KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching

Kowalczyk D, Leus R.

KBI_1631

A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching

Daniel Kowalczyk, Roel Leus
ORSTAT, Faculty of Economics and Business, KU Leuven, Naamsestraat 69, 3000 Leuven, Belgium
daniel.kowalczyk@kuleuven.be, roel.leus@kuleuven.be

We study the parallel machine scheduling problem to minimize the sum of the weighted completion times of the jobs to be scheduled (problem $Pm||\sum w_jC_j$ in the standard three-field notation). We use the set covering formulation that was introduced by van den Akker et al. (1999) for this problem, and we improve the computational performance of their branch-and-price (B&P) algorithm by a number of techniques, including a different generic branching scheme, zero-suppressed binary decision diagrams (ZDDs) to solve the pricing problem, dual-price smoothing as a stabilization method, and Farkas pricing to handle infeasibilities. We report computational results that show the effectiveness of the algorithmic enhancements, which depends on the characteristics of the instances. To the best of our knowledge, we are also the first to use ZDDs to solve the pricing problem in a B&P algorithm for a scheduling problem.

Key words: parallel machine scheduling, weighted completion times, branch and price, ZDD, stabilization

1. Introduction

In this paper we report on our improvements to the branch-and-price (B&P) algorithm of van den Akker et al. (1999) for the parallel machine scheduling problem with weighted completion-time objective on identical machines, which is written as $Pm||\sum w_jC_j$ in the standard three-field notation. For brevity we refer to the problem as WCT. A B&P algorithm is a branch-and-bound (B&B) algorithm in which at every node of the B&B tree a lower bound is computed via a linear-programming (LP) relaxation with an exponential number of variables. The LP relaxation, called the *master problem*, is solved using column generation (CG). B&P algorithms are used in many areas of operations research, such as vehicle routing, vertex coloring, bin packing, etc. In a given node of the search tree, the master problem only contains a restricted number of promising variables and is called the *restricted* master problem (RMP). The RMP is solved with standard LP techniques. It can, but need not, yield an optimal solution to the full master problem with all columns included. The search for a new column with negative reduced cost to be included in RMP is called the *pricing problem*, which is an optimization problem that depends on the definition

of the variables, and which has a non-linear objective function in our case. In van den Akker et al. (1999) the authors solve the pricing problem using a general dynamic-programming (DP) recursion. In our work, we use a *zero-suppressed binary decision diagram* (ZDD) to do this, which is a data structure that represents a family of sets. Concretely, a ZDD is constructed as a directed acyclic graph (DAG) such that each machine schedule corresponds with a path from the root node of the DAG to one particular leaf node, which will allow to find a schedule with minimum (negative) reduced cost. Solving pricing problems with the help of ZDDs was first done in Morrison et al. (2016b), who show how to adjust the ZDD when a standard integer branching scheme is applied for vertex coloring (when fractional variable λ equals α, then two children are created with additional constraints $\lambda \leq \lfloor \alpha \rfloor$ and $\lambda \geq \lceil \alpha \rceil$). With standard integer branching, the structure of the pricing problem is typically destroyed, and so van den Akker et al. (1999) devise a specialized branching scheme that allows to re-use the same pricing algorithm in every node of the B&B tree. In this paper we explain how to adapt the ZDD when the generic branching scheme of Ryan and Foster (1981) is used, which will have a clear impact on the convergence of the B&P algorithm, meaning that the algorithm will explore less nodes in the search tree.

Another improvement that we include in the B&P algorithm is the use of stabilization techniques for CG. It is well known that CG methods for machine scheduling suffer from poor convergence because of extreme primal degeneracy and alternative dual solutions. These phenomena have a large impact especially when the number of jobs per machine is high. One of these techniques is *dual-price smoothing*, which was introduced in Wentges (1997). This technique corrects the optimal solution of the dual problem of the restricted LP relaxation based on past information before it is plugged into the pricing algorithm. This stabilization is very important for calculating the lower bound, but identifying upper bounds is obviously also of vital importance. In our computational experiments we will see that the branching scheme of van den Akker et al. (1999) performs rather poorly on some instance classes, even when stabilization is applied, and then the choice of the branching scheme matters. As pointed out by Morrison et al. (2016b), using ZDDs for pricing gives the possibility to implement a generic B&P algorithm.

The remainder of this paper is structured as follows. In Section 2 we first provide a formal problem statement and a number of pointers to the existing literature. Subsequently, in Section 3 we describe some characteristics of optimal solutions and give an integer linear

formulation. The formulation is solved by means of B&P, the main aspects of which are explained in Section 4. Section 5 then provides more information on ZDDs in general, and on the way in which we use ZDDs to solve the pricing problem. More details on Farkas pricing as a way to handle infeasibilities, on stabilization and on the branching strategy are given in Sections 6, 7 and 8, respectively. The main findings of our computational experiments are discussed in Section 9, and we conclude the paper in Section 10.

2. Problem definition and literature overview

We consider a set $J = \{1, \ldots, n\}$ of n independent jobs with associated processing times $p_j \in \mathbb{N}_0$ for $j \in J$, which need to be processed on a set $M = \{1, \ldots, m\}$ of m identical machines. Each machine can process at most one job at a time and preemption is not allowed. The objective of problem WCT is to find an assignment of the jobs to the machines and to decide the sequencing of the jobs on the machines such that the objective function $\sum_{j=1}^{n} w_j C_j$ is minimized, where $w_j \in \mathbb{N}_0$ is the weight of job $j \in J$ and C_j the completion time. This problem is $\mathcal{NP}$-hard for $m \geq 2$, see for instance Bruno et al. (1974) or Lenstra et al. (1977). We assume that $n > m$ in order to avoid trivialities. Case $m = 1$ (only one machine) is known to be solvable in polynomial time, because one can show that there exists an optimal schedule with the following two properties:

PROPERTY 1. *The jobs are sequenced following Smith's shortest weighted processing-time (SWPT) rule (Smith, 1956), which orders the jobs j in non-increasing order of the ratio $\frac{w_j}{p_j}$.*

PROPERTY 2. *The jobs are processed contiguously from time zero onward.*

Since the machines can be considered independently of each other after the assignment of the jobs to machines, an optimal solution exists for WCT that has the same properties; the difficulty of the problem therefore resides in the job-machine assignment. Several DP approaches have been proposed for WCT that exploit this idea. These algorithms run in $O(n(\sum_{j \in J} p_j)^{m-1})$ time (Rothkopf, 1966; Lawler and Moore, 1969), or in $O(n(\sum_{j \in J} w_j)^{m-1})$ time (Lee and Uzsoy, 1992). Consequently, these algorithms become unmanageable when the number of machines, the processing times or the weights increase.

In the literature, B&P methods have been proposed that exploit Properties 1 and 2. Chen and Powell (1999) and van den Akker et al. (1999) devise an algorithm with CG-based lower bound resulting from a partition formulation of WCT, where a variable corresponds

to a single machine schedule with a specific structure (see Section 3). The algorithms differ in their branching scheme, as well as in the pricing algorithm (although both are DP-based). It should be noted that the lower bounds are very tight.

B&P approaches for parallel machine scheduling do not always exploit Property 1, or another optimal ordering rule. Dyer and Wolsey (1990) introduce a time-indexed formulation for a problem with a general time-dependent objective function. The number of variables is $O(nT)$, where T depends on the processing times. This type of formulation has very good bounds but cannot be applied directly because of the pseudo-polynomial number of variables and constraints. This problem is partially mitigated in van den Akker et al. (2000) by applying Dantzig-Wolfe decomposition, and then CG is used for computing the lower bound. The pricing problem is a shortest path problem on a network of size $O(nT)$. Branching can be applied on the original variables of the time-indexed formulation without much effort, i.e., the pricing algorithm stays the same, only the network in which a shortest path is calculated changes.

Another model is the arc-time-indexed formulation proposed by Pessoa et al. (2010), where a binary variable z_{ijt} is defined for every pair of jobs (i, j) and every period t such that $z_{ijt} = 1$ if job i finishes and job j starts at t. The resulting formulation is huge, but also has some advantages over the time-indexed model. Dantzig-Wolfe decomposition is then applied to obtain a reformulation with an exponential number of variables, each corresponding to a path in a suitable network. The running time of the pricing algorithm is $O(n^2T)$. Pessoa et al. (2010) recognize that stabilization techniques are important to quickly compute the lower bound and overcome the convergence issue for this type of formulation.

Other techniques to resolve the slow convergence of B&P for time-indexed formulations have been proposed by Bigras et al. (2008), who use temporal decomposition, and by Sadykov and Vanderbeck (2013), who put forward the technique of column-and-row generation together with stabilization. Note that, depending on the instance, the duality gap for these formulations may be difficult to close, and so the B&B tree can still be very large.

3. Structure of optimal solutions and linear formulation

We assume that the jobs are indexed in non-increasing order of the ratio $\frac{w_j}{p_j}$, i.e., $\frac{w_1}{p_1} \geq \cdots \geq \frac{w_n}{p_n}$, so that Property 1 is easily invoked for each machine separately. Denote

by $p_{[i]}$ the $[i]$-th smallest processing time among the jobs in J, and let $p_{\max} = p_{[n]}$ be the largest value among all p_j. We can now state some further properties of optimal solutions to WCT, with S_j the starting time of job j:

PROPERTY 3 **(Belouadah and Potts, 1994)**. *There exists an optimal solution for which the latest completion time on any machine is not greater than* $H_{\max} = \frac{\sum_{j\in J} p_j}{m} + \frac{m-1}{m} p_{\max}$.

PROPERTY 4 **(Azizoglu and Kirca, 1999)**. *There exists an optimal solution for which the latest completion time on any machine is not less than* $H_{\min} = \frac{1}{m}\left(\sum_{j\in J} p_j - \sum_{h=1}^{m-1} p_{[n-h+1]}\right)$.

PROPERTY 5 **(Elmaghraby and Park, 1974)**. *$S_{j_1} \leq S_{j_2}$ in an optimal solution if one of the following conditions holds:*

- *$p_{j_1} < p_{j_2}$ and $w_{j_1} \geq w_{j_2}$, or*
- *$p_{j_1} \leq p_{j_2}$ and $w_{j_1} > w_{j_2}$.*

PROPERTY 6 **(Azizoglu and Kirca, 1999)**. *$S_{j_1} \leq S_{j_2}$ in an optimal solution if*

$$\sum_{h=1}^{j_1-1} p_h \leq \frac{1}{m}\left(\sum_{j\in J} p_j - \sum_{h=1}^{m-1} p_{[n-h+1]}\right) - \sum_{h=j_2}^{n} p_h.$$

These properties have consequences for the execution intervals of the jobs. The execution interval of job j is described by a release date r_j, before which the job cannot be started, and a deadline $\overline{d}_j$, by which the job has to be completed. By Property 3 we can set $r_j = 0$ and $\overline{d}_j = H_{\max}$ for every $j \in J$. In van den Akker et al. (1999) the authors use only Property 5 to deduce tighter release dates and deadlines for every job. In this text we will also use Property 6; this can have a great influence on the tightness of the time windows.

