

Multivalued Decision Diagrams for Sequencing Problems

Andre A. Cire, Willem-Jan van Hoeve

Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA
{acire,vanhoeve}@andrew.cmu.edu

Sequencing problems are among the most prominent problems studied in operations research, with primary application in, e.g., scheduling and routing. We propose a novel approach to solving generic sequencing problems using multivalued decision diagrams (MDDs). Because an MDD representation may grow exponentially large, we apply MDDs of limited size as a discrete relaxation to the problem. We show that MDDs can be used to represent a wide range of sequencing problems with various side constraints and objective functions, and demonstrate how MDDs can be added to existing constraint-based scheduling systems. Our computational results indicate that the additional inference obtained by our MDDs can speed up a state-of-the-art solver by several orders of magnitude, for a range of different problem classes.

Key words: sequencing, single machine scheduling, networks/graphs, programming, decision diagrams

History: Submitted December, 2012. Revised May, 2013 (minor revision).

1. Introduction

Sequencing problems are among the most widely studied problems in operations research. Specific variations of sequencing problems include single machine scheduling, the traveling salesman problem with time windows, and precedence-constrained machine scheduling. Sequencing problems are those where the best order for performing a set of tasks must be determined, which in many cases leads to an NP-hard problem (Garey and Johnson 1979, Section A5). Sequencing problems are prevalent in manufacturing and routing applications, including production plants where jobs should be processed one at a time in an assembly line, and in mail services where packages must be scheduled for delivery on a vehicle. Industrial problems that involve multiple facilities may also be viewed as sequencing problems in certain scenarios, e.g. when a machine is the bottleneck of a manufacturing plant (Pinedo 2008). Existing methods for sequencing problems either follow a dedicated heuristic for a specific problem class, or utilize a generic solving methodology such as integer programming or constraint programming. Given the practical importance and computational hardness, understanding how sequencing problems can be solved more effectively is an active research area.

In this work we propose a new approach for solving sequencing problems based on *multivalued decision diagrams* (MDDs). Decision diagrams are compact graphical representations of Boolean functions, originally introduced for applications in circuit design by Lee (1959), and widely studied and applied in computer science. They have been recently used to represent the feasible set of discrete optimization problems, as demonstrated in Becker et al. (2005) and Bergman et al. (2011, 2012). This is done by perceiving the constraints of a problem as a Boolean function $f(x)$ representing whether a solution x is feasible. Nonetheless, such MDDs can grow exponentially large, which makes any practical computation prohibitive in general.

To circumvent this issue, Andersen et al. (2007) introduced the concept of a *relaxed MDD*, which is a diagram of limited size that represents instead an over-approximation of the feasible solution set of a problem. We argue in this paper that such MDDs can be particularly useful as a discrete relaxation of the feasible set of sequencing problems. In particular, a relaxed MDD can be embedded within a complete search procedure such as branch and bound for integer programming or backtracking search for constraint programming (Andersen et al. 2007, Hoda et al. 2010).

We focus on a broad class of sequencing problems where jobs should be scheduled on a single machine and are subject to precedence and time window constraints, and in which setup times can be present. It generalizes a number of single machine scheduling problems and variations of the traveling salesman problem (TSP). The relaxation provided by the MDD, however, is suitable to any problem where the solution is defined by a permutation of a fixed number of tasks, and it does not directly depend on particular constraints or on the objective function.

The main contributions of this work are as follows. We propose a novel formulation of the feasible set of a sequencing problem as an MDD, and show how it can be relaxed so that its size is limited according to a given parameter. We present how the MDD can be used to compute bounds on typical objective functions in scheduling, such as the makespan and total tardiness. Moreover, we demonstrate how to derive more structured sequencing information from the relaxed MDD, in particular a valid set of precedence relations that must hold in any feasible solution.

We also propose a number of techniques for strengthening the MDD relaxation, which take into account the precedence and time window constraints. We demonstrate that these generic techniques can be used to derive a polynomial-time algorithm for a particular TSP variant introduced by Balas (1999) by showing that the associated MDD has polynomial size.

To demonstrate the use of relaxed MDDs in practice, we apply our techniques to *constraint-based scheduling* (Baptiste et al. 2001). Constraint-based scheduling plays a central role as a general-purpose methodology in complex and large-scale scheduling problems. Examples of commercial applications that apply this methodology include yard planning of the Singapore port and gate allocation of the Hong Kong airport (Freuder and Wallace 2000), Brazilian oil-pipeline scheduling

(Lopes et al. 2010), and home health care scheduling (Rendl et al. 2012). We show that, by using the relaxed MDD techniques described here, we can improve the performance of the state-of-the-art constraint-based schedulers by orders of magnitude on single machine problems without losing the generality of the method. In particular, we were able to close three open TSPLIB instances for the sequencing ordering problem.

The paper is organized as follows. Section 2 presents a brief overview of related literature. Section 3 defines the general sequencing problem that will be considered throughout the paper, and Section 4 shows how its feasible set is represented with an MDD. Section 5 describes relaxed MDDs and the basic operations for strengthening its representation. Sections 6 and 7 present detailed methods to filter and refine relaxed MDDs. In Section 8, we present an efficient procedure to deduce precedence relations from the relaxation. Section 9 demonstrates how the techniques can be used to obtain a polynomial-size MDD that exactly represents the feasible set of a particular TSP variant. Finally, Section 10 presents the application of MDDs to constraint-based scheduling, and concluding remarks are given in Section 11.

2. Related Work

Relaxed MDDs were first introduced by Andersen et al. (2007) as a discrete relaxation for arbitrary constraint satisfaction problems. They were proposed as an alternative to the domain store relaxation that is commonly used in constraint programming techniques. Following that work, Hadzic et al. (2008) and Hoda et al. (2010) developed generic methods for systematically compiling relaxed MDDs. The fundamental idea of their approaches is to construct the diagram in an incremental fashion, associating a particular *state* information with the MDD nodes so as to indicate how new nodes and arcs should be added to the diagram. Bergman et al. (2011, 2012) propose similar techniques for the purpose of obtaining bounds for combinatorial optimization problems, which were shown to be superior to their corresponding continuous relaxations for a wide range of instances.

The application of relaxed MDDs to disjunctive scheduling was first proposed by Hoda et al. (2010), and studied in the context of constraint-based propagators by Cire and van Hoeve (2012). Our work expands on the ideas presented in these previous papers, showing new theoretical properties and improved techniques that are applicable to arbitrary sequencing problems.

The techniques we develop here are based on associating a state information with the nodes of the MDD, as in Hoda et al. (2010). By doing so, our method is closely related to that of *state-space relaxations* by Christofides et al. (1981) for routing problems. A similar idea was exploited by Hernádvölgyi (2003), which considers a branch-and-bound algorithm based on homomorphic abstractions of the search space for the sequential ordering problem. In our case, the state-space relaxation is implicitly represented by the nodes of the MDD, which allow us to accommodate

multiple constraints more easily in the relaxation. Moreover, we are able to work with different state relaxations simultaneously; namely, one from a top-down perspective of the diagram, and another from a bottom-up perspective, as will become clear in later sections.

Lastly, decision diagrams have also been considered in other areas of optimization, e.g., cut generation in integer programming (Becker et al. 2005) and 0-1 vertex and facet enumeration (Behle and Eisenbrand 2007).

3. Problem Definition

In this work we focus on generic sequencing problems, presented here in terms of ‘unary machine’ scheduling. Note that a machine may refer to any resource capable of handling at most one activity at a time.

Let $\mathcal{J} = \{j_1, \dots, j_n\}$ be a set of n jobs to be processed on a machine that can perform at most one job at a time. Each job $j \in \mathcal{J}$ has an associated *processing time* p_j , which is the number of time units the job requires from the machine, and a *release date* r_j , the time from which job j is available to be processed. For each pair of distinct jobs $j, j' \in \mathcal{J}$ a *setup time* $t_{j,j'}$ is defined, which indicates the minimum time that must elapse between the end of j and the beginning of j' if j' is the first job processed after j finishes. We assume that jobs are *non-preemptive*, i.e. we cannot interrupt a job while it is being processed on the machine.

