2	60.02	48.01	154.00
76	0	0	0	1	1	0	0.14	0.14	0.16
81	908	791	908	3,935	1,523	5	110.89	50.68	146.54
86	36,686	36,206	36,209	163,328	1,360	297	1,318.04	46.00	4,981.33
91	129,929	129,588	129,802	52,557	1,259	24	945.45	38.14	5,021.95
96	254,194	254,140	254,145	42,980	1,539	23	608.86	32.32	5,128.69
101	0	0	0	1	1	0	0.24	0.24	0.28
106	0	0	0	1	1	0	0.23	0.23	0.26
111	84,220	84,003	84,055	53,064	1,468	26	909.90	58.55	5,075.99
116	191,205	191,085	191,107	77,643	1,465	41	832.52	36.03	5,039.19
121	242,019	241,954	241,957	67,337	1,706	36	808.76	54.49	5,021.20

Table 14 **B&P results per instance for $m = 4$ and $n = 100$**

#id	*UB*	*LB root*	*LB*	*#iter*	*#iter root*	*#nodes*	*time LP*	*time LP root*	*time*
1	2,001	1,990	2,001	708	393	3	14.54	4.03	26.54
6	16,893	16,893	16,893	631	631	0	7.54	7.54	7.56
11	50,234	50,196	50,216	101,050	710	393	1,677.41	9.06	4,943.34
16	110,219	110,104	110,219	34,870	713	57	971.12	7.52	2,905.22
21	237,392	237,389	237,392	1,731	1,146	1	30.26	20.16	51.67
26	141	141	141	378	378	0	5.32	5.32	5.34
31	7,130	7,080	7,130	8,102	473	23	442.34	5.81	712.41
36	30,791	30,773	30,791	1,053	696	1	24.25	9.66	45.38
41	126,185	126,130	126,185	10,125	690	22	254.39	14.51	914.88
46	219,536	219,526	219,536	3,448	1,017	4	59.95	17.04	190.07
51	0	0	0	1	1	0	0.10	0.10	0.11
56	3,076	3,021	3,076	2,873	522	7	64.60	8.18	94.04
61	24,863	24,805	24,825	134,520	742	415	1,586.59	7.92	4,976.25
66	67,967	67,947	67,967	39,896	840	119	673.85	17.34	2,242.08
71	170,694	170,674	170,675	70,736	1,109	104	1,233.12	24.32	5,012.55
76	0	0	0	1	1	0	0.10	0.10	0.11
81	819	754	819	1,417	844	3	24.33	14.35	28.48
86	21,282	21,202	21,256	75,623	636	136	1,406.58	21.54	5,025.99
91	70,606	70,582	70,606	35,852	919	74	615.14	15.18	2,357.56
96	133,587	133,571	133,587	24,286	995	42	231.11	26.19	751.82
101	0	0	0	1	1	0	0.13	0.13	0.15
106	0	0	0	1	1	0	0.10	0.10	0.12
111	46,705	46,616	46,658	56,541	776	67	1,223.29	11.38	5,016.42
116	101,571	101,514	101,517	50,825	1,004	86	1,281.40	29.08	5,028.39
121	127,618	127,593	127,618	30,859	1,018	60	521.97	23.93	1,866.95