Define the following subsets of jobs for each $j \in J$:

$$\mathcal{P}_j^1 = \{k \in J \mid (w_k > w_j \wedge p_k \leq p_j) \vee (w_k \geq w_j \wedge p_k < p_j)\} \tag{1}$$

and

$$\mathcal{P}_j^2 = \left\{ k \in J \;\middle|\; k < j \text{ and } \sum_{h=1}^{k-1} p_h \leq \frac{1}{m}\left(\sum_{h=1}^{n} p_h + \sum_{h=1}^{m-1} p_{[n-h+1]}\right) - \sum_{h=j}^{n} p_h \right\}. \tag{2}$$

Let $\mathcal{P}_j$ be the union of $\mathcal{P}_j^1$ and $\mathcal{P}_j^2$. Because of Properties 5 and 6 we know that there exists an optimal solution in which all jobs of $\mathcal{P}_j$ start no later than job j. Hence if $|\mathcal{P}_j| > m-1$,

then we know that there are at least $|\mathcal{P}_j| - m + 1$ jobs that should be completed before job j is started. Denote by ρ_j the sum of the durations of the $|\mathcal{P}_j| - m + 1$ jobs in $\mathcal{P}_j$ with smallest processing time, then we can set $r_j = \left\lceil \frac{\rho_j}{m} \right\rceil$. The derivation of tighter deadlines can proceed similarly: define

$$\mathcal{Q}_j^1 = \{k \in J \mid k > j,\ w_k \leq w_j \text{ and } p_k \geq p_j\} \tag{3}$$

and

$$\mathcal{Q}_j^2 = \left\{ k \in J \;\middle|\; k > j \text{ and } \sum_{h=1}^{j-1} p_h \leq \frac{1}{m} \left(\sum_{h=1}^{n} p_h + \sum_{h=1}^{m-1} p_{[n-h+1]} \right) - \sum_{h=k}^{n} p_h \right\}. \tag{4}$$

Let $\mathcal{Q}_j$ be the union of $\mathcal{Q}_j^1$ and $\mathcal{Q}_j^2$. Similarly as before, there exists an optimal solution for which $S_j \leq S_{j'}$ for every $j' \in \mathcal{Q}_j$, so the amount of work that has to be done between S_j and $H_{\max}$ is equal to $\sum_{j' \in \mathcal{Q}_j} p_{j'} + p_j$. Consequently, job j cannot start later than $\delta_j = H_{\max} - \left\lceil \left(\sum_{j' \in \mathcal{Q}_j} p_{j'} + p_j \right) / m \right\rceil$, and if $\delta_j + p_j < \overline{d}_j$ then we update $\overline{d}_j$ to value $\delta_j + p_j$.

Below we describe an integer linear formulation for WCT. Every binary variable λ_s corresponds to an assignment of a subset $s \subset J$ of jobs to one machine, with which we can associate a unique schedule via $C_j(s) = \sum_{i \in s : i \leq j} p_i$ for each $j \in s$. We only consider job sets s that respect the execution intervals corresponding to the ready times and deadlines that where established above. Concretely, let $\mathcal{S}$ be the set of all sets $s \subset J$ that lead to a *feasible* schedule, meaning that $r_j + p_j \leq C_j(s) \leq \overline{d}_j$ for all $j \in s$, and that the completion time of the last job included in s is between $H_{\min}$ and $H_{\max}$. We should note that van den Akker et al. (1999) use a weaker form of Property 4 for computing $H_{\min}$. In the formulation below we represent such schedules by binary vectors of dimension n. Every schedule s has an associated cost

$$c_s = \sum_{j \in s} w_j \left(\sum_{k \in s : k \leq j} p_k \right). \tag{5}$$

We now need to find m schedules of $\mathcal{S}$ such that every job $j \in J$ is chosen and the total weighted completion time is minimized. This can be cast into the following set covering formulation:

$$\text{minimize} \quad \sum_{s \in \mathcal{S}} c_s \lambda_s \tag{6a}$$

$$\text{subject to} \quad \sum_{s \in \mathcal{S} : j \in s} \lambda_s \geq 1 \qquad \text{for each } j \in J \tag{6b}$$

$$\sum_{s \in \mathcal{S}} \lambda_s \leq m \tag{6c}$$

$$\lambda_s \in \{0,1\} \qquad \text{for each } s \in \mathcal{S} \tag{6d}$$

This formulation has n covering (assignment) constraints and one capacity constraint. Condition (6b) ensures that every job j is assigned to one machine, constraints (6c) impose that we use only m machines, and constraints (6d) state that the variables are all binary.

Although it might be more intuitive to write equality in (6b) and (6c), resulting in a partition formulation, our experimental results indicate that the covering formulation performs better than the partition formulation in the LP relaxation phase. The reason is that the dual variables corresponding to the equality sign in the constraints (6b) and (6c) of the LP relaxation of the partition formulation are less constrained. The LP relaxation of the set covering formulation has faster convergence and is more stable than the relaxation of the partition formulation. Similar observations are reported in Lopes and de Carvalho (2007).

4. A B&P algorithm for WCT

Formulation (6) has an exponential number of variables. Listing all the schedules of $\mathcal{S}$ would be impractical and we will therefore devise a B&P algorithm. The main differences between the algorithm of van den Akker et al. (1999) and our implementation are the following:

1. We use a generic branching scheme that was introduced by Ryan and Foster (1981) for partitioning formulations. In van den Akker et al. (1999) the authors develop a problem-specific branching scheme that does not destroy the structure of the pricing problem.
2. Since our branching scheme does affect the structure of the pricing problem in the root node, we develop a different pricing algorithm that is based on ZDDs. The details of this pricing routine are described in Section 5. To the best of our knowledge, we are the first to use ZDDs to solve the pricing problem in a B&P algorithm for a scheduling problem.
3. When a new branching decision is made in the search tree, it is possible that the RMPs in the newly created child nodes are infeasible. We handle such infeasibility by applying Farkas pricing; we refer to Section 6 for more details.

4. We also use a stabilization technique in order to manage several drawbacks inherent in the use of CG; see Section 7. This will be very important for calculating the lower bound in the root node, in particular for instances with many jobs per machine.

We first seek to solve the LP relaxation of (6), which corresponds with the objective function (6a), the constraints (6b) and (6c), and

$$\lambda_s \geq 0 \qquad \text{for each } s \in \mathcal{S} \tag{7}$$

The dual of this LP relaxation is given by:

$$\text{maximize} \sum_{j \in J} \pi_j - m\sigma \tag{8a}$$

$$\text{subject to} \sum_{j \in s} \pi_j - \sigma \leq c_s \qquad \text{for each } s \in \mathcal{S} \tag{8b}$$

$$\pi_j \geq 0 \qquad \text{for each } j \in J \tag{8c}$$

$$\sigma \geq 0 \tag{8d}$$

where values π_j for $j \in J$ are the dual variables associated to constraints (6b) and σ is the dual variable associated to constraint (6c). The constraints (8b) derive from the primal variables λ. The LP relaxation is solved by CG, so we iteratively solve a RMP instead of the full relaxation and check whether there exists a column that can be added to improve the current optimal solution, which is done in the pricing problem. The columns are schedules from a restricted set $\mathcal{S}_r \subset \mathcal{S}$. At each iteration we check whether constraint (8b) is violated, and if so then we add the corresponding primal variable. The termination conditions for this CG procedure depend on the stabilization technique, and will be explained in Section 7.

The pricing problem is the following: if π_j^* for $j \in J$ and σ^* represent the current optimal solution of the dual of the RMP, then does there exist a schedule $s \in \mathcal{S}$ for which $\sum_{j \in s} \pi_j^* - \sigma^* > c_s$? The LP relaxation is typically solved faster if we first add constraints that are strongly violated, and we will search for schedules with most negative reduced cost. This can be modeled as follows:

$$\text{minimize } c_s - \sum_{j \in s} \pi_j^* \tag{9a}$$

$$\text{subject to } s \in \mathcal{S}, \tag{9b}$$

because σ^* is independent of $\mathcal{S}$. Note that c_s in the objective function (9a) is quadratic (see Equation (5)). This pricing problem is solved by DP in van den Akker et al. (1999). Their forward recursion uses the insight from Property 1 that each machine schedule can follow the SWPT rule. The DP-based algorithm runs in $O(nH_{\max})$ time. In the next section, we will show how ZDDs can be used in a pricing algorithm.

5. Zero-suppressed binary decision diagrams for the pricing problem

5.1. General introduction to ZDDs

Zero-suppressed binary decision diagrams (ZDDs) are data structures that allow to represent and manipulate families of sets that can be linearly ordered in a useful manner. They were proposed by Minato (1993) as extensions of binary decision diagrams (BDDs). BDDs were introduced by Lee (1959) and Akers (1978) as DAGs that can be obtained by reducing binary decision trees that represent the decision process through a set of binary variables of a Boolean function. A ZDD Z is a DAG that has two terminal nodes, which are called the **1**-node and **0**-node. Every nonterminal node i is associated to an element $v(i)$ (the "label" of node i) of a set and has two outgoing edges. One edge is called the high edge and points to a node $\text{hi}(i)$, which is called the high child of the node. The other edge is called the low edge and points to the low child node $\text{lo}(i)$. The label associated to any nonterminal node is strictly smaller than the labels of its children, i.e. for every node i of Z we have that $v(i) < v(\text{hi}(i))$ and $v(i) < v(\text{lo}(i))$. There is also exactly one node that is not a child of any other node in the DAG; this node is the "highest" node in the topological ordering of the DAG and is called the root node. For both types of decision diagrams (DDs) the size can be reduced by merging nodes with the same label (so associated to the same element of the set) and the same low and high child, but ZDDs entail an extra reduction process. In a ZDD, every node with a high edge pointing to the terminal node **0** is deleted.

Let us describe how a subset A of a ground set N induces a path P_A from the root node to **1** and **0** in a ZDD Z. We start at the root node of Z and iteratively choose the next node in the path as follows: if a is the current node on the path, then the next node on the path is $\text{hi}(a)$ if $v(a) \in A$ and $\text{lo}(a)$ otherwise. We call the last node along the path P_A the output of A on Z, which is denoted by $Z(A)$; clearly $Z(A)$ is equal to **1** or **0**. We say that Z accepts A if $Z(A) = \mathbf{1}$, otherwise we say that Z rejects A. We say that Z characterizes a family $\mathcal{F} \subset 2^N$ if Z accepts all the sets in the family $\mathcal{F}$ and rejects all the sets not in $\mathcal{F}$. This ZDD is denoted by $Z_{\mathcal{F}}$.

Constructing a ZDD $Z_{\mathcal{F}}$ associated to a family $\mathcal{F}$ of subsets of ground set N can be done in different ways. One way is to construct a BDD for the indicator function of the family $\mathcal{F}$, and delete all the nodes with a high edge pointing to $\mathbf{0}$. There also exist recursive algorithms; see for example Knuth (2009) for a recursive procedure that constructs a ZDD associated to a set of paths between two nodes of an undirected graph fulfilling specific properties. Iwashita and Minato (2013) propose an efficient framework for constructing a ZDD with recursive specification for the family of sets, and in this paper we use their framework to construct the solution space of the pricing problem; the generic method of building a ZDD with a recursive specification proceeds as follows.

Let $N = \{1, \ldots, n\}$. A *configuration* is a pair (j, t) that serves as a node label, where j is an element of N that is associated with the node and t describes the state of the node. Configurations $(n+1, 0)$ and $(n+1, 1)$ represent the $\mathbf{0}$- and $\mathbf{1}$-node of $Z_{\mathcal{F}}$, respectively. A *recursive specification* S of $Z_{\mathcal{F}}$ is defined as pair of functions $ROOT_S()$ and $CHILD_S((j,t), b)$ that return configurations: $ROOT_S()$ is a function without arguments that returns the configuration of the root node of $Z_{\mathcal{F}}$, and $CHILD_S((j,t), b)$ 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.

A generic approach to constructing the ZDD then follows Algorithm 1. This algorithm finds all the reachable configurations of recursive specification S from the root node to the terminal nodes. Using hash tables, one can establish a one-to-one correspondence between the nodes of the DD before the reduction and the configurations. If we assume the hash table operations and the recursive operations to run in constant time, then the runtime of Algorithm 1 is linear in the number of reachable configurations.