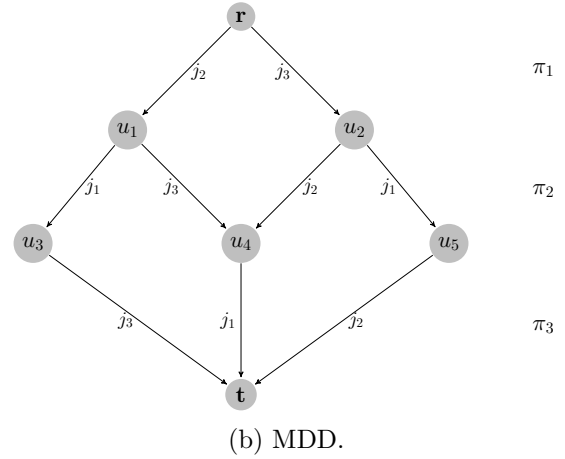
We are interested in assigning a *start time* $s_j \geq r_j$ for each job $j \in \mathcal{J}$ such that job processing intervals do not overlap, the resulting schedule observes a number of constraints, and an objective function f is minimized. Two types of constraints are considered in this work: *precedence constraints*, requiring that $s_j \leq s_{j'}$ for certain pairs of jobs $(j, j') \in \mathcal{J} \times \mathcal{J}$, which we equivalently write $j \ll j'$; and *time window constraints*, where the *completion time* $c_j = s_j + p_j$ of each job $j \in \mathcal{J}$ must be such that $c_j \leq d_j$ for some *deadline* d_j . Furthermore, we study three representative objective functions in scheduling: the *makespan*, where we minimize the completion time of the schedule, or $\max_{j \in \mathcal{J}} c_j$; the *total tardiness*, where we minimize $\sum_{j \in \mathcal{J}} (\max\{0, c_j - \delta_j\})$ for given *due dates* δ_j ; and the *sum of setup times*, where we minimize the value obtained by accumulating the setup times $t_{j,j'}$ for all consecutive jobs j, j' in a schedule. Note that for these objective functions we can assume that jobs should always be processed as early as possible (i.e., idle times do not decrease the value of the objective function).

Since jobs are processed one at a time, any solution to such scheduling problem can be equivalently represented by a total ordering $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ of \mathcal{J} . The start time of the job j implied by π is given by $s_j = r_j$ if $j = \pi_1$, and $s_j = \max\{r_j, s_{\pi_{i-1}} + p_{\pi_{i-1}} + t_{\pi_{i-1}, j}\}$ if $j = \pi_i$ for some $i \in \{2, \dots, n\}$. We say that an ordering π of \mathcal{J} is *feasible* if the implied job times observe the precedence and time window constraints, and *optimal* if it is feasible and minimizes f .

Job Parameters			
Job	Release (r_j)	Deadline (d_j)	Processing (p_j)
j_1	2	20	3
j_2	0	14	4
j_3	1	14	2

Setup Times			
	j_1	j_2	j_3
j_1	-	3	2
j_2	3	-	1
j_3	1	2	-

(a) Instance data.



(b) MDD.

Figure 1 Example of an MDD for a scheduling problem.

4. MDD Representation

For the purpose of this work, an MDD \mathcal{M} is a directed acyclic graph whose paths represent the feasible orderings of \mathcal{J} . The set of nodes of \mathcal{M} are partitioned into $n + 1$ layers L_1, \dots, L_{n+1} , where layer L_i corresponds to the i -th position π_i of the feasible orderings encoded by \mathcal{M} , for $i = 1, \dots, n$. Layers L_1 and L_{n+1} are singletons representing the root \mathbf{r} and the terminal \mathbf{t} , respectively. An arc $a = (u, v)$ of \mathcal{M} is always directed from a *source node* u in some layer L_i to a *target node* v in the subsequent layer L_{i+1} , $i \in \{1, \dots, n\}$. We write $\ell(a)$ to indicate the layer of the source node u of the arc a (i.e., $u \in L_{\ell(a)}$).

With each arc a of \mathcal{M} we associate a label $val(a) \in \mathcal{J}$ that represents the assignment of the job $val(a)$ to the $\ell(a)$ -th position of the orderings identified by the paths traversing a . Hence, an arc-specified path (a_1, \dots, a_n) from \mathbf{r} to \mathbf{t} identifies the ordering $\pi = (\pi_1, \dots, \pi_n)$, where $\pi_i = val(a_i)$ for $i = 1, \dots, n$. Every feasible ordering is identified by some path from \mathbf{r} to \mathbf{t} in \mathcal{M} , and conversely every path from \mathbf{r} to \mathbf{t} identifies a feasible ordering.

EXAMPLE 1. We provide an MDD representation for a sequencing problem with three jobs j_1 , j_2 , and j_3 . The instance data is presented in Figure 1a, and the associated MDD \mathcal{M} is depicted in Figure 1b. No precedence constraints are considered. There are 4 feasible orderings in total, each identified by a path from \mathbf{r} to \mathbf{t} in \mathcal{M} . In particular, the path traversing nodes \mathbf{r} , u_2 , u_4 , and \mathbf{t} represents a solution where jobs j_3 , j_2 , and j_1 are performed in this order. The completion times for this solution are $c_{j_1} = 15$, $c_{j_2} = 9$, and $c_{j_3} = 3$. Note that we can never have a solution where j_1 is first on the machine, otherwise either the deadline of j_2 or j_3 would be violated. Hence, there is no arc a with $val(a) = j_1$ directed out of \mathbf{r} . \square

Some additional notation follows. The incoming arcs at a node u are denoted by $in(u)$, and the outgoing arcs leaving u by $out(u)$. The *width* of a layer L_i is $|L_i|$, and the width of \mathcal{M} is the maximum width among all layers. The MDD in Figure 1b has a width of 3.

Two nodes u, v on the same layer in an MDD are *equivalent* (or belong to the same *equivalence class*) if the set of u - \mathbf{t} paths is equal to the set of v - \mathbf{t} paths. That is, for any u - \mathbf{t} arc-specified path (a_1, a_2, \dots, a_k) there exists a v - \mathbf{t} arc-specified path $(a'_1, a'_2, \dots, a'_k)$ such that $val(a_1) = val(a'_1)$, $val(a_2) = val(a'_2)$, \dots , $val(a_k) = val(a'_k)$, and vice-versa. An MDD \mathcal{M} is *reduced* if no two nodes in any layer are equivalent. This is the case, for example, for the MDD in Figure 1b. A standard result in decision diagram theory is that there exists a unique reduced MDD representing the feasible orderings of \mathcal{J} , provided that we do not change the mapping between the layers L_i of \mathcal{M} and the ordering positions π_i ; see, e.g., Wegener (2000). The reduced MDD also has the smallest width among the MDDs encoding the feasible orderings of \mathcal{J} .

We next show how to compute the orderings that yield the optimal makespan and the optimal sum of setup times in polynomial time in the size of \mathcal{M} . For the case of total tardiness and other similar objective functions, we are able to provide a lower bound on its optimal value also in polynomial time in \mathcal{M} .

- *Makespan.* For each arc a in \mathcal{M} , define the *earliest completion time* of a , or ect_a , as the minimum completion time of the job $val(a)$ among all orderings that are identified by the paths in \mathcal{M} containing a . If the arc a is directed out of \mathbf{r} , then a assigns the first job that is processed in such orderings, thus $ect_a = r_{val(a)} + p_{val(a)}$. For the remaining arcs, recall that the completion time c_{π_i} of a job π_i depends only on the completion time of the previous job π_{i-1} , the setup time t_{π_{i-1}, π_i} , and on the specific job parameters; namely, $c_{\pi_i} = \max\{r_{\pi_i}, c_{\pi_{i-1}} + t_{\pi_{i-1}, \pi_i}\} + p_{\pi_i}$. It follows that the earliest completion time of an arc $a = (u, v)$ can be computed by the relation

$$ect_a = \max\{r_{val(a)}, \min\{ect_{a'} + t_{val(a'), val(a)} : a' \in in(u)\}\} + p_{val(a)}. \quad (1)$$

The minimum makespan is given by $\min_{a \in in(\mathbf{t})} ect_a$, as the arcs directed to \mathbf{t} assign the last job in all orderings represented by \mathcal{M} . An optimal ordering can be obtained by recursively retrieving the minimizer arc $a' \in in(u)$ in the “min” of (1).