Bergman et al. (2016) describe a general B&B algorithm for discrete optimization where computations in BDDs replace the traditional LP relaxation. They construct relaxed BDDs that provide bounds and guidance for branching, and restricted BDDs that lead to a primal heuristic. The construction of the BDDs of the different problems studied in Bergman et al. (2016) is based on a DP model for the given problem. The construction of an exact BDD for a given problem as defined in Bergman et al. (2016) is equivalent to the construction of BDDs as described in Iwashita and Minato (2013). The authors of these two papers have different goals, however. Bergman et al. (2016) aim to solve a discrete optimization problem, while Iwashita and Minato (2013) wish to develop a convenient framework for manipulating

Algorithm 1: Generic top-down/breadth-first ZDD construction

```
Data: Recursive specification S for F
Result: ZDD Z_F
(j_0, s_0) = ROOT_S();
Create a new node with configuration (j_0, s_0);
for j = j_0 to n
    for all nodes r with configuration (j, s) for some s
        for b ∈ {0, 1}
            (j', s') = CHILD_S((j, s), b);
            if (j', s') corresponds to one of the terminal nodes then
                Set the terminal node to be the b-child of r;
            else
                Find or create a node r' with configuration (j', s');
                Set r' to be the b-child of r;
Apply the reduction algorithms for ZDDs to the constructed DD;
```

a family of sets. Moreover, the BDDs in Bergman et al. (2016) are built in such a way that the optimization problem is solved as a shortest or longest path problem. This restriction has an influence on the size of the DDs. In this paper we apply the reduction rules for ZDDs on the constructed DD and solve the pricing problem with a DP algorithm. The reason for this is that we manipulate the family of sets that form a solution to the pricing problem upon branching, and ZDDs are more suitable for such manipulation, see Minato (1993).

5.2. Solving the pricing problem with ZDDs

The pricing problem will be solved using a ZDD that represents the set of all feasible schedules $\mathcal{S}$, for which we will define a recursive specification (see the building phase below). Subsequently, we will explain how to find a schedule with most negative reduced cost given values for the dual variables (the solving phase).

Building phase Remember that the jobs of J are indexed following Property 1. The time windows of the jobs (ready time r_j and deadline $\overline{d}_j$ for every $j \in J$) were also calculated

Table 1 Example instance with four jobs on two machines

job j	p_j	w_j	r_j	$\overline{d}_j$
1	5	89	0	8
2	2	31	0	8
3	6	74	0	11
4	2	12	3	11

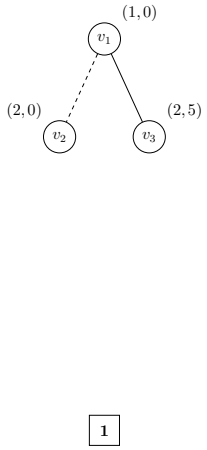
under this ordering. The ZDD will follow the same ordering. Define $sum_j = \sum_{i=j}^{n} p_j$ and $m_j = \min\{p_i \,|\, j \leq i \leq n\}$. A configuration (j,t) of a nonterminal node is a pair consisting of a job index j and the total processing time t of all the processed jobs i with $i < j$, in other words t is the starting time of job j. The **1**- and **0**-node are represented respectively by $(n+1,1)$ and $(n+1,0)$. Function $ROOT_S()$ returns configuration $(1,0)$, and procedure $CHILD_S((j,t),b)$ is described in Algorithm 2. Each node (j,t) in the DD apart from the terminal nodes has two child nodes. The high edge representing inclusion of job j leads to $(j',t+p_j)$, where j' is the job with the smallest index greater than j for which $r_{j'} \leq t+p_j$ and $t+p_j+p_{j'} \leq \overline{d}_{j'}$. The low edge (exclusion of job j) leads to configuration (j',t), where j' is a job with similar properties. If no such job j' exists, the high edge points to the **1**-node if $t+p_j \in [H_{\min}; H_{\max}]$ and to the **0**-node otherwise. For the low edge, the same holds but based on the value of t instead of $t+p_j$.

In Section 5.1 we mentioned that nodes in the DD associated with the same job can be merged if they have the same low and high child. We also implement this reduction process, but we note that different configurations (j,t) and (j,t') might be merged in this way, pointing to different possible starting times t and t' for the job j. These starting times are important for the determination of the cost of the schedules (and also constitute sufficient information). Therefore, with every node p of $Z_{\mathcal{S}}$ we associate the set T_p as the set of all possible starting times of job $v(p)$. These sets can be easily calculated with a top-down/breadth-first method starting at the root node of $Z_{\mathcal{S}}$.

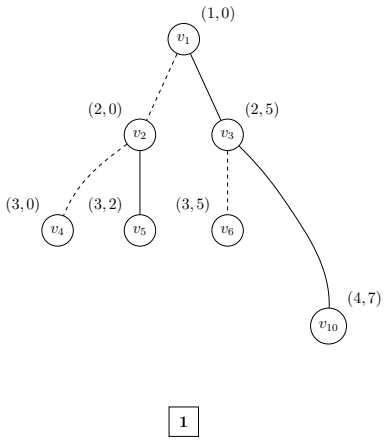
We illustrate the foregoing by means of a small example with $n=4$ and $m=2$, for which $H_{\min}=7$ and $H_{\max}=11$; further data are shown in Table 1. Figure 1 visualizes the different steps of Algorithm 1 for the recursive specification in Algorithm 2. The number of nodes in the ZDD in Figure 1(e) is clearly lower than the DD in Figure 1(d), before the reduction.

Solving phase In each iteration of the CG procedure we will be searching for a variable with negative reduced cost. Hence we must associate to each **1**-path of $Z_{\mathcal{S}}$, i.e. to each

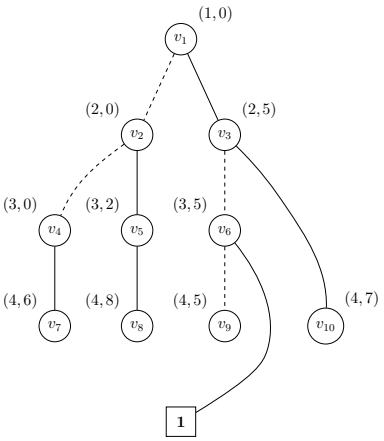
Figure 1 Visualisation of the steps in Algorithm 1 with the specification given by Algorithm 2



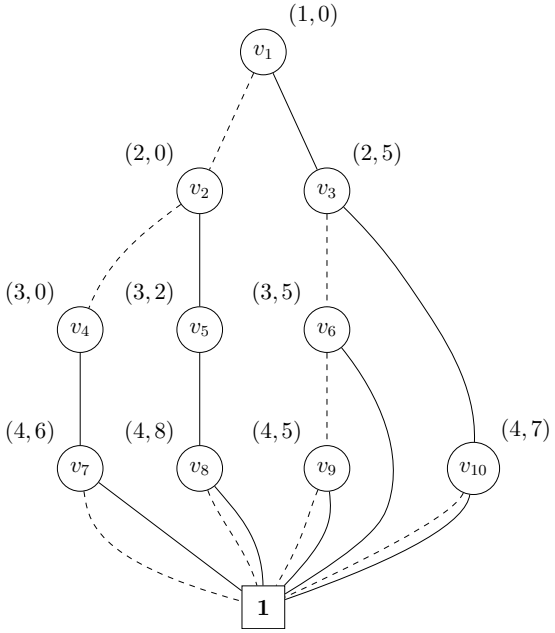
(a) Initialize the root node v_1 and calculate the child nodes v_2 and v_3 of the root node.



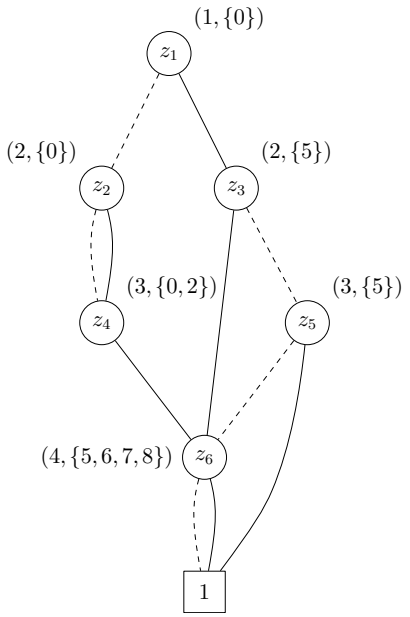
(b) Generate the child nodes of v_2 and v_3. The next high child of v_3 has label 4 because job 3 can only start between times 0 and 5.



(c) Generate the children of v_4, v_5 and v_6. The low edge of the nodes v_4 and v_5 points to the terminal node **0** because $r_4 = 3$. The high child of v_6 is the terminal node **1** because $H_{\max} = 11$.



(d) The only child of v_7, v_8, v_9 and v_{10} is the terminal node **1**.



(e) The result of the reduction algorithms for ZDDs, where the second component in each node label is set T_p (for node p).

Algorithm 2: Calculation of all reachable configurations of $\mathcal{S}$

Function $CHILD_S((j,t),b)$

- **if** $b=1$ **then**
  - $t' \leftarrow t+p_j$;
- **else**
  - $t' \leftarrow t$;
- $j' \leftarrow MINJOB(j,t')$;
- **if** $j' \leq n$ **then**
  - **if** $t'+sum_{j'} < H_{\min}$ **then**
    - **return** $(n+1,0)$;
  - **if** $t' \in [H_{\min}, H_{\max}]$ *and* $t'+m_{j'} > H_{\max}$ **then**
    - **return** $(n+1,1)$;
- **else**
  - **if** $t' \in [H_{\min}, H_{\max}]$ **then**
    - **return** $(n+1,1)$;
  - **return** $(n+1,0)$
- **return** (j',t');

Function $MINJOB(j,t)$

- **if** $\min\{i>j | t \in [r_i, d_i-p_i]\}$ *exists* **then**
  - **return** $\min\{i>j | t \in [r_i, d_i-p_i]\}$;
- **return** $n+1$;

path from the root node to the **1**-node, a value that is equal to the reduced cost of the associated schedule. We implicitly find a path with lowest reduced cost by means of a DP algorithm, which is different from the one presented in Morrison et al. (2016b) because the cost associated to each column (schedule) is nonlinear in function of the job selection. The pricing problem in Morrison et al. (2016b) was a binary combinatorial optimization problem with a linear function, because the cost associated to every variable of the RMP was equal to 1, and then the pricing problem was equivalent to finding a shortest path. In

our case, the nonlinearity is removed by explicitly considering each possible starting time for each job in the definition of a configuration.

Let (π, σ) be an optimal dual vector of the RMP at some iteration of the CG, and let $Z_{\mathcal{S}}$ be a ZDD that represents the family of sets $\mathcal{S}$. Let $|Z_{\mathcal{S}}|$ be the size of $Z_{\mathcal{S}}$ (the number of nodes) and $z_1, \ldots, z_{|Z_{\mathcal{S}}|}$ its nodes, where z_1 is the root node, and $z_{|Z_{\mathcal{S}}|-1}$ and $z_{|Z_{\mathcal{S}}|}$ correspond to the **1**- and **0**-node, respectively. We also assume that each parent node z_i has a lower index i than its child nodes. In the DP algorithm we fill (only the relevant entries of) two tables O and B of size $|Z_{\mathcal{S}}| \times H_{\max}$. Value $O[k][t]$ represents the minimal reduced cost of a *partial schedule* that starts with job $v(k)$ (the job associated to node k) at time t, minimized over all partial schedules that can execute jobs in $\{v(k), v(k)+1, \ldots, n\}$, where a partial schedule contiguously executes a set of jobs from a time instant greater than or equal to zero onwards. The reduced cost of a partial schedule is its contribution to the reduced cost of a schedule $s \in \mathcal{S}$ that contains the partial schedule. This value is associated with a path from configuration $(v(k), t)$ to the **1**-node before the reduction of the DD. Thus, we seek to determine $O[z_1][0]$.

Stepwise, we fill the tables by traversing $Z_{\mathcal{S}}$ from the terminal nodes to the root node; Algorithm 3 illustrates this process. For each node k and starting time t, we store the best choice (selection or not of job $v(k)$, corresponding to values 1 and 0) in $B[k][t]$. We do not examine all values for t ranging from $r_{v(k)}$ to $\overline{d}_{v(k)} - p_{v(k)}$, but we only consider values in set T_k. We compare $O[\text{hi}(z_i)][t + p_{v(z_i)}] - \pi_{v(z_i)} + w_{v(z_i)}(t + p_{v(z_i)})$ and $O[\text{lo}(z_i)][t]$ for all $t \in T_{z_i}$, and we set $O[z_i][t]$ to be the minimum of the two.

6. Farkas pricing

The Farkas lemma is an important result of LP that implies, with the column set $\mathcal{S}' \subset \mathcal{S}$ in the RMP, that exactly one of the following two statements is true:

1. $\exists \lambda \in \mathbb{R}^{|\mathcal{S}'|}$ such that Equations (6b) and (6c) hold, or
2. $\exists \pi \in \mathbb{R}^n_+$ and $\sigma \in \mathbb{R}_+$ such that $\forall s \in \mathcal{S}'$: $\sum_{j \in s} \pi_j - \sigma \leq 0$ and $\sum_{j \in J} \pi_j - m\sigma > 0$.

Therefore, if our RMP is infeasible then there exists a vector (π, σ), which can be interpreted as a ray in the dual, proving the infeasibility of the RMP. With CG we can now try to render the formulation feasible by adding a variable λ_s such that $\sum_{j \in s} \pi_j - \sigma > 0$, i.e., we destroy the proof of infeasibility by the inclusion of this λ_s. Such a variable can be found by solving an optimization problem $\max_{s \in \mathcal{S}} \sum_{j \in s} \pi_j - \sigma$, which is similar to the

Algorithm 3: DP algorithm based on $Z_{\mathcal{S}}$ for the pricing problem

Data: ZDD $Z_{\mathcal{S}}$, optimal solution (π, σ) of the dual

Result: Schedule s of $\mathcal{S}$ with most negative reduced cost

$O[z_{|Z_{\mathcal{S}}|-1}][t] \leftarrow \sigma$ **for all** $t \in T_{z_{|Z_{\mathcal{S}}|-1}}$;

$O[z_{|Z_{\mathcal{S}}|}][t] \leftarrow +\infty$ **for all** $t \in T_{z_{|Z_{\mathcal{S}}|}}$;

for $i = |Z_{\mathcal{S}}| - 2$ *to* 1 **do**

  for $t \in T_{z_i}$ **do**

    if $O[\mathrm{hi}(z_i)][t + p_{v(z_i)}] - \pi_{v(z_i)} + w_{v(z_i)}(t + p_{v(z_i)}) \geq O[\mathrm{lo}(z_i)][t]$ **then**

      $O[z_i][t] \leftarrow O[\mathrm{lo}(z_i)][t]$;

      $B[z_i][t] \leftarrow 0$;

    else

      $O[z_i][t] \leftarrow O[\mathrm{hi}(z_i)][t + p_{v(z_i)}] - \pi_{v(z_i)} + w_{v(z_i)}(t + p_{v(z_i)})$;

      $B[z_i][t] \leftarrow 1$;

$z \leftarrow z_1, t \leftarrow 0$ and $s \leftarrow \emptyset$;

while $z \neq z_{|Z_{\mathcal{S}}|-1}$ **do**

  if $B[z][t] = 0$ **then**

    $s \leftarrow s \cup \{v(z)\}, t \leftarrow t + p_{v(z)}, z \leftarrow \mathrm{hi}(z)$;

  else

    $z \leftarrow \mathrm{lo}(z)$;

pricing problem (9) but without the cost function c_s. If the optimal objective value of this pricing problem is positive, then we iteratively add the new variable to the restricted LP and solve it again, until we have shown that the LP relaxation is either feasible or infeasible. Otherwise, if the optimal objective value of this new pricing problem is negative then we conclude that we cannot remove the infeasibility. This so-called Farkas pricing problem can again be solved using ZDDs. Since the schedule cost is not part of the objective, the problem equates to finding a longest path from the root node to the **1**-node of $Z_{\mathcal{S}}$.

Our foregoing approach for resolving infeasibilities is different from the one in van den Akker et al. (1999). The latter authors add infeasible variables with a "big-M"-type penalty cost to the restricted LP relaxation, and thus at least one feasible solution always exists.

Selecting the value of these big-M penalties is difficult, however. Lower M leads to tighter upper bounds on the respective dual variables and may reduce the so-called "heading-in" effect of initially produced irrelevant columns (due to the lack of compatibility of the initial column pool with the newly added columns). Additionally, van den Akker et al. (1999) also use a number of heuristics in order to test whether or not there exists a feasible solution.

7. Stabilization techniques

It is well known that the convergence of a CG algorithm for scheduling problems can be slow due to primal degeneracy; this situation can be improved by using stabilization techniques. One of these techniques is *dual-price smoothing*, which was introduced in Wentges (1997). This technique corrects the optimal solution of the dual problem of the restricted LP relaxation based on past information before it is plugged into the pricing algorithm. We first introduce a number of concepts before the stabilization technique itself can be explained.

We first describe a dual bound for the LP relaxation of (6), in which we relax the covering constraints (6b). This leads to the following Lagrangian subproblem for any Lagrangian penalty vector $\pi \in \mathbb{R}^n_+$:

$$\text{minimize} \quad \sum_{s\in\mathcal{S}}(c_s - \sum_{j\in s}\pi_j)\lambda_s + \sum_{j\in J}\pi_j \tag{10a}$$

$$\text{subject to} \quad \sum_{s\in\mathcal{S}}\lambda_s \leq m \tag{10b}$$

$$\lambda_s \geq 0 \qquad \text{for each } s\in\mathcal{S} \tag{10c}$$

Using equations (10b) and (10c) we obtain that the Lagrangian dual function $L:\mathbb{R}^n \to \mathbb{R}$ of the Lagrangian relaxation is given by:

$$L(\pi) = \min\left\{0, \min_{s\in\mathcal{S}}\{c_s - \sum_{j\in s}\pi_j\}m\right\} + \sum_{j\in J}\pi_j \tag{11}$$

for every Lagrangian penalty vector $\pi \in \mathbb{R}^n_+$. From (11) we deduce that each time the pricing problem (9) is solved for a dual vector π, we immediately also obtain the dual bound $L(\pi)$ associated to that dual vector. Maximizing the Lagrangian dual function over $\pi \in \mathbb{R}^n_+$ gives the best possible dual bound that can be derived from the Lagrangian relaxation; this is called the Lagrangian dual bound. We define the Lagrangian dual problem as:

$$\max_{\pi\in\mathbb{R}^n_+} L(\pi), \tag{12}$$

which is a max-min problem. Formulating this max-min problem as a linear program will give us a relation between the Lagrangian dual bound and the lower bound obtained from the LP relaxation of (6):

$$\begin{aligned}\max_{\pi\in\mathbb{R}^n_+} L(\pi) &= \max_{\pi\in\mathbb{R}^n_+}\left\{\sum_{j\in J}\pi_j + \min\left\{0, m\min_{s\in\mathcal{S}}\{c_s - \sum_{j\in s}\pi_j\}\right\}\right\}\\ &= \max\left\{\sum_{j\in J}\pi_j - m\sigma \;\middle|\; \pi\in\mathbb{R}^n_+, -\sigma\le 0 \text{ and } -\sigma\le c_s - \sum_{j\in s}\pi_j, \forall s\in\mathcal{S}\right\}\\ &= \min\left\{\sum_{s\in\mathcal{S}} c_s\lambda_s \;\middle|\; \sum_{s\in\mathcal{S}:j\in s}\lambda_s \ge 1, \forall j\in J, \sum_{s\in\mathcal{S}}\lambda_s \le m \text{ and } \lambda_s\ge 0, \forall s\in\mathcal{S}\right\}.\end{aligned}$$

In this way we see that the Lagrangian dual bound is equal to the bound obtained by the LP relaxation of (6).

Wentges (1997) proposes not to simply use the last-obtained vector $\bar{\pi}\in\mathbb{R}^n_+$ of dual prices associated to the covering constraints (6b) of the restricted master in every iteration of the CG, but rather to smoothen these dual prices into the direction of the *stability center*, which is the dual solution $\hat{\pi}$ that has generated the best dual bound $\hat{L} = L(\hat{\pi})$ so far (over the different iterations of the CG). Wentges (1997) uses the following dual-price smoothing rule:

$$\widetilde{\pi} := \alpha\hat{\pi} + (1-\alpha)\bar{\pi} = \hat{\pi} + (1-\alpha)(\bar{\pi} - \hat{\pi}). \tag{13}$$

This vector is plugged into the pricing problem, with $\alpha\in[0,1)$. Define by $\widetilde{s}$ an optimal solution of problem (9) with respect to $\widetilde{\pi}$ and let $\bar{\sigma}$ be the last-obtained dual price associated to constraint (6c) in the restricted master. If $\widetilde{s}$ has negative reduced cost with respect to $\bar{\pi}$ (i.e., $c_{\widetilde{s}} - \sum_{j\in\widetilde{s}}\bar{\pi}_j + \bar{\sigma} < 0$), then the corresponding column $\lambda_{\widetilde{s}}$ can be added to the RMP. Moreover, the stability center $\hat{\pi}$ is updated each time the Lagrangian bound $L(\widetilde{\pi}) = \min\{0, m(c_{\widetilde{s}} - \sum_{j\in\widetilde{s}}\widetilde{\pi}_j)\} + \sum_{j\in J}\widetilde{\pi}_j$ is improved: if $L(\widetilde{\pi}) > \hat{L}$ then we update $\hat{\pi} := \widetilde{\pi}$. This is repeated until $\sum_{j\in J}\bar{\pi}_j - m\bar{\sigma} - \hat{L} < \epsilon$ with $\epsilon > 0$ and sufficiently small. We call this stabilized column generation.

Remark that the pricing problem with smoothed dual vector $\widetilde{\pi}$ might not yield a new variable with negative reduced cost, i.e., $c_{\widetilde{s}} - \sum_{j\in\widetilde{s}}\bar{\pi} + \bar{\sigma} \ge 0$ even if such a variable can be found with the optimal dual vector $(\bar{\pi}, \bar{\sigma})$. We call such a situation *misspricing*. With the following lemma one can show that the number of missprices is polynomially bounded:

LEMMA 1 **(Pessoa et al., 2010)**. *If misspricing occurs for* $\widetilde{\pi}$*, then* $\sum_{j\in J}\bar{\pi}-m\bar{\sigma}-L(\widetilde{\pi}) \leq \alpha(\sum_{j\in J}\bar{\pi}-m\bar{\sigma}-\hat{L})$.

Hence we have that each missprice guarantees that the gap between the primal bound $\sum_{j\in J}\bar{\pi}_j - m\bar{\sigma}$ and the dual bound $\hat{L}$ is reduced by at least a factor $\frac{1}{\alpha}$. Using Lemma 1 one can show now that stabilized CG is correct, meaning that the number of iterations in stabilized GCG is finite and that the procedure finishes with an optimal solution of the LP relaxation of (6) if ϵ is sufficiently small. This establishes that stabilized CG using Wentges (1997) smoothing is asymptotically convergent. For a complete proof we refer to Pessoa et al. (2015), where a link is made between dual-price smoothing and in-out separation (see also Ben-Ameur and Neto, 2007), and where it is shown that other dual-price smoothing schemes in the literature also lead to asymptotically convergent stabilized CG algorithms.

8. Branching rules

At each node of the B&P tree we solve the LP relaxation of formulation (6) and obtain an optimal solution λ^*. One of the following two cases will occur at every node: either the LP relaxation has an integral solution and the new solution can be stored, or the LP solution is fractional. In the second case, we apply a branching strategy to close the integrality gap and find an integer solution.

8.1. General

In van den Akker et al. (1999) it is shown that some fractional solutions can be transformed into an integral solution without much effort, using the following result:

THEOREM 1. *If for every job* $j \in J$ *it holds that* $C_j(s) := C_j$ *is the same for every* $s \in \mathcal{S}$ *with* $\lambda^*_s > 0$*, then the schedule obtained by processing* j *in the execution interval* $[C_j - p_j, C_j]$ *is feasible and has minimum cost.*

The branching strategy of van den Akker et al. (1999) is also based on Theorem 1, and is applied if the LP solution λ^* does not satisfy the conditions of the theorem. In that case, one can deduce from Theorem 1 that there exists at least one job j for which $\sum_{s\in\mathcal{S}} C_j(s)\lambda^*_s > \min\{C_j(s) \,|\, \lambda^*_s > 0\}$; this job is called the fractional job. Using the previous property, the authors design a binary B&B tree for which in every node the fractional job j is first identified and, if any, then two child nodes are created. The first child has the condition that the deadline $\overline{d}_j = \min\{C_j(s) \,|\, \lambda^*_s > 0\}$, in the second child the release date of this

job becomes $r_j = \min\{C_j(s) \mid \lambda_s^* > 0\} + 1 - p_j$. A convenient consequence of this partition strategy is that the pricing algorithm proposed in van den Akker et al. (1999) can stay the same also in the child nodes. It may occur, however, that the newly constructed instance does not have feasible solutions, i.e., that there exists no solution for which $r_j + p_j \leq C_j \leq \overline{d}_j$ for every job $j \in J$.