- *Sum of Setup Times.* The minimum sum of setup times is computed analogously: For an arc $a = (u, v)$, let st_a represent the minimum sum of setup times up to job $val(a)$ among all orderings that are represented by the paths in \mathcal{M} containing a . If a is directed out of \mathbf{r} , we have $st_a = 0$; otherwise,

$$st_a = \min\{st_{a'} + t_{val(a'), val(a)} : a' \in in(u)\}. \quad (2)$$

The minimum sum of of setup times is given by $\min_{a \in in(\mathbf{t})} st_a$.

- *Total Tardiness.* The *tardiness* of a job j is defined by $\max\{0, c_j - \delta_j\}$ for some due date δ_j . Unlike the previous two cases, the tardiness value that a job attains in an optimal solution depends on the sequence of all activities, not only on its individual contribution or the value of its immediate predecessor. Nonetheless, as the tardiness function for a job is non-decreasing in its completion time, we can utilize the earliest completion time as follows. For any arc $a = (u, v)$, the value $\max\{0, ect_a - \delta_{val(a)}\}$ yields a lower bound on the tardiness of the job $val(a)$ among all orderings that are represented by the paths in \mathcal{M} containing a . Hence, a lower bound on the total tardiness is given by the length of the shortest path from \mathbf{r} to \mathbf{t} , where the length of an arc a is set to $\max\{0, ect_a - \delta_{val(a)}\}$. Observe that this bound is tight if the MDD is composed by a single path.

We remark that valid bounds for many other types of objective in the scheduling literature can be computed in an analogous way as above. For example, suppose the objective is to minimize $\sum_{j \in \mathcal{J}} f_j(c_j)$, where f_j is a function defined for each job j and which is non-decreasing on the completion time c_j . Then, as in total tardiness, the value $f_{val(a)}(ect_a)$ for an arc $a = (u, v)$ yields a lower bound on the minimum value of $f_{val(a)}(c_{val(a)})$ among all orderings that are identified by the paths in \mathcal{M} containing a . Using such bounds as arc lengths, the shortest path from \mathbf{r} to \mathbf{t} represents a lower bound on $\sum_{j \in \mathcal{J}} f_j(c_j)$. This bound is tight if $f_j(c_j) = c_j$, or if \mathcal{M} is composed by a single path. Examples of such objectives include weighted total tardiness, total square tardiness, sum of (weighted) completion times, and number of late jobs.

EXAMPLE 2. In the instance depicted in Figure 1, we can apply the recurrence relation (1) to obtain $ect_{\mathbf{r}, u_1} = 4$, $ect_{\mathbf{r}, u_2} = 3$, $ect_{u_1, u_3} = 10$, $ect_{u_1, u_4} = 7$, $ect_{u_2, u_4} = 9$, $ect_{u_2, u_5} = 7$, $ect_{u_3, \mathbf{t}} = 14$, $ect_{u_4, \mathbf{t}} = 11$, and $ect_{u_5, \mathbf{t}} = 14$. The optimal makespan is $\min\{ect_{u_3, \mathbf{t}}, ect_{u_4, \mathbf{t}}, ect_{u_5, \mathbf{t}}\} = ect_{u_4, \mathbf{t}} = 11$; it corresponds to the path $(\mathbf{r}, u_1, u_4, \mathbf{t})$, which identifies the optimal ordering (j_2, j_3, j_1) . The same ordering also yields the optimal sum of setup times with a value of 2.

Suppose now that we are given due dates $\delta_{j_1} = 13$, $\delta_{j_2} = 8$, and $\delta_{j_3} = 3$. The length of an arc a is given by $l_a = \max\{0, ect_a - \delta_{val(a)}\}$, as described earlier. We have $l_{u_1, u_4} = 4$, $l_{u_2, u_4} = 1$, $l_{u_3, \mathbf{t}} = 11$, and $l_{u_5, \mathbf{t}} = 6$; all remaining arcs a are such that $l_a = 0$. The shortest path in this case is $(\mathbf{r}, u_2, u_4, \mathbf{t})$ and has a value of 1. The minimum tardiness, even though it is given by the ordering identified by this same path, (j_3, j_2, j_1) , has a value of 3.

The reason for this gap is that the ordering with minimum tardiness does not necessarily coincide with the schedule corresponding to the earliest completion time. Namely, we computed $l_{u_4, \mathbf{t}} = 0$ considering $ect_{u_4, \mathbf{t}} = 11$, since the completion time of the job $val(u_4, \mathbf{t}) = j_1$ is 11 in (j_2, j_3, j_1) . However, in the optimal ordering (j_3, j_2, j_1) for total tardiness, the completion time of j_1 would be 15; this solution yields a better cost than (j_2, j_3, j_1) due to the reduction on the tardiness of j_3 . \square

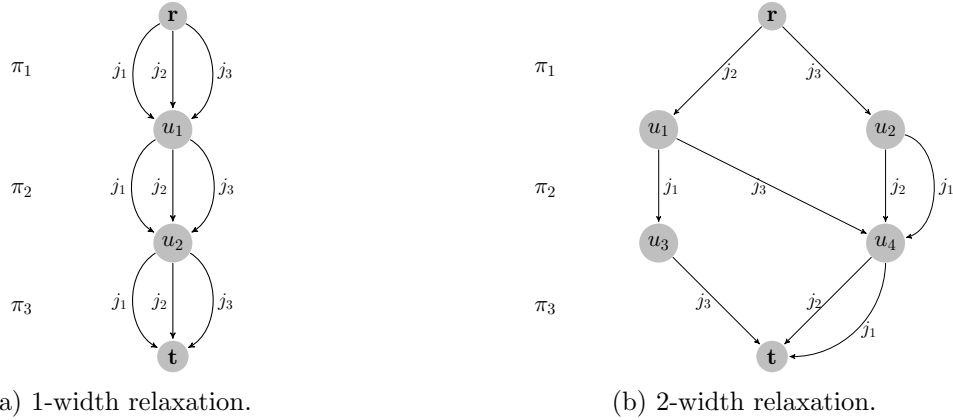


Figure 2 Two relaxed MDDs for the sequencing problem in Figure 1.

5. Relaxed MDDs

A *relaxed MDD* is an MDD \mathcal{M} that represents a superset of the feasible orderings of \mathcal{J} ; i.e., every feasible ordering is identified by some path in \mathcal{M} , but not necessarily all paths in \mathcal{M} identify a feasible ordering. We construct relaxed MDDs by limiting the size to a fixed maximum allowed width W . Thus, the strength of the relaxed MDD can be controlled by increasing W ; we obtain an exact MDD by setting W to infinity.

Figures 2a and 2b present two examples of a relaxed MDD with maximum width $W = 1$ and $W = 2$, respectively, for the problem depicted in Figure 1. In particular, the MDD in Figure 2a encodes all the orderings represented by permutations of \mathcal{J} with repetition, hence it trivially contains the feasible orderings of any sequencing problem. It can be generally constructed as follows: We create one node u_i for each layer L_i and connect the pair of nodes u_i and u_{i+1} , $i = 1, \dots, n$, with arcs a_1, \dots, a_n such that $val(a_i) = j_k$ for each job j_k .

It can also be verified that the MDD in Figure 2b contains all the feasible orderings of the instance in Figure 1. However, the right-most path going through nodes \mathbf{r} , u_2 , u_4 , and \mathbf{t} identifies an ordering $\pi = (j_3, j_1, j_1)$, which is infeasible as job j_1 is assigned twice in π .