For set covering formulations such as (6), however, a more generic branching scheme can also be applied, which is based on the following observation: if the solution λ^* of (6) is fractional, then there exists a job pair (j, j') for which $0 < \sum_{s \in \mathcal{S}: j,j' \in s} \lambda_s^* < 1$ (see Ryan and Foster, 1981). One can now separate the fractional solution λ^* using the disjunction $\sum_{s \in \mathcal{S}: j,j' \in s} \lambda_s^* \leq 0$ or $\sum_{s \in \mathcal{S}: j,j' \in s} \lambda_s^* \geq 1$. We call the first branch the DIFF child, in which the jobs j and j' cannot be scheduled on the same machine. The second branch is referred to as the SAME child, where the jobs j and j' have to be scheduled on the same machine. Obviously, these constraints have implications for the pricing problem in the child nodes.

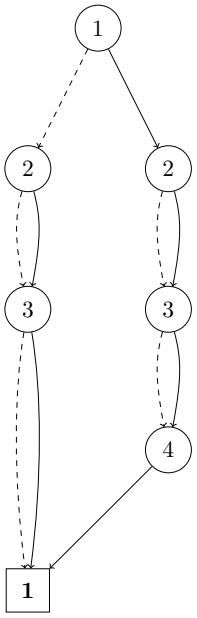
8.2. Constructing ZDDs for the child nodes

We showed in Section 5.2 how the pricing problem in the root node of the search tree can be solved with ZDDs that hold all the feasible schedules. Below we explain how to manipulate a ZDD after imposing branching constraints. First define for each $s \in \mathcal{S}$ an n-dimensional binary vector a_s with $a_{sj} = 1$ if $j \in s$, otherwise $a_{sj} = 0$.

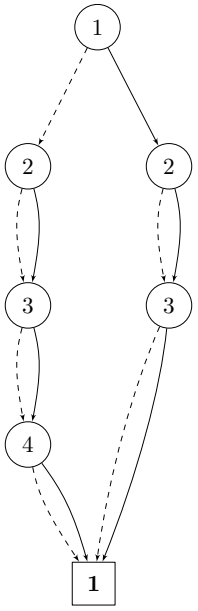
A ZDD for a child node in the B&B tree can be conceived as an intersection of two ZDDs, and so we can construct this ZDD by using the generic binary intersection operation $\cap$ on ZDDs (see Minato, 1993). Suppose for example that we have $Z_{\mathcal{S}}$ in the root node of the search tree, and we wish to constrain this family of sets by considering only schedules s with $a_{sj} = a_{sj'}$. We first define the family $\mathcal{F}_{j,j'}^{SAME} = \{s \subset J \mid a_{sj} = a_{sj'}\}$, represented by the ZDD $Z_{\mathcal{F}_{j,j'}^{SAME}}$. The ZDD $Z_{\mathcal{S}} \cap Z_{\mathcal{F}_{j,j'}^{SAME}}$ is then the ZDD of the SAME child of the root node. A similar construction can be set up for the ZDD of the DIFF child, where the family $\mathcal{F}_{j,j'}^{DIFF} = \{s \subset J \mid a_{sj} + a_{sj'} \leq 1\}$ can be used. In Figure 2 we depict the ZDDs $Z_{\mathcal{F}_{1,4}^{SAME}}$ and $Z_{\mathcal{F}_{1,4}^{DIFF}}$ when $|J| = 4$. Figure 3 visualizes the ZDDs $Z_{\mathcal{S}} \cap Z_{\mathcal{F}_{1,4}^{SAME}}$ and $Z_{\mathcal{S}} \cap Z_{\mathcal{F}_{1,4}^{DIFF}}$ for the instance of Table 1 (with $Z_{\mathcal{S}}$ in Figure 1(e)).

In our implementation we deviate slightly from the foregoing description, in that we first compute the intersection of the recursive specifications and then construct the ZDD with Algorithm 1 given this specification; with this approach we follow Iwashita and Minato (2013). We first note that in a given node of the B&B tree we only know the ZDD Z and

Figure 2 **The ZDDs $Z_{\mathcal{F}^{SAME}_{1,4}}$ and $Z_{\mathcal{F}^{DIFF}_{1,4}}$**

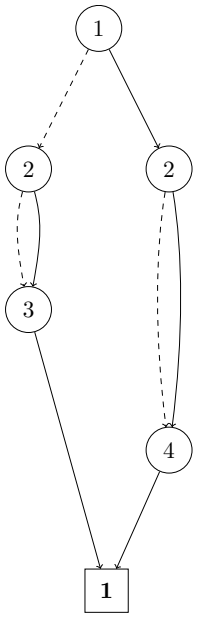


(a) $Z_{\mathcal{F}^{SAME}_{1,4}}$

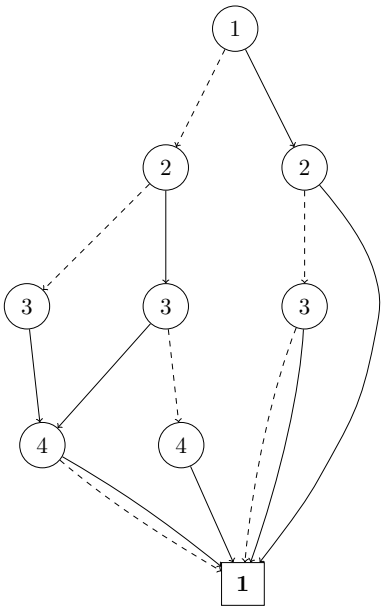


(b) $Z_{\mathcal{F}^{DIFF}_{1,4}}$

Figure 3 **ZDDs in the child nodes after branching with jobs** 1 **and** 4



(a) SAME child



(b) DIFF child

not its recursive specification, but Iwashita and Minato (2013) describe how to re-construct a recursive specification S for Z. The configuration in this case is a pair (i, p), where p is a node of ZDD Z and i is the label of p. Clearly, the $ROOT_S$ function of S returns configuration (i, r), where r is the root node of Z and i the label of the root node. The

function $CHILD_S$ of recursive specification S with arguments (i,p) and $b \in \{0,1\}$ returns (i',p'), where p' is the b-child of p and i' is the label of p'. These two functions are such that when one applies Algorithm 1 with recursive specification S, then this algorithm returns a ZDD that is equivalent to the original ZDD Z.

We now need to construct ZDDs for the child nodes of the B&B tree. For this we present a rather general recursive specification. Let (J, E_{SAME}) be an undirected graph that joins two nodes j and j' if they have to be scheduled on the same machine, and (J, E_{DIFF}) is an undirected graph that joins two nodes j and j' if they have to be scheduled on different machines. Let

$$\mathcal{F}_{SAME} = \{s \subset J \mid a_{sj} = a_{sj'}, \forall \{j,j'\} \in E_{SAME}\}$$

and

$$\mathcal{F}_{DIFF} = \{s \subset J \mid a_{sj} + a_{sj'} \leq 1, \forall \{j,j'\} \in E_{DIFF}\}.$$

We derive a recursive specification $S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ for $\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}$. For every $j \in J$, define A_j as the set $\{j' \in J \mid \{j,j'\} \in E_{SAME}\}$ and B_j as the set $\{j' \in J \mid \{j,j'\} \in E_{DIFF}\}$. The configurations for recursive specification $S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ are pairs $(j, (S_{ADD}, S_{REMOVE}))$ where $j \in J$, $S_{ADD} \subset J$ is a job set that contains all the jobs have that to be added based on the choices made higher in the ZDD, and $S_{REMOVE} \subset J$ contains all the jobs that cannot be added based on the previous choices. The configurations of the **1**-node and the **0**-node are defined as before. The function $ROOT_{S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}}$ returns a pair for which the first component is 1 and the second component is $(\emptyset, \emptyset)$. The function $CHILD_{S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}}$ is summarized as Algorithm 4.

The ZDD of the child nodes in the B&B algorithm can now be set up as follows. Suppose that we apply the generic branching scheme on the jobs j and j'. Let Z be the ZDD at the parent node and S its recursive configuration. The ZDD of the SAME child is the output of Algorithm 1 with recursive specification $S \cap S_{SAME}$, where $S_{SAME} = S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ with $\mathcal{F}_{SAME} = \{s \subset J \mid a_{sj} = a_{sj'}\}$ and $\mathcal{F}_{DIFF} = 2^J$. The ZDD of the DIFF child can be obtained similarly with recursive specification $S \cap S_{SAME}$, where $S_{SAME} = S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ with $\mathcal{F}_{DIFF} = \{s \subset J \mid a_{sj} + a_{sj'} \leq 1\}$ and $\mathcal{F}_{SAME} = 2^J$.

8.3. Branching choice

It is very important to make a good choice for the branching jobs j and j'. This choice has a great influence on the computation time of the algorithm, partly because it decides how

Algorithm 4: Calculation of all reachable configurations of $\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}$

Function $CHILD_S((j,(S_{ADD},S_{REMOVE})),b)$

- **if** $A_j \cap B_j \neq \emptyset$ **then**
  - **return** $(n+1,0)$
- **if** $b=1$ **then**
  - **if** $j \in S_{REMOVE}$ **then**
    - **return** $(n+1,0)$;
  - $S'_{ADD} \leftarrow S_{ADD} \cup A_j$;
  - $S'_{REMOVE} \leftarrow S_{REMOVE} \cup B_j$;
- **else**
  - **if** $j \in S_{ADD}$ **then**
    - **return** $(n+1,0)$;
  - $S'_{REMOVE} \leftarrow S_{REMOVE} \cup A_j$;
- $j' \leftarrow \min(\{i > j \mid i \in S'_{REMOVE}\} \cup \{n+1\})$;
- **if** $j' = n+1$ *and* b **then**
  - **if** $j \notin S_{REMOVE}$ **then**
    - **return** $(n+1,1)$;
  - **return** $(n+1,0)$
- **else**
  - **if** $j \notin S_{ADD}$ **then**
    - **return** $(n+1,1)$;
  - **return** $(n+1,0)$
- **return** $(j',(S'_{ADD},S'_{REMOVE}))$

the B&B algorithm traverses the search tree, but also because it has an impact on the size of the ZDD in the pricing problem.

In our first experiments we used a selection criterion that was introduced by Held et al. (2012) for the vertex coloring problem, but which can be applied to every set covering formulation. For each job pair $\{j,j'\}$ define

$$p(j,j') = \frac{\sum_{s \in \mathcal{S}: j,j' \in s} \lambda_s}{\frac{1}{2}(\sum_{s \in \mathcal{S}: j \in s} \lambda_s + \sum_{s \in \mathcal{S}: j' \in s} \lambda_s)}. \tag{14}$$

For all pairs j, j' we have that $p(j,j') \in [0,1]$ and the value is well defined. When $p(j,j')$ is close to 0 then jobs j and j' tend to be assigned to different machines in the fractional solution, whereas if $p(j,j')$ is close to 1, then j and j' are mostly assigned to the same machines. Therefore, it is preferable to choose a pair j, j' for which the value $p(j,j')$ is (the) close(st) to 0.5, because otherwise the lower bound of child nodes will probably not radically change.