The procedures in Section 4 for computing the optimal makespan and the optimal sum of setup times now yield a lower bound on such values when applied to a relaxed MDD, since all feasible orderings of \mathcal{J} are encoded in the diagram. Moreover, the lower bounding technique for total tardiness remains valid.

Considering that a relaxed MDD \mathcal{M} can be easily constructed for any sequencing problem (e.g., the 1-width relaxation of Figure 2a), we will now present techniques to modify \mathcal{M} in order to strengthen the relaxation it provides while observing the maximum width W . These are based on the compilation procedures developed by Hadzic et al. (2008) and Hoda et al. (2010) for general constraint satisfaction systems. Under certain conditions, we obtain the reduced MDD representing exactly the feasible orderings of \mathcal{J} , provided that W is sufficiently large.

Namely, we modify a relaxed MDD \mathcal{M} by applying the operations of *filtering* and *refinement*, which aim at approximating \mathcal{M} to an *exact* MDD, i.e., one that exactly represents the feasible orderings of \mathcal{J} . They are described as follows.

- *Filtering.* We write that an arc a is *infeasible* if all the paths in \mathcal{M} containing a represent orderings that are not feasible. Filtering consists of identifying infeasible arcs and removing them from \mathcal{M} , which would hence eliminate one or more infeasible orderings that are encoded in \mathcal{M} . We will provide details on the filtering operation in Section 6.

- *Refinement.* A relaxed MDD can be intuitively perceived as a diagram obtained by merging non-equivalent nodes of an exact MDD for the problem. Refinement consists of identifying these nodes in \mathcal{M} that are encompassing multiple equivalence classes, and *splitting* them into two or more new nodes to represent such classes more accurately (as long as the maximum width W is not violated). In particular, a node u in layer L_i can be split if there exist two partial orderings π'_1, π'_2 identified by paths from \mathbf{r} to u such that, for some $\pi^* = (\pi_i, \dots, \pi_n)$, (π'_1, π^*) is a feasible ordering while (π'_2, π^*) is not. If this is the case, then the partial paths in \mathcal{M} representing such orderings must end in different nodes of the MDD, which will be necessarily non-equivalent by definition. We will provide details on the refinement operation in Section 7.

Observe that if a relaxed MDD \mathcal{M} does not have any infeasible arcs and no nodes require splitting, then by definition \mathcal{M} is exact. However, it may not necessarily be reduced.

Filtering and refinement are independent operations that can be applied to \mathcal{M} in any order that is suitable for the problem at hand. In this work we assume a top-down approach: We traverse layers L_2, \dots, L_{n+1} one at a time in this order. At each layer L_i , we first apply filtering to remove infeasible arcs that are directed to the nodes in L_i . After the filtering is complete, we perform refinement to split the nodes in layer L_i as necessary, while observing the maximum width W .

EXAMPLE 3. Figure 3 illustrates the top-down application of filtering and refinement for layers L_2 and L_3 . Assume a scheduling problem with three jobs $\mathcal{J} = \{j_1, j_2, j_3\}$ and subject to a single precedence constraint stating that job j_2 must precede job j_1 . The initial relaxed MDD is an 1-width relaxation depicted in Figure 3a. Our maximum width is set to $W = 2$.

We start by processing the incoming arcs at layer L_2 . The filtering operation detects that the arc $a \in in(u)$ with $val(a) = j_1$ is infeasible, otherwise we will have an ordering starting with job j_1 , violating the precedence relation. Refinement will split node u into nodes u_1 and u_2 , since for any feasible ordering starting with job j_2 , i.e. (j_2, π') for some π' , the ordering (j_3, π') is infeasible as it will necessarily assign job j_3 twice. The resulting MDD is depicted in Figure 3b. Note that when a node is split, we replicate its outgoing arcs to each of the new nodes.

We now process the incoming arcs at layer L_3 . The filtering operation detects that the arc with label j_2 directed out of u_1 and the arc with label j_3 directed out of u_2 are infeasible, since the

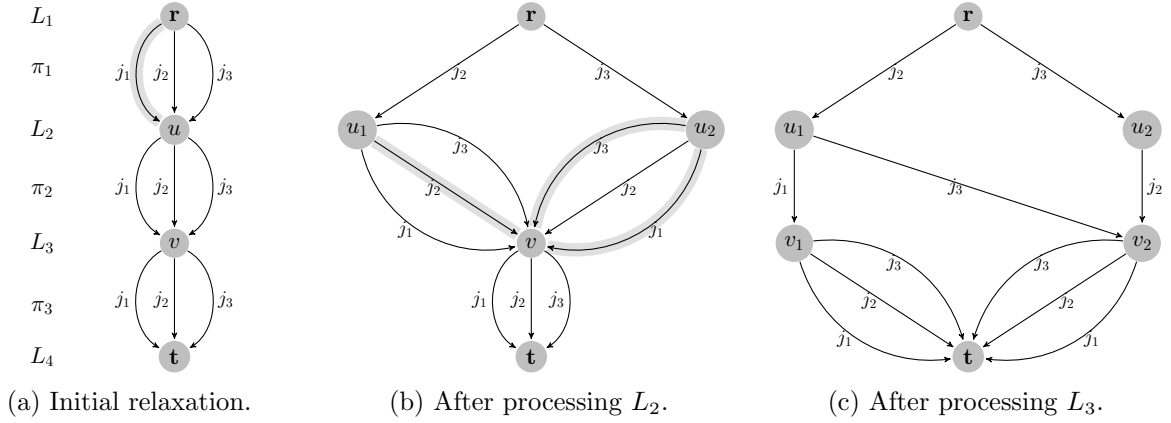


Figure 3 Example of filtering and refinement. The scheduling problem is such that job j_2 must precede j_1 in all feasible orderings. Shaded arrows represent infeasible arcs detected by the filtering.

corresponding paths from r to v would yield orderings that assign some job twice. The arc with label j_1 leaving node u_2 is also infeasible, since we cannot have any ordering with prefix (j_3, j_1) . Finally, refinement will split node v into nodes v_1 and v_2 ; note in particular that the feasible orderings prefixed by (j_2, j_3) and (j_3, j_2) have the same completions, namely (j_1) , therefore the corresponding paths end at the same node v_1 . The resulting MDD is depicted in Figure 3c. \square

6. Filtering

In this section we apply a methodology derived from Andersen et al. (2007) and Hoda et al. (2010) to identify necessary conditions for the infeasibility of an arc in \mathcal{M} . This is done as follows. For each constraint type \mathcal{C} , we equip the nodes and arcs of \mathcal{M} with a *state information* $S_{\mathcal{C}}$. Each state $S_{\mathcal{C}}$ is then considered separately to identify conditions that deem an arc as infeasible according to the particular structure of \mathcal{C} . Note that, in general, we are not able to derive efficient infeasibility conditions that are necessary and sufficient for all constraints of the problem (if $P \neq NP$), since it is NP-hard to decide if there is a feasible solution to a scheduling problem with arbitrary release dates and deadlines.

The tests presented here can be computed in polynomial-time in the size of the relaxed MDD \mathcal{M} . Namely, we restrict our state definitions to those with size $O(|\mathcal{J}|)$ and that satisfy a *Markovian* property, in that they only depend on the states of the nodes and arcs in the adjacent layers. Thus, the states can be computed simultaneously with the filtering and refinement operations during the top-down approach described in Section 5. We also describe additional states that are obtained through an extra bottom-up traversal of the MDD and that, when combined with the top-down states, lead to stronger tests.

6.1. Filtering invalid permutations

The feasible orderings of any sequencing problem are permutations of \mathcal{J} without repetition, which can be perceived as an implicit constraint to be observed. To identify conditions that indicate when all orderings identified by paths having an arc a always assign some job more than once, we define the states introduced by Andersen et al. (2007) and Hoda et al. (2010) for *AllDifferent* constraints in the area of constraint programming.