We noticed that the size of the ZDDs of the child nodes typically grows quite fast when this selection criterion is applied. This is not very surprising because the imposed constraints severely change the structure of the pricing problem in the child nodes. This growth in size can be controlled, however, by choosing a branching pair j, j' such that the difference between j and j' is small, and this is an extra aspect that we would like to take along in the branching choice. Assume that $j < j'$. Based on some preliminary experiments, we have come up with the following heuristic priority value:

$$s(j,j') = |p(j,j') - q(j,j') + (j'-j)r(j,j')|\,, \tag{15}$$

and we choose a job pair $\{j, j'\}$ such that $s(j,j')$ is minimal. In this expression,

$$q(j,j') = \sum_{s \in \mathcal{S}: j,j' \in s} \frac{0.5 - |\lambda_s - \lfloor \lambda_s \rfloor - 0.5|}{|\{s \in \mathcal{S} | j, j' \in s\}|} \tag{16}$$

and

$$r(j,j') = \frac{|p(j,j') - q(j,j')| + \epsilon}{n \times r(j,j')}. \tag{17}$$

The underlying reasoning is that we wish to give priority to job pairs for which (1) the difference between j and j' is small, which is achieved by adding the term $(j'-j)r(j,j')$ to $p(j,j')$; and (2) the distance between $p(j,j')$ and $q(j,j')$ is small, which means that the jobs j and j' do not frequently appear on the same machine in the pool of generated columns that have a non-zero λ–value.

9. Computational experiments

9.1. Experimental setup

We have implemented two B&P algorithms, referred to as VHV and RF below, which mainly differ in their branching strategy. VHV uses the branching strategy of van den Akker et al. (1999), but contrary to the original reference our implementation applies Farkas pricing to solve infeasibilities (see Section 6). The pricing problem is solved either

with the DP algorithm presented in van den Akker et al. (1999) or with a ZDD, leading to the variants VHV-DP and VHV-ZDD, respectively. In VHV-ZDD we build a new ZDD that represents the set of feasible schedules with the new time windows at every node of the B&B tree. RF applies the generic branching scheme of Ryan and Foster (1981); the pricing problem is solved with ZDDs as explained in Sections 5.2 and 8.2.

The algorithms have been implemented in the C++ programming language and compiled with gcc version 5.4.0 with full optimization pack -O5. We have used and adjusted the implementation of Iwashita and Minato (2013) that can be found on Github[1] to construct the ZDDs. The computational experiments have been performed on one core of a system with Intel Core i7–3770 processor at 3.4 GHz and 8 GB of RAM under a Linux OS. All LPs are solved with Gurobi 6.5.2 using default settings and only one core.

We test the algorithms on six classes of randomly generated instances, as follows:

class 1: $p_j \sim U[1,10]$ and $w_j \sim U[10,100]$;

class 2: $p_j \sim U[1,100]$ and $w_j \sim U[1,100]$;

class 3: $p_j \sim U[10,20]$ and $w_j \sim U[10,20]$;

class 4: $p_j \sim U[90,100]$ and $w_j \sim U[90,100]$;

class 5: $p_j \sim U[90,100]$ and $w_j \sim U[p_j-5, p_j+5]$;

class 6: $p_j \sim U[10,100]$ and $w_j \sim U[p_j-5, p_j+5]$.

With these settings we follow van den Akker et al. (1999, 2002). We generate instances with $n = 20, 50, 100$ and 150 jobs and $m = 3, 5, 8, 10$ and 12 machines. For each combination of n, m and instance class we construct 20 instances.

9.2. Comparison of pricing algorithms and effect of stabilization

We first compare the different pricing algorithms only in the root node of the B&B tree. Each run is interrupted after 200 seconds. Note that the pricing algorithms in RF and VHV-ZDD are identical in the root node. Table 2 contains a summary of our findings. In this table, *#sol* is the number of instances out of 120 for which an optimal solution for the LP relaxation is found within 200 seconds, *col* is the average number of generated columns over these solved instances and *cpu* is the average CPU time (in seconds) over the solved instances.

From Table 2 we conclude that both for the case with and the case without stabilization, the runtimes for the DP solver and for the ZDD solver are more less comparable in the

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

Table 2 Computation of lower bound in the root node

		No stabilization						With stabilization					
		ZDD			DP			ZDD			DP		
n	m	*#sol*	*cpu*	*col*	*#sol*	*cpu*	*col*	*#sol*	*cpu*	*col*	*#sol*	*cpu*	*col*
20	3	120	0.02	105.3	120	0.01	107.0	120	0.02	154.7	120	0.04	154.8
20	5	120	0.01	73.5	120	0.01	74.0	120	0.01	112.9	120	0.02	109.7
20	8	120	0.00	46.7	120	0.00	47.3	120	0.01	70.1	120	0.01	72.6
20	10	120	0.00	30.0	120	0.00	30.2	120	0.00	42.0	120	0.00	43.4
20	12	120	0.00	23.2	120	0.00	23.0	120	0.00	33.3	120	0.00	33.7
50	3	120	0.67	665.7	120	0.69	661.8	120	0.48	473.3	120	0.56	468.2
50	5	120	0.26	256.0	120	0.26	256.3	120	0.18	233.6	120	0.25	234.5
50	8	120	0.13	103.0	120	0.13	103.1	120	0.08	124.8	120	0.13	128.1
50	10	120	0.08	61.9	120	0.09	62.1	120	0.06	100.5	120	0.09	106.4
50	12	120	0.08	42.4	120	0.06	42.0	120	0.06	86.9	120	0.09	87.0
100	3	101	51.14	24,437.4	105	52.77	27,137.0	120	4.73	1,209.5	120	4.44	1,210.6
100	5	120	4.51	2,130.1	120	4.41	2,146.0	120	2.30	798.9	120	2.32	796.9
100	8	120	1.50	702.8	120	1.51	697.6	120	1.16	511.6	120	1.24	509.3
100	10	120	0.85	409.2	120	0.86	407.5	120	0.64	322.1	120	0.72	319.6
100	12	120	0.68	288.5	120	0.68	288.3	120	0.52	237.8	120	0.59	238.0
150	3	63	73.07	10,640.3	65	67.31	11,755.2	120	22.81	1,927.9	120	20.14	1,925.7
150	5	99	34.84	6,126.4	97	26.87	5,436.8	120	12.59	1,389.2	120	11.39	1,382.7
150	8	120	10.63	1,780.7	120	10.03	1,791.9	120	7.32	1,047.8	120	6.93	1,047.0
150	10	120	6.28	1,212.3	120	6.07	1,213.1	120	4.75	822.8	120	4.58	820.2
150	12	120	4.63	954.3	120	4.52	954.4	120	3.52	663.8	120	3.52	667.4

root node. In the next section we will see that the ZDD solver will nevertheless usually be preferable, by the fact that the RF branching strategy requires less nodes for finding an optimal solution.

The stabilization itself has a beneficial effect especially when the number of jobs per machine is high, so for high ratio n/m. The number of generated columns is also much lower for instances with high n/m. This will also have an important effect for identifying integral solutions at lower levels of the B&B tree.

9.3. Computational results

We will compare the three algorithms VHV-ZDD, VHV-DP and RF. Each run of the algorithm (for one instance) is interrupted after 3600 seconds. We first report the results aggregated over the six instances classes, and subsequently we discuss the detailed performance per class.

Overall computational results Table 3 shows the overall performance of the three algorithms. The columns labeled *#opt* contain the number of solved instances out of 120 within one hour, *node* is the average number of explored nodes of the B&B tree over the solved instances, *cpu* is the average CPU time over the solved instances (in seconds), and *gap* is the maximum absolute gap over the unsolved instances.

Table 3 Comparison of the B&P algorithms aggregated over the six instance classes

		RF				VHV-ZDD				VHV-DP			
n	m	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*
20	3	120	0.8	0	0.03	120	0.8	0	0.02	120	0.8	0	0.03
20	5	120	0.9	0	0.02	120	0.9	0	0.01	120	1.0	0	0.02
20	8	120	0.6	0	0.01	120	0.6	0	0.01	120	0.6	0	0.01
20	10	120	0.3	0	0.00	120	0.3	0	0.00	120	0.3	0	0.00
20	12	120	0.4	0	0.00	120	0.4	0	0.00	120	0.4	0	0.01
50	3	120	4.3	0	1.51	120	5.5	0	1.97	120	5.1	0	1.92
50	5	120	8.8	0	1.09	119	11.6	1	1.55	120	12.2	0	1.76
50	8	120	7.9	0	0.52	119	7.6	1	0.48	119	8.2	1	0.56
50	10	120	9.2	0	0.48	120	9.2	0	0.39	120	9.3	0	0.46
50	12	120	6.1	0	0.30	120	6.7	0	0.27	120	5.9	0	0.26
100	3	119	18.8	1	68.84	120	41.8	0	174.15	120	39.8	0	153.40
100	5	119	32.0	3	39.47	119	69.1	1	118.04	118	68.5	1	112.55
100	8	118	37.9	1	23.06	118	80.1	1	71.39	118	59.6	1	48.82
100	10	119	43.6	1	17.33	120	56.3	0	23.60	120	57.0	0	24.69
100	12	117	48.2	1	16.79	118	70.3	5	25.04	119	75.0	1	26.57
150	3	118	41.1	1	782.67	103	113.8	34	1,911.84	117	116.2	3	1,799.61
150	5	116	58.4	6	443.92	116	196.0	6	1,218.23	114	192.7	6	1,157.16
150	8	114	71.9	13	249.78	114	162.8	6	649.01	114	161.9	9	636.42
150	10	116	87.5	11	199.62	116	161.9	2	372.75	117	171.5	2	427.24
150	12	117	74.3	3	116.73	119	125.3	1	252.07	120	127.1	0	268.51

We first notice that the branching scheme RF needs to explore less nodes on average to find an optimal solution. This has a great impact on the total runtime, and leads to better results especially when the number of jobs per machine is high. Consequently, not only stabilization contributes to the efficiency of the algorithm, but also the branching scheme is of vital importance. We also see that algorithm VHV-DP solves slightly more instances to optimality. The reason why algorithm RF sometimes fails to find optimal solutions is probably a bad exploration of the B&B tree: we use a hybrid depth-first approach and one of the problems associated with depth-first exploration is *trashing*, which occurs when different regions of the search fail for the same or similar reason, see for instance Morrison et al. (2016a). For example a single constraint can lead to infeasibility, but this infeasibility is only detected after exploration of subproblems deeper in the B&B tree.

Computational results for class 1 Table 4 pertains to 20 instances instead of 120. The table shows that the algorithms perform well on the first class of instances. Algorithms VHV-ZDD and VHV-DP solve all the instances to optimality and the average CPU time tends to be smaller than for RF, unless the ratio $\frac{n}{m}$ is at least 18. For instances with the latter property, it becomes more difficult for algorithms VHV-ZDD and VHD-DP to find an optimal solution, the search requires more nodes for these instances and algorithm RF

Table 4 Comparison of the B&P algorithms: Class 1

		RF				VHV-ZDD				VHV-DP			
n	m	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*
20	3	20	1.0	0	0.03	20	0.8	0	0.02	20	0.8	0	0.02
20	5	20	0.7	0	0.02	20	0.6	0	0.02	20	0.7	0	0.01
20	8	20	0.9	0	0.01	20	0.8	0	0.01	20	0.8	0	0.01
20	10	20	0.4	0	0.01	20	0.4	0	0.00	20	0.4	0	0.00
20	12	20	0.5	0	0.00	20	0.5	0	0.01	20	0.5	0	0.01
50	3	20	5.3	0	1.87	20	3.0	0	1.18	20	2.7	0	1.05
50	5	20	7.7	0	0.89	20	3.8	0	0.50	20	4.4	0	0.56
50	8	20	8.7	0	0.44	20	2.9	0	0.18	20	3.0	0	0.17
50	10	20	8.1	0	0.32	20	2.7	0	0.11	20	2.6	0	0.13
50	12	20	4.7	0	0.14	20	2.3	0	0.07	20	2.0	0	0.09
100	3	20	19.8	0	62.66	20	15.4	0	59.79	20	16.6	0	64.54
100	5	19	27.3	3	31.94	20	18.2	0	31.50	20	19.5	0	32.45
100	8	19	34.4	1	15.18	20	12.1	0	7.01	20	12.4	0	7.10
100	10	19	35.9	1	9.79	20	10.8	0	3.66	20	10.8	0	3.78
100	12	20	37.6	0	7.33	20	8.9	0	2.21	20	8.6	0	2.14
150	3	18	31.8	1	570.85	20	67.5	0	1,162.02	20	61.1	0	1,123.36
150	5	20	43.7	0	323.75	20	104.0	0	813.94	20	107.1	0	813.59
150	8	18	59.6	1	128.41	20	67.2	0	275.92	20	65.2	0	261.47
150	10	20	69.4	0	85.50	20	45.7	0	117.39	20	49.1	0	132.09
150	12	20	69.2	0	52.80	20	32.9	0	52.99	20	34.0	0	54.76

finds optimal solutions more quickly. When the ratio $\frac{n}{m}$ becomes bigger we have to set tighter time windows for more jobs.

Computational results for class 2 The first major difference with class 1 is that the lower bound at the root node for class 2 is not tight when the number of machines and jobs increases, because we obtain more fractional solutions. The number of explored nodes also grows, and thus the average runtime also goes up. Clearly, the values of $\frac{w_j}{p_j}$ for every $j \in J$ are often different from each other and the final position of every job on its assigned machine is very clear. Consequently, the assignment of the jobs becomes very important for these instances. We have chosen the weights and the processing times from the uniform distribution $U[1, 100]$. Because of this the number of fractional solutions grows and hence the set covering formulation is less tight. Moreover, the average size of the ZDDs for this class of instances is significantly larger than for class 1 (see Section 9.4). Overall, all three the algorithms solve the problem very well and the average CPU time is small. Algorithm RF performs best especially on instances with 150 jobs.

Computational results for class 3 For instances in this class the processing time p_j and weight w_j for every $j \in J$ are close to each other, and so the ratio between the weight and processing time is close to 1. Consequently, all the jobs have similar priority, and there

Table 5 Comparison of the B&P algorithms: Class 2

		RF				VHV-ZDD				VHV-DP			
n	m	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*
20	3	20	0.9	0	0.03	20	0.9	0	0.02	20	1.0	0	0.02
20	5	20	1.0	0	0.01	20	1.0	0	0.01	20	1.1	0	0.02
20	8	20	0.7	0	0.01	20	0.7	0	0.01	20	0.7	0	0.01
20	10	20	0.5	0	0.00	20	0.5	0	0.01	20	0.5	0	0.01
20	12	20	0.4	0	0.01	20	0.4	0	0.01	20	0.4	0	0.01
50	3	20	1.1	0	0.69	20	1.1	0	0.64	20	1.1	0	0.67
50	5	20	2.8	0	0.60	19	2.8	1	0.59	20	3.0	0	0.64
50	8	20	3.3	0	0.39	19	3.4	1	0.29	19	4.2	1	0.36
50	10	20	3.8	0	0.29	20	2.4	0	0.16	20	2.4	0	0.20
50	12	20	1.6	0	0.10	20	1.4	0	0.09	20	1.4	0	0.11
100	3	20	3.9	0	23.02	20	5.9	0	37.58	20	5.7	0	33.97
100	5	20	23.7	0	56.00	19	44.6	1	112.03	18	24.4	1	61.24
100	8	20	56.4	0	53.16	20	113.5	0	133.55	20	90.3	0	101.02
100	10	20	67.2	0	41.40	20	25.1	0	18.10	20	25.0	0	17.85
100	12	20	100.0	0	50.75	20	163.9	0	74.76	20	169.6	0	75.03
150	3	20	8.3	0	362.30	15	21.1	3	921.16	17	16.9	3	714.94
150	5	16	66.8	6	897.72	16	53.0	6	956.54	15	57.1	6	1,009.13
150	8	17	101.4	13	582.98	16	143.3	6	1,151.20	16	141.4	9	1,130.34
150	10	16	177.4	11	691.58	16	118.6	2	579.59	17	154.0	2	739.64
150	12	18	96.1	3	266.01	20	96.7	0	348.92	20	99.6	0	356.42

will be many relevant feasible schedules. The algorithms VHV-DP and VHV-ZDD perform quite weakly on these instances; this was already observed in van den Akker et al. (1999). It is nevertheless surprising that the algorithms require a long time for finding a feasible solution, because the formulation is very tight. This can be explained by the fact that the set covering formulation often returns a fractional solution. The branching strategy of RF clearly needs less nodes to find an optimal solution and hence the average runtime over the solved instances is lower. We conjecture that in general, the positions of the jobs in the final solution are more predetermined by assigning two jobs to the same machine, rather than by tightening the time window of one job.

Computational results for class 4 These instances have the same properties as the previous class, only the processing times and the weights are larger. Table 7 shows that this class requires higher average runtime than class 3 for algorithms RF and VHV-ZDD. The same number of nodes is explored to find an integer solution, so the higher average computation time can only be explained by the size of the ZDDs, which determines the running time of the pricing problem; see our discussion in Section 9.4.

Computational results for class 5 These instances have the same properties as the previous class, but the different values $\frac{w_j}{p_j}$ for every $j \in J$ are now closer to each other than

Table 6 Comparison of the B&P algorithms: Class 3

		RF				VHV-ZDD				VHV-DP			
n	m	$\#opt$	$node$	gap	cpu	$\#opt$	$node$	gap	cpu	$\#opt$	$node$	gap	cpu
20	3	20	0.8	0	0.03	20	0.7	0	0.02	20	0.6	0	0.02
20	5	20	0.8	0	0.02	20	0.9	0	0.01	20	1.0	0	0.02
20	8	20	0.3	0	0.01	20	0.3	0	0.01	20	0.3	0	0.01
20	10	20	0.2	0	0.00	20	0.2	0	0.00	20	0.2	0	0.00
20	12	20	0.3	0	0.00	20	0.3	0	0.00	20	0.3	0	0.00
50	3	20	9.7	0	2.11	20	14.8	0	4.02	20	14.9	0	4.17
50	5	20	12.0	0	1.05	20	18.5	0	1.91	20	18.5	0	2.01
50	8	20	13.4	0	0.58	20	13.8	0	0.60	20	13.6	0	0.62
50	10	20	10.5	0	0.37	20	13.4	0	0.44	20	14.8	0	0.51
50	12	20	6.0	0	0.18	20	6.3	0	0.18	20	6.1	0	0.21
100	3	20	27.6	0	41.52	20	58.4	0	158.79	20	59.3	0	162.29
100	5	20	37.9	0	22.37	20	83.3	0	109.53	20	83.4	0	103.74
100	8	20	42.8	0	10.66	20	79.1	0	40.79	20	79.0	0	39.34
100	10	20	39.5	0	7.10	20	69.3	0	20.47	20	68.2	0	20.01
100	12	20	42.5	0	6.06	20	68.2	0	12.23	20	67.1	0	12.39
150	3	20	59.5	0	430.16	20	151.4	0	1,508.28	20	152.1	0	1,517.54
150	5	20	69.4	0	166.64	20	232.4	0	965.47	20	230.7	0	1,063.91
150	8	20	72.7	0	71.93	20	210.3	0	504.77	20	206.4	0	517.08
150	10	20	78.5	0	51.05	20	183.9	0	360.87	20	185.7	0	361.57
150	12	20	76.7	0	36.81	20	176.7	0	224.36	20	181.2	0	245.16

Table 7 Comparison of the B&P algorithms: Class 4

		RF				VHV-ZDD				VHV-DP			
n	m	$\#opt$	$node$	gap	cpu	$\#opt$	$node$	gap	cpu	$\#opt$	$node$	gap	cpu
20	3	20	0.6	0	0.03	20	0.6	0	0.02	20	0.7	0	0.03
20	5	20	0.7	0	0.01	20	0.7	0	0.01	20	0.7	0	0.01
20	8	20	0.5	0	0.01	20	0.5	0	0.01	20	0.5	0	0.01
20	10	20	0.2	0	0.00	20	0.2	0	0.00	20	0.2	0	0.00
20	12	20	0.5	0	0.00	20	0.5	0	0.00	20	0.5	0	0.00
50	3	20	0.9	0	0.70	20	1.0	0	0.69	20	1.5	0	1.01
50	5	20	10.7	0	1.17	20	16.6	0	2.13	20	17.3	0	2.59
50	8	20	6.5	0	0.42	20	8.1	0	0.49	20	7.8	0	0.62
50	10	20	10.3	0	0.39	20	12.9	0	0.45	20	12.6	0	0.62
50	12	20	3.1	0	0.13	20	3.6	0	0.13	20	3.5	0	0.17
100	3	20	17.1	0	69.73	20	48.8	0	186.70	20	40.3	0	169.07
100	5	20	39.1	0	33.34	20	100.3	0	147.82	20	99.8	0	159.27
100	8	20	29.0	0	14.84	20	57.2	0	44.29	20	54.9	0	47.10
100	10	20	40.6	0	9.70	20	85.0	0	31.49	20	80.9	0	34.77
100	12	19	35.5	1	7.29	20	60.6	0	18.41	20	57.4	0	20.36
150	3	20	54.0	0	973.53	16	153.8	8	2,261.55	20	154.8	0	2,108.39
150	5	20	65.0	0	383.71	20	266.8	0	1,281.31	20	267.2	0	1,239.42
150	8	20	63.3	0	164.27	20	200.6	0	612.88	20	193.1	0	648.76
150	10	20	75.1	0	93.25	20	220.7	0	358.59	20	218.2	0	429.64
150	12	20	69.3	0	68.99	19	152.6	1	292.79	20	161.4	0	287.08

in the previous two instance classes. From Table 8 we see that the average CPU time of algorithm RF is lower than for algorithms VHV-ZDD and VHV-DP, but the average CPU time is higher than in the previous class. This can be explained by the fact that we need

Table 8 Comparison of the B&P algorithms: Class 5

		RF				VHV-ZDD				VHV-DP			
n	m	$\#opt$	$node$	gap	cpu	$\#opt$	$node$	gap	cpu	$\#opt$	$node$	gap	cpu
20	3	20	0.6	0	0.03	20	0.6	0	0.02	20	0.6	0	0.03
20	5	20	1.0	0	0.02	20	1.0	0	0.02	20	1.0	0	0.02
20	8	20	0.5	0	0.01	20	0.5	0	0.01	20	0.5	0	0.01
20	10	20	0.1	0	0.00	20	0.1	0	0.00	20	0.1	0	0.00
20	12	20	0.4	0	0.00	20	0.4	0	0.00	20	0.4	0	0.01
50	3	20	3.3	0	1.46	20	4.3	0	1.82	20	4.0	0	1.94
50	5	20	11.7	0	1.49	20	16.8	0	2.34	20	18.5	0	3.02
50	8	20	4.4	0	0.34	20	8.2	0	0.58	20	8.4	0	0.72
50	10	20	7.9	0	0.37	20	10.8	0	0.44	20	10.4	0	0.57
50	12	20	4.1	0	0.18	20	4.6	0	0.17	20	3.6	0	0.19
100	3	19	25.4	1	141.01	20	73.5	0	359.54	20	69.3	0	290.36
100	5	20	39.3	0	54.97	20	108.3	0	180.43	20	120.7	0	200.69
100	8	19	30.4	1	18.72	18	156.1	1	152.82	18	59.6	1	54.35
100	10	20	40.1	0	12.94	20	84.4	0	35.22	20	85.8	0	39.51
100	12	18	33.0	1	9.05	18	60.6	5	21.49	19	88.3	1	30.91
150	3	20	60.0	0	1,538.03	14	165.6	13	2,866.70	20	166.1	0	2,508.50
150	5	20	67.9	0	584.08	20	305.4	0	1,488.36	19	290.7	4	1,355.12
150	8	19	77.2	2	335.50	18	200.0	3	770.27	18	199.8	1	736.50
150	10	20	75.9	0	164.37	20	238.1	0	422.13	20	260.3	0	541.05
150	12	19	66.3	1	97.01	20	149.1	0	291.82	20	150.1	0	402.01

more nodes in order to find an optimal solution. Remark also that the ratio of the average CPU time of algorithm VHV-ZDD or VHV-DP over the average CPU time of algorithm RF is also smaller than in the previous class of instances. This can be explained by the fact that the size of ZDDs is much larger than in the previous class.