Let us associate two states $All_u^\downarrow \subseteq \mathcal{J}$ and $Some_u^\downarrow \subseteq \mathcal{J}$ to each node u of \mathcal{M} . The state All_u^\downarrow is the set of arc labels that appear in *all* paths from the root node \mathbf{r} to u , while the state $Some_u^\downarrow$ is the set of arc labels that appear in *some* path from the root node \mathbf{r} to u . For example, in Figure 3b without the shaded arcs, $All_v^\downarrow = \{j_2\}$ and $Some_v^\downarrow = \{j_1, j_2, j_3\}$ for node v .

We trivially have $All_{\mathbf{r}}^\downarrow = Some_{\mathbf{r}}^\downarrow = \emptyset$. Furthermore, it follows from the definitions that All_v^\downarrow and $Some_v^\downarrow$ for some node $v \neq \mathbf{r}$ can be recursively computed through the relations

$$All_v^\downarrow = \bigcap_{a=(u,v) \in in(v)} (All_u^\downarrow \cup \{val(a)\}), \quad (3)$$

$$Some_v^\downarrow = \bigcup_{a=(u,v) \in in(v)} (Some_u^\downarrow \cup \{val(a)\}). \quad (4)$$

LEMMA 1. *An arc $a = (u, v)$ is infeasible if any of the following conditions holds:*

$$val(a) \in All_u^\downarrow, \quad (5)$$

$$|Some_u^\downarrow| = \ell(a) \quad \text{and} \quad val(a) \in Some_u^\downarrow. \quad (6)$$

Proof. The proof argument follows from Andersen et al. (2007). Let π' be any partial ordering identified by a path from \mathbf{r} to u that does not assign any job more than once. In condition (5), $val(a) \in All_u^\downarrow$ indicates that $val(a)$ is already assigned to some position in π' , therefore appending the arc label $val(a)$ to π' will necessarily induce a repetition. For condition (6), notice first that the paths from \mathbf{r} to u are composed of $\ell(a)$ arcs, and therefore π' represents an ordering with $\ell(a)$ positions. If $|Some_u^\downarrow| = \ell(a)$, then any $j \in Some_u^\downarrow$ is already assigned to some position in π' , hence appending $val(a)$ to π' also induces a repetition. \square

We are also able to obtain stronger tests by equipping the nodes with additional states that can be derived from a bottom-up perspective of the MDD. Namely, as in Hoda et al. (2010), we define two new states $All_u^\uparrow \subseteq \mathcal{J}$ and $Some_u^\uparrow \subseteq \mathcal{J}$ for each node u of \mathcal{M} . They are equivalent to the states All_u^\downarrow and $Some_u^\downarrow$, but now they are computed with respect to the paths from \mathbf{t} to u instead of the paths from \mathbf{r} to u . As before, they are recursively obtained through the relations

$$All_u^\uparrow = \bigcap_{a=(u,v) \in out(u)} (All_v^\uparrow \cup \{val(a)\}), \quad (7)$$

$$Some_u^\uparrow = \bigcup_{a=(u,v) \in out(u)} (Some_v^\uparrow \cup \{val(a)\}), \quad (8)$$

which can be computed by a bottom-up breadth-first search before the top-down procedure.

LEMMA 2. An arc $a = (u, v)$ is infeasible if any of the following conditions holds:

$$val(a) \in All_v^\uparrow, \quad (9)$$

$$|Some_v^\uparrow| = n - \ell(a) \quad \text{and} \quad val(a) \in Some_v^\uparrow, \quad (10)$$

$$|Some_u^\downarrow \cup \{val(a)\} \cup Some_v^\uparrow| < n. \quad (11)$$

Proof. The proofs for conditions (9) and (10) follow from an argument in Hoda et al. (2010) and are analogous to the proof of Lemma 1. Condition (11) implies that any ordering identified by a path containing a will never assign all jobs \mathcal{J} . \square

6.2. Filtering precedence constraints

Suppose now we are given a set of precedence constraints, where we write $j \ll j'$ if a job j should precede job j' in any feasible ordering. We assume the precedence relations are not trivially infeasible, i.e. there are no cycles of the form $j \ll j_1 \ll \dots \ll j_m \ll j$. We can apply the same states defined in Section 6.1 for this particular case.

LEMMA 3. An arc $a = (u, v)$ is infeasible if any of the following conditions hold:

$$\exists j \in (\mathcal{J} \setminus Some_u^\downarrow) \text{ s.t. } j \ll val(a), \quad (12)$$

$$\exists j \in (\mathcal{J} \setminus Some_v^\uparrow) \text{ s.t. } val(a) \ll j. \quad (13)$$

Proof. Let π' be any partial ordering identified by a path from \mathbf{r} to u , and consider (12). By definition of $Some_u^\downarrow$, we have that any job j in the set $(\mathcal{J} \setminus Some_u^\downarrow)$ is not assigned to any position in π' . Thus, if any of such jobs j must precede $val(a)$, then all orderings prefixed by $(\pi', val(a))$ will violate this precedence constraint, and the arc is infeasible. The condition (13) is the symmetrical version of (12). \square

6.3. Filtering time window constraints

Consider now that a deadline d_j is imposed for each job $j \in \mathcal{J}$. With each arc a we associate the state ect_a as defined in Section 4: It corresponds to the minimum completion time of the job in the $\ell(a)$ -th position among all orderings that are identified by paths in \mathcal{M} containing the arc a . As in relation (1), the state ect_a for an arc $a = (u, v)$ is given by the recurrence

$$ect_a = \begin{cases} r_{val(a)} + p_{val(a)} & \text{if } a \in out(\mathbf{r}), \\ \max\{r_{val(a)}, \min\{ect_{a'} + t_{val(a'), val(a)} : a' \in in(u), val(a) \neq val(a')\}\} + p_{val(a)} & \text{otherwise,} \end{cases}$$

where we added the trivial condition $val(a) \neq val(a')$ to strengthen the bound on the time above. We could also include the condition $val(a) \ll val(a')$ if precedence constraints are imposed over $val(a)$.

We next consider a symmetrical version of ect_a to derive a necessary infeasibility condition for time window constraints. Namely, with each arc a we associate the state lst_a , which represents the *latest start time* of a : For all orderings that are identified by paths in \mathcal{M} containing the arc a , the value lst_a corresponds to an upper bound on the maximum start time of the job in the $\ell(a)$ -th position so that no deadlines are violated in such orderings. The state lst_a for an arc $a = (u, v)$ is given by the following recurrence, which can be computed through a single bottom-up traversal of \mathcal{M} :

$$lst_a = \begin{cases} d_{val(a)} - p_{val(a)} & \text{if } a \in in(\mathbf{t}), \\ \min\{d_{val(a)}, \max\{lst_{a'} - t_{val(a), val(a')} : a' \in out(v), val(a) \neq val(a')\}\} - p_{val(a)} & \text{otherwise.} \end{cases}$$

LEMMA 4. *An arc $a = (u, v)$ is infeasible if*

$$ect_a > lst_a + p_{val(a)}. \quad (14)$$

Proof. The value $lst_a + p_{val(a)}$ represents an upper bound on the the maximum time the job $val(a)$ can be completed so that no deadlines are violated in the orderings identified by paths in \mathcal{M} containing a . Since ect_a is the minimum time that job $val(a)$ will be completed among all such orderings, no feasible ordering identified by a path traversing a exists if rule (14) holds. \square

6.4. Filtering objective function bounds

Let z^* be an upper bound of the objective function value (e.g., corresponding to the best feasible solution found during the search for an optimal solution). Given z^* , an arc a is infeasible with respect to the objective if all paths in \mathcal{M} that contain a have objective value greater than z^* . However, the associated filtering method depends on the form of the objective function. For example, if the objective is to minimize makespan, we can replace the deadline d_j by $d'_j = \min\{d_j, z^*\}$ for all jobs j and consider the same infeasibility condition in Lemma 4.

If z^* corresponds to an upper bound on the sum of setup times, we proceed as follows. For each arc $a = (u, v)$ in \mathcal{M} , let st_a^\downarrow be the minimum possible sum of setup times incurred by the partial orderings represented by paths from \mathbf{r} to v that contain a . We have

$$st_a^\downarrow = \begin{cases} 0, & \text{if } a \in out(\mathbf{r}), \\ \min\{t_{val(a'), val(a)} + st_{a'}^\downarrow : a' \in in(u), val(a) \neq val(a')\}, & \text{otherwise.} \end{cases}$$

Now, for each arc $a = (u, v)$ let st_a^\uparrow be the minimum possible sum of setup times incurred by the partial orderings represented by paths from u to \mathbf{t} that contain a . The state st_a^\uparrow is given below, computed through a bottom-up traversal of \mathcal{M} :

$$st_a^\uparrow = \begin{cases} 0, & \text{if } a \in in(\mathbf{t}), \\ \min\{t_{val(a), val(a')} + st_{a'}^\uparrow : a' \in out(v), val(a) \neq val(a')\}, & \text{otherwise.} \end{cases}$$

LEMMA 5. *An arc a is infeasible if*

$$st_a^\downarrow + st_a^\uparrow > z^*. \quad (15)$$

Proof. It follows directly from the definitions of st_a^\downarrow and st_a^\uparrow . \square

To impose an upper bound z^* on the total tardiness, assume ect_a is computed for each arc a . We define the length of an arc a as $l_a = \max\{0, ect_a - \delta_{val(a)}\}$. For a node u , let sp_u^\downarrow and sp_u^\uparrow be the shortest path from \mathbf{r} to u and from \mathbf{t} to u , respectively, with respect to the lengths l_a . That is,

$$sp_u^\downarrow = \begin{cases} 0, & \text{if } u = \mathbf{r}, \\ \min\{l_a + sp_v^\downarrow : a = (v, u) \in in(u)\}, & \text{otherwise.} \end{cases}$$

and

$$sp_u^\uparrow = \begin{cases} 0, & \text{if } u = \mathbf{t}, \\ \min\{l_a + sp_v^\uparrow : a = (u, v) \in out(u)\}, & \text{otherwise.} \end{cases}$$

LEMMA 6. *A node u should be removed from \mathcal{M} if*

$$sp_u^\downarrow + sp_u^\uparrow > z^*, \quad (16)$$

Proof. Length l_a represents a lower bound on the tardiness of job $val(a)$ with respect to solutions identified by \mathbf{r} - \mathbf{t} paths that contain a . Thus, sp_u^\downarrow and sp_u^\uparrow are a lower bound on the total tardiness for the partial orderings identified by paths from \mathbf{r} to u and \mathbf{t} to u , respectively, since the tardiness of a job is non-decreasing on its completion time. \square

7. Refinement

Recall from Section 5 that a relaxed MDD can be strengthened by a refinement operation. Ideally, refinement should modify a layer so that each of its nodes exactly represents a particular equivalence class. However, as it may be necessary to create an exponential number of nodes to represent all equivalence classes, we apply in this section a heuristic refinement procedure that observes the maximum width W when creating new nodes in a layer.

Our goal is to be as precise as possible with respect to the equivalence classes that refer to jobs with a higher *priority*, where the priority of a job is defined according to the problem data. More specifically, we will develop a refinement heuristic that, when combined with the infeasibility conditions for the permutation structure described in Section 6.1, yields a relaxed MDD where the jobs with a high priority are represented exactly with respect to that structure; that is, these jobs are assigned to exactly one position in all orderings encoded by the relaxed MDD.

Thus, if higher priority is given to jobs that play a greater role in the feasibility or optimality of the sequencing problem at hand, the relaxed MDD may represent more accurately the feasible orderings of the problem, providing, e.g., better bounds on the objective function value. For

example, suppose we wish to minimize the makespan on an instance where certain jobs have a very large release date and processing times in comparison to other jobs. If we construct a relaxed MDD where these longer jobs are assigned exactly once in all orderings encoded by the MDD, the bound on the makespan would be potentially tighter with respect to the ones obtained from other possible relaxed MDDs for this same instance. Examples of job priorities for other objective functions are presented in Section 10.

To achieve this property, the refinement heuristic we develop is based on the following theorem, which we will prove constructively later. Note that it takes into account the maximum width W .

THEOREM 1. *Let $W > 0$. There exists a relaxed MDD \mathcal{M} where at least $\lfloor \log_2 W \rfloor$ jobs are assigned to exactly one position in all orderings identified by \mathcal{M} .*

Let us represent the job priorities by defining a *ranking* of jobs $\mathcal{J}^* = \{j_1^*, \dots, j_n^*\}$, where jobs with smaller index in \mathcal{J}^* have a higher priority. We can thus achieve the desired property of our heuristic refinement by constructing the relaxed MDD \mathcal{M} based on Theorem 1, where we ensure that the jobs exactly represented in \mathcal{M} are those with a higher ranking.

Before proving Theorem 1, we first identify conditions on when a node violates the desired refinement property and needs to be modified. To this end, let \mathcal{M} be any relaxed MDD. Assume the states All_u^\downarrow and $Some_u^\downarrow$ as described in Section 6.1 are computed for all nodes u in \mathcal{M} , and no arcs satisfy the infeasibility conditions (5) to (11). We have the following Lemma.

LEMMA 7. *A job j is assigned to exactly one position in all orderings identified by \mathcal{M} if and only if $j \notin Some_u^\downarrow \setminus All_u^\downarrow$ for all nodes $u \in \mathcal{M}$.*

Proof. Suppose first that a job j is assigned to exactly one position in all orderings identified by \mathcal{M} , and take a node u in \mathcal{M} such that $j \in Some_u^\downarrow$. From the definition of $Some_u^\downarrow$, there exists a path from \mathbf{r} to u with an arc labeled j . This implies by hypothesis that all paths from u to \mathbf{t} do not have any arcs labeled j , otherwise we will have a path that identifies an ordering where j is assigned more than once. But then, also by hypothesis, all paths from \mathbf{r} to u must necessarily have some arc labeled j , thus $j \in All_u^\downarrow$, which implies $j \notin Some_u^\downarrow \setminus All_u^\downarrow$.

Conversely, suppose $j \in Some_u^\downarrow \setminus All_u^\downarrow$ for all nodes u in \mathcal{M} . Then a node u can only have an outgoing arc a with $val(a) = j$ if $j \notin Some_u^\downarrow$, which is due to the filtering rule (5). Thus, no job is assigned more than once in any ordering encoded by \mathcal{M} . Finally, rule (11) ensures that j is assigned exactly once in all paths. \square

We now provide a constructive proof for Theorem 1.

Proof of Theorem 1 Let \mathcal{M} be an 1-width relaxation. We can obtain the desired MDD applying filtering and refinement on \mathcal{M} in a top-down approach as described in Section 5. For filtering,

remove all arcs satisfying the infeasibility rules in Section 6. For refining a particular layer L_i , apply the following procedure: For each job $j = j_1, \dots, j_n$ in this order, select a node $u \in L_i$ such that $j \in \text{Some}_u^\downarrow \setminus \text{All}_u^\downarrow$. Create two new nodes u_1 and u_2 , and redirect the incoming arcs at u to u_1 and u_2 as follows: if the arc $a = (v, u)$ is such that $j \in (\text{All}_v^\downarrow \cup \{\text{val}(a)\})$, redirect it to u_1 ; otherwise, redirect it to u_2 . Replicate all the outgoing arcs of u to u_1 and u_2 , remove u , and repeat this until the maximum width W is met, there are no nodes satisfying this for j , or all jobs were considered.