Computational results for class 6 Here the ratios $\frac{w_j}{p_j}$ are close to each other just as in class 5, but the initial time windows of the different jobs are much tighter (see Section 3). In both cases Algorithm RF has to explore less nodes than algorithms VHV-ZDD and VHV-DP in order to find an optimal solution. The instances become more difficult with higher ratio of n over m. The ratio of cpu of VHV over cpu of RF is clearly lower for instances with smaller $\frac{n}{m}$. Table 9 indicates that algorithm RF performs better overall than algorithms VHV-ZDD and VHV-DP for this class.

Conclusion In general we can conclude from the Tables 4–9 that algorithm RF performs better than algorithms VHV-ZDD and VHV-DP with respect to the number of nodes explored in the B&B tree. This will have consequences for the overall running time. Only for instances of class 1 and 2 the algorithm performs weakly for instances where the ratio $\frac{n}{m}$ is low. One reason for this is that the bound of the child nodes is often not much better than the lower bound of the parent node when we apply the branching strategy of Ryan

Table 9 Comparison of the B&P algorithms: Class 6

		RF				VHV-ZDD				VHV-DP			
n	m	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*	*#opt*	*node*	*gap*	*cpu*
20	3	20	1.1	0	0.03	20	1.1	0	0.03	20	1.1	0	0.04
20	5	20	1.2	0	0.03	20	1.4	0	0.02	20	1.4	0	0.03
20	8	20	1.1	0	0.02	20	1.0	0	0.01	20	1.1	0	0.01
20	10	20	0.6	0	0.01	20	0.6	0	0.00	20	0.6	0	0.01
20	12	20	0.7	0	0.00	20	0.7	0	0.01	20	0.7	0	0.00
50	3	20	5.8	0	2.25	20	8.6	0	3.44	20	6.8	0	2.66
50	5	20	7.8	0	1.37	20	10.7	0	1.80	20	11.4	0	1.75
50	8	20	11.1	0	0.96	20	9.1	0	0.70	20	11.9	0	0.85
50	10	20	14.5	0	1.13	20	13.0	0	0.74	20	13.3	0	0.73
50	12	20	17.1	0	1.05	20	21.9	0	0.96	20	18.7	0	0.80
100	3	20	19.4	0	78.70	20	49.3	0	242.48	20	47.6	0	200.14
100	5	20	24.5	0	37.83	20	58.9	0	126.66	20	58.7	0	112.79
100	8	20	34.3	0	25.21	20	70.2	0	58.02	20	61.7	0	44.59
100	10	20	37.8	0	22.65	20	63.5	0	32.67	20	71.1	0	32.22
100	12	20	38.4	0	18.98	20	58.7	0	20.81	20	60.1	0	18.81
150	3	20	32.4	0	799.97	18	125.0	34	2,965.41	20	131.4	0	2,662.24
150	5	20	39.7	0	398.39	20	186.2	0	1,751.43	20	174.3	0	1,434.67
150	8	20	60.7	0	257.70	20	155.4	0	691.60	20	165.3	0	633.14
150	10	20	66.8	0	210.39	20	155.9	0	439.32	20	159.2	0	406.29
150	12	20	70.2	0	192.70	20	145.2	0	303.59	20	136.4	0	265.66

and Foster (1981). The branching strategy of van den Akker et al. (1999) performs better on instances where the number of jobs per machine is low. This is not surprising because a tighter time window for a job has a larger influence on the space of feasible solutions if there are not many jobs per machine. Moreover, trashing occurs when we apply a depth-first exploration strategy of the B&B tree, and this has an important effect for this type of instances because the lower bound at the root node is not as tight as for the other cases.

9.4. Evolution of the size of the ZDDs

In Table 10 we report the average (*avg*) and maximum (*max*) size of the ZDD at the root node for every n, m and instance class. We conclude that the pricing problem is the hardest to solve for the instances with ratio $\frac{w_j}{p_j}$ close to 1 for every job j. The range of possible processing times and the number of jobs per machine also have a major influence on the size of the ZDDs. For constant n, the size of the ZDDs decreases with increasing m. Overall, the size of the ZDDs is considerable, hence it is important to prevent an additional fast growth upon applying the branching constraints. We also remark that the CPU time for building the ZDD at the root node is very low; on average this is less than 0.06 seconds. This is much lower than the construction times for the maximum independent set problem that are reported in Morrison et al. (2016b).

Table 10 Average and maximum size of the ZDD at the root node

		Class 1		Class 2		Class 3		Class 4		Class 5		Class 6	
n	m	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*	*avg*	*max*
20	3	240.8	310	1,018.8	1,208	466.6	535	345.2	454	464.0	566	1,271.0	1,400
20	5	197.0	262	601.3	800	258.8	308	90.8	152	119.7	202	707.6	870
20	8	146.2	172	306.1	422	129.2	166	77.1	95	67.0	93	375.1	513
20	10	130.6	170	224.6	385	95.7	122	68.1	95	85.4	107	266.7	437
20	12	114.7	146	178.0	244	75.6	103	51.2	72	39.9	51	209.5	322
50	3	1,513.4	1,907	11,808.1	15,303	3,653.7	4,344	7,408.6	9,394	12,463.7	15,137	18,153.5	19,947
50	5	1,144.2	1,336	8,765.8	10,611	2,570.4	2,876	4,030.3	5,712	7,429.3	8,103	13,550.6	15,456
50	8	906.3	1,052	6,532.7	8,074	1,741.4	2,033	1,542.8	2,040	2,106.6	2,556	9,618.4	10,508
50	10	777.7	895	5,484.6	7,130	1,373.6	1,773	1,432.0	1,722	2,317.9	2,559	7,690.8	8,645
50	12	700.9	796	4,673.9	5,453	1,119.1	1,384	845.1	977	1,113.0	1,314	6,292.5	6,840
100	3	5,235.6	6,267	45,232.2	53,007	15,539.2	17,702	59,735.1	73,532	103,731.4	109,710	77,922.9	87,083
100	5	3,943.7	4,595	36,614.3	43,186	10,741.1	12,274	34,212.3	38,944	60,533.0	65,571	57,572.1	63,864
100	8	3,098.6	3,573	25,808.9	28,909	7,462.0	8,345	14,485.3	18,378	23,451.2	26,527	40,907.6	43,930
100	10	2,706.6	3,066	22,379.5	27,536	5,817.8	6,499	11,457.5	12,716	18,747.6	20,367	34,953.8	39,827
100	12	2,440.3	2,950	19,577.8	22,914	5,231.6	6,096	6,573.9	8,540	9,252.4	10,766	30,160.2	32,381
150	3	11,520.6	12,592	101,307.5	113,608	33,584.1	38,954	161,673.2	185,651	267,742.3	280,935	172,325.7	186,026
150	5	8,415.1	9,271	75,276.2	84,689	23,528.7	26,732	98,735.7	113,617	172,946.1	185,661	128,118.4	146,306
150	8	6,073.3	6,807	56,595.0	62,024	15,869.0	17,634	53,812.8	62,751	93,094.4	98,813	93,031.9	104,062
150	10	5,340.5	5,969	48,426.1	54,678	13,041.3	14,093	38,050.7	43,006	63,917.9	66,859	78,653.1	84,328
150	12	4,749.1	5,213	41,274.8	46,559	11,453.7	13,007	25,859.8	30,273	41,082.4	45,856	69,401.9	77,230

Table 11 Evolution of the size of the ZDDs for instances with 50 jobs

	$m=3$		$m=5$		$m=8$		$m=10$		$m=12$	
depth	H	C	H	C	H	C	H	C	H	C
5	1.34	−0.04	1.14	−0.02	0.53	0.02	0.20	0.04	0.15	0.07
10	4.20	−0.08	1.83	−0.05	0.93	0.03	0.22	0.07	0.12	0.15
15	7.67	—	1.99	−0.14	1.62	0.08	0.39	0.22	0.51	0.33
20	—	—	—	—	3.17	−0.04	0.76	—	—	0.45

Table 11 displays the average grow rate of the size of the ZDDs at various depths of the search tree compared to the size at the root node of the tree, for instances with 50 jobs. For each value of m there are two columns, with H containing the average size using the decision heuristic of Held et al. (2012), and C the average size with the rule introduced in Section 8. A tabel entry of 1.34, for instance, means that for this setting the average size of the ZDD is around 134% of the size at the root. We report only the values for depths of the B&B tree that are a multiple of 5. For every number of machines we calculate over all six instance classes. We observe that our branching choice allows to control the size of the ZDDs; in some cases the size of the ZDDs is even reduced compared to the root node.

10. Summary and conclusion

In this paper we have augmented the B&P algorithm of van den Akker et al. (1999) by adding several new features such as dual stabilization and Farkas pricing, which are important for calculating the LP lower bound at every node of the search tree. Additionally,

we have also examined the use of ZDDs for solving the pricing problem. This creates the opportunity to use a generic branching scheme (Ryan and Foster, 1981) for this problem. We have observed that this branching scheme performs very well on instances that could not be solved using the branching scheme of van den Akker et al. (1999). More generally, this generic branching scheme can be used to solve scheduling problems on parallel machines for which the single-machine problem is optimally solved using a priority rule. For future research it would be interesting to evaluate whether it is possible to use ZDDs in the pricing problem for machine scheduling problems that do not have this property.

References

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

Azizoglu, M., O. Kirca. 1999. On the minimization of total weighted flow time with identical and uniform parallel machines. *European Journal of Operational Research* **113** 91–100.

Belouadah, H., C.N. Potts. 1994. Scheduling identical parallel machines to minimize total weighted completion time. *Discrete Applied Mathematics* **48** 201–218.

Ben-Ameur, W., J. Neto. 2007. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks* **49** 3–17.

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.

Bruno, J., E.G. Jr Coffman, R. Sethi. 1974. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM* **17** 382–387.

Chen, Z-L., W.B. Powell. 1999. Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing* **11** 78–94.

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.

Elmaghraby, S.E., S.H. Park. 1974. Scheduling jobs on a number of identical machines. *AIIE transactions* **6** 1–13.

Held, S., W. Cook, E.C. Sewell. 2012. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation* **4** 363–381.

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.

Knuth, D.E. 2009. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. 12th ed. Addison-Wesley Professional.

Lawler, E.L., J.M. Moore. 1969. A functional equation and its application to resource allocation and sequencing problems. *Management Science* **16** 77–84.

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

Lee, C-Y., R. Uzsoy. 1992. A new dynamic programming algorithm for the parallel machines total weighted completion time problem. *Operations Research Letters* **11** 73–75.

Lenstra, J.K., A.H.G. Rinnooy Kan, P. Brucker. 1977. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* **1** 343–362.

Lopes, M.J.P., J.M.V. de Carvalho. 2007. A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research* **176** 1508–1527.

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.

Morrison, D.R., S.H. Jacobson, J.J. Sauppe, E.C. Sewell. 2016a. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization* **19** 79–102.

Morrison, D.R., E.C. Sewell, S.H. Jacobson. 2016b. Solving the pricing problem in a branch-and-price algorithm for graph coloring using zero-suppressed binary decision diagrams. *INFORMS Journal on Computing* **28** 67–82.

Pessoa, A., R. Sadykov, E. Uchoa, F. Vanderbeck. 2015. Automation and combination of linear-programming based stabilization techniques in column generation. Tech. Rep. hal-01077984. URL `https://hal.inria.fr/hal-01077984`.

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

Rothkopf, M.H. 1966. Scheduling independent tasks on parallel processors. *Management Science* **12** 437–447.

Ryan, D.M., B.A. Foster. 1981. An integer programming approach to scheduling. A. Wren, ed., *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam, 269–280.

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

Smith, W.E. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3** 59–66.

van den Akker, J.M., H. Hoogeveen, S. van de Velde. 2002. Combining column generation and lagrangean relaxation to solve a single-machine common due date problem. *INFORMS Journal on Computing* **14** 37–51.

van den Akker, J.M., J.A. Hoogeveen, S.L. van de Velde. 1999. 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.

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

FACULTY OF ECONOMICS AND BUSINESS
Naamsestraat 69 bus 3500
3000 LEUVEN, BELGIË
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
info@econ.kuleuven.be
www.econ.kuleuven.be