We now show that this refinement procedure suffices to produce a relaxed MDD satisfying the conditions of the Theorem. Observe first the conditions of Lemma 7 are satisfied by any job at the root node \mathbf{r} , since $\text{Some}_\mathbf{r}^\downarrow = \emptyset$. Suppose, by induction hypothesis, that the conditions of Lemma 7 are satisfied for some job j at all nodes in layers $L_1, \dots, L_{i'}$, $i' < i$, and consider we created nodes u_1 and u_2 from some node $u \in L_i$ such that $j \in \text{Some}_u^\downarrow \setminus \text{All}_u^\downarrow$ as described above. By construction, any incoming arc $a = (v, u_2)$ at u_2 satisfies $j \notin (\text{All}_v^\downarrow \cup \{\text{val}(a)\})$; by induction hypothesis, $j \notin \text{Some}_v^\downarrow$, hence $j \notin \text{Some}_{u_2}^\downarrow \setminus \text{All}_{u_2}^\downarrow$ by relation (3). Analogously, we can show $j \in \text{All}_{u_1}^\downarrow$, thus $j \notin \text{Some}_{u_1}^\downarrow \setminus \text{All}_{u_1}^\downarrow$.

Since the jobs \mathcal{J} are processed in the same order for all layers, we just need now to compute the minimum number of jobs for which all nodes violating Lemma 7 were split when the maximum width W was attained. Just observe that, after all the nodes were verified with respect to a job, we at most duplicated the number of nodes in a layer (since each split produces one additional node). Thus, if m jobs were considered, we have at most 2^m nodes in a layer, thus at least $\lceil \log_2 W \rceil$ nodes will be exactly represented in \mathcal{M} . \square

We can utilize Theorem 1 to guide our top-down approach for filtering and refinement, following the refinement heuristic based on the job ranking \mathcal{J}^* described in the proof of Theorem 1. Namely, we apply the following refinement at a layer L_i : For each job $j^* = j_1^*, \dots, j_n^*$ in the order defined by \mathcal{J}^* , identify the nodes u such that $j^* \in \text{Some}_u^\downarrow \setminus \text{All}_u^\downarrow$ and split them into two nodes u_1 and u_2 , where an incoming arc $a = (v, u)$ is redirected to u_1 if $j^* \in (\text{All}_v^\downarrow \cup \{\text{val}(a)\})$ or u_2 otherwise, and replicate all outgoing arcs for both nodes. Moreover, if the relaxed MDD is a 1-width-relaxation, then we obtain the bound guarantee on the number of jobs that are exactly represented.

This procedure also yields a *reduced* MDD \mathcal{M} for certain structured problems. The following corollary, stated without proof, is directly derived from Lemma 7 and Theorem 1.

COROLLARY 1. *Assume $W = +\infty$. For a sequencing problem having only precedence constraints, the relaxed MDD \mathcal{M} that results from the constructive proof of Theorem 1 is a reduced MDD that exactly represents the feasible orderings of this problem.*

Lastly, recall that equivalence classes corresponding to constraints other than the permutation structure may also be taken into account during refinement. Therefore, if the maximum width

W is not met in the refinement procedure above, we assume that we will further split nodes by arbitrarily partitioning their incoming arcs. Even though this may yield false equivalence classes, the resulting \mathcal{M} is still a valid relaxation and may provide a stronger representation.

8. Inferring Precedence Relations from Relaxed MDDs

Given a set of precedence relations to a problem (e.g., that were possibly derived from other relaxations), we can use the filtering rules (12) and (13) from Section 6.2 to strengthen a relaxed MDD. In this section, we show that a converse relation is also possible. Namely, given a relaxed MDD \mathcal{M} , we can deduce all precedence relations that are satisfied by the partial orderings represented by \mathcal{M} in polynomial time in the size of \mathcal{M} . To this end, assume that the states All_u^\downarrow , All_u^\uparrow , $Some_u^\downarrow$, and $Some_u^\uparrow$ as described in Section 6.1 are computed for all nodes u in \mathcal{M} . We have the following results.

THEOREM 2. *Let \mathcal{M} be an MDD that exactly identifies all the feasible orderings of \mathcal{J} . A job j must precede job j' in any feasible ordering if and only if $(j' \notin All_u^\downarrow)$ or $(j \notin All_u^\uparrow)$ for all nodes u in \mathcal{M} .*

Proof. Suppose there exists a node u in layer L_i , $i \in \{1, \dots, n+1\}$, such that $j' \in All_u^\downarrow$ and $j \in All_u^\uparrow$. By definition, there exists a path $(\mathbf{r}, \dots, u, \dots, \mathbf{t})$ that identifies an ordering where job j' starts before job j . This can only be true if and only if job j does not precede j' in any feasible ordering. \square

COROLLARY 2. *The set of all precedence relations that must hold in any feasible ordering can be extracted from \mathcal{M} in $O(n^2 |\mathcal{M}|)$.*

Proof. Construct a digraph $G^* = (\mathcal{J}, E^*)$ by adding an arc (j, j') to E^* if and only if there exists a node u in \mathcal{M} such that $j' \in All_u^\downarrow$ and $j \in All_u^\uparrow$. Checking this condition for all pair of jobs takes $O(n^2)$ for each node in \mathcal{M} , and hence the time complexity to construct G^* is $O(n^2 |\mathcal{M}|)$. According to Theorem 2 and the definition of G^* , the complement graph of G^* contains an edge (j, j') if and only if $j \ll j'$. \square

As we are mainly interested in relaxed MDDs, we derive an additional corollary of Theorem 2.

COROLLARY 3. *Given a relaxed MDD \mathcal{M} , an activity j must precede activity j' in any feasible solution if $(j' \notin Some_u^\downarrow)$ or $(j \notin Some_u^\uparrow)$ for all nodes u in \mathcal{M} .*

Proof. It follows from the state definitions that $All_u^\downarrow \subseteq Some_u^\downarrow$ and $All_u^\uparrow \subseteq Some_u^\uparrow$. Hence, if the conditions for the relation $j \ll j'$ from Theorem 2 are satisfied by $Some_u^\downarrow$ and $Some_u^\uparrow$, they must be also satisfied by any MDD which only identifies feasible orderings. \square

By Corollary 3, the precedence relations implied by the solutions of a relaxed MDD \mathcal{M} can be extracted by applying the algorithm in Corollary 2 to the states $Some_v^\downarrow$ and $Some_v^\uparrow$. Since \mathcal{M} has at most $O(nW)$ nodes and $O(nW^2)$ arcs, the time to extract the precedences has a worst-case complexity of $O(n^3W^2)$ by the presented algorithm. These precedences can then be used for guiding search or communicated to other methods or relaxations that may benefit from them.

9. Encoding Size for Structured Precedence Relations

The actual constraints that define a problem instance greatly impact the size of an MDD. If these constraints carry a particular structure, we may be able to compactly represent that structure in an MDD, perhaps enabling us to bound its width.

In this section we present one of such cases for a problem class introduced by Balas (1999), in which jobs are subject to *discrepancy* precedence constraints: For a fixed parameter $k \in \{1, \dots, n\}$, the relation $j_p \ll j_q$ must be satisfied for any two jobs $j_p, j_q \in \mathcal{J}$ if $q \geq p + k$. This precedence structure was motivated by a real-world application in steel rolling mill scheduling. The work by Balas and Simonetti (2000) also demonstrates how solution methods to this class of problems can serve as auxiliary techniques in other cases, for example as heuristics for the traveling salesman problem and vehicle routing with time windows.

We stated in Corollary 1 that we are able to construct the reduced MDD \mathcal{M} when only precedence constraints are imposed and a sufficiently large W is given. We have the following results for \mathcal{M} if the precedence relations satisfy the discrepancy structure for a given k .

LEMMA 8. *For a node $v \in L_{m+1}$, $m = 1, \dots, n$, we have $All_v^\downarrow \subseteq \{j_1, \dots, j_{\min\{m+k-1, n\}}\}$.*

Proof. If $m + k - 1 > n$ we obtain the redundant condition $All_u^\downarrow \subseteq \mathcal{J}$, therefore assume $m + k - 1 \leq n$. Suppose there exists $j_l \in All_v^\downarrow$ for some $v \in L_{m+1}$ such that $l > m + k - 1$. Then, for any $i = 1, \dots, m$, we have $l - i \geq m + k - i \geq m + k - m = k$. This implies $\{j_1, \dots, j_m\} \subset All_v^\downarrow$, since job j_i belongs to a partial ordering π only if all jobs j_i for which $l - i \geq k$ are already accounted in π . But then $|All_v^\downarrow| \geq m + 1$, which is a contradiction since $v \in L_{m+1}$ implies that $|All_v^\downarrow| = m$, as any partial ordering identified by a path from \mathbf{r} to v must contain m distinct jobs. \square

THEOREM 3. *The width of \mathcal{M} is 2^{k-1} .*

Proof. Let us first assume $n \geq k + 2$ and restrict our attention to layer L_{m+1} for some $m \in \{k, \dots, n - k + 1\}$. Also, let $\mathcal{F} := \{All_u^\downarrow : u \in L_{m+1}\}$. It can be shown that if \mathcal{M} is reduced, no two nodes $u, v \in L_{m+1}$ are such that $All_u^\downarrow = All_v^\downarrow$. Thus, $|\mathcal{F}| = |L_{m+1}|$.

We derive the cardinality of \mathcal{F} as follows. Take $All_v^\downarrow \in \mathcal{F}$ for some $v \in L_{m+1}$. Since $|All_v^\downarrow| = m$, there exists at least one job $j_i \in All_v^\downarrow$ such that $i \geq m$. According to Lemma 8, the maximum index of a job in All_v^\downarrow is $m + k - 1$. So consider the jobs indexed by $m + k - 1 - l$ for $l = 0, \dots, k - 1$; at

least one of them is necessarily contained in All_v^\downarrow . Due to the discrepancy precedence constraints, $j_{m+k-1-l} \in All_v^\downarrow$ implies that any j_i with $i \leq m-l-1$ is also contained in All_v^\downarrow (if $m-l-1 > 0$).

Now, consider the sets in \mathcal{F} which contain a job with index $m+k-1-l$, but do *not* contain any job with index greater than $m+k-1-l$. Any of such set All_u^\downarrow contain the jobs j_1, \dots, j_{m-l-1} according to Lemma 8. Hence, the remaining $m - (m-l-1) - 1 = l$ job indices can be freely chosen from $m-l, \dots, m+k-l-2$. Notice there are no imposed precedences on these remaining $m+k-l-2 - (m-l) + 1 = k-1$ elements; thus, there exist $\binom{k-1}{l}$ of such subsets. But these sets define a partition of \mathcal{F} . Therefore

$$|\mathcal{F}| = |L_{m+1}| = \sum_{l=0}^{k-1} \binom{k-1}{l} = \binom{k-1}{0} + \dots + \binom{k-1}{k-1} = 2^{k-1}.$$

We can use an analogous argument for the layers L_{m+1} such that $m < k$ or $m > n-k+1$, or when $k = n-1$. The main technical difference is that we will have less than $k-1$ possibilities for the new combinations, and hence the maximum number of nodes is strictly less than 2^{k-1} for these cases. The width of \mathcal{M} is therefore 2^{k-1} . \square

According to Theorem 3, \mathcal{M} has $O(n2^{k-1})$ nodes as it contains $n+1$ layers. Since arcs only connect nodes in adjacent layers, the MDD contains $O(n2^{2k-2})$ arcs (assuming a worst-case scenario where all nodes in a layer are adjacent to all nodes in the next layer, yielding at most $2^{k-1} \cdot 2^{k-1} = 2^{2k-2}$ arcs directed out of a layer). Using the recursive relation (2) in Section 4, we can compute, e.g., the minimum sum of setup times in worst-case time complexity of $O(n^2 2^{2k-2})$. The work by Balas (1999) provides an algorithm that minimizes this same function in $O(nk^2 2^{k-2})$, but that is restricted to this particular objective.

10. Application of Relaxed MDDs to Constraint-based Scheduling

We added the techniques described here to *ILOG CP Optimizer* (CPO), the current state-of-the-art general-purpose scheduler. Given a sequencing problem as considered in this work, CPO applies a depth-first branch-and-bound search where jobs are recursively appended to the end of a partial ordering until no jobs are left unsequenced. At each node of the branching tree, a number of sophisticated *propagators* are used to reduce the possible candidate jobs to be appended to the ordering. Examples of such propagators include *edge-finding*, *not-first/not-last rules*, and *deductible precedences*; details can be found in Baptiste et al. (2001) and Vilím (2004).

We have implemented our techniques as a user-defined propagator, which maintains a relaxed MDD and runs one round of top-down filtering and refinement when activated at each node of the branching tree. In particular, the filtering operation takes into account the search decisions up to that point (i.e., the jobs that are already fixed in the partial ordering) and possible precedence constraints that are deduced by CPO. At the end of a round, we use the relaxed MDD to reduce

the number of candidate successor jobs (by analyzing the arc labels in the appropriate layers) and to communicate new precedence constraints as described in Section 8, which may trigger additional propagation by CPO. Our implementation follows the guidelines from ILOG (2012).

In this section we present computational results for different variations of single machine sequencing problems using the MDD-based propagator. Our goal is twofold. First, we want to analyze the sensitivity of the relaxed MDD with respect to the width and refinement strategy. Second, we wish to provide experimental evidence that combining a relaxed MDD with existing techniques for sequencing problems can improve the performance of constraint-based solvers.

10.1. Experimental setup

Three formulations were considered for each problem: a CPO model with its default propagators, denoted by CPO; a CPO model containing only the MDD-based propagator, denoted by MDD; and a CPO model with the default and MDD-based propagators combined, denoted by CPO+MDD. The experiments mainly focus on the comparison between CPO and CPO+MDD, as these indicate whether incorporating the MDD-based propagator can enhance existing methods.

We have considered two heuristic strategies for selecting the next job to be appended to a partial schedule. The first, denoted by *lex search*, is a static method that always tries to first sequence the job with the smallest index, where the index of a job is fixed per instance and defined by the order in which it appears in the input. This allows for a more accurate comparison between two propagation methods, since the branching tree is fixed. In the second strategy, denoted by *dynamic search*, the CPO engine automatically selects the next job according to its own state-of-the-art scheduling heuristics. The purpose of the experiments that use this search is to verify how the MDD-based propagator is influenced by strategies that are known to be effective for constraint-based solvers. The dynamic search is only applicable to CPO and CPO+MDD.

We measure two performance indicators: the total solving time and the *number of fails*. The number of fails corresponds to the number of times during search that a partial ordering was detected to be infeasible, i.e., either some constraint is violated or the objective function is greater than a known upper bound. The number of fails is proportional to the size of the branching tree and, hence, to the total solving time of a particular technique.

The techniques presented here do not explore any additional problem structure that was not described in this work, such as specific search heuristics, problem relaxations, or dominance criteria (except only if such structure is already explored by CPO). More specifically, we used the same MDD-based propagator for all problems, which dynamically determines what node state and refinement strategy to use according to the input constraints and the objective function.

The experiments were performed on a computer equipped with an Intel Xeon E5345 at 2.33GHz with 8 Gb RAM. The MDD code was implemented in C++ using the CPO

callable library from the ILOG CPLEX Academic Studio V.12.4.01. We have set the following additional CPO parameters for all experiments: `Workers=1`, to use a single computer core; `DefaultInferenceLevel=Extended`, to use the maximum possible propagation available in CPO; and `SearchType=DepthFirst`.

10.2. Impact of the MDD Parameters

We first investigate the impact of the maximum width and refinement on the number of fails and total solving time for the MDD approaches. As a representative test case, we consider the traveling salesman problem with time windows (TSPTW). The TSPTW is the problem of finding a minimum-cost tour in a weighted digraph starting from a selected vertex (the depot), visiting each vertex within a given time window, and returning to the original vertex. In our case, each vertex is a job, the release dates and deadlines are defined according to the vertex time windows, and travel distances are perceived as setup times. The objective function is to minimize the sum of setup times.

We selected the instance `n20w200.001` from the well-known Gendreau benchmark proposed by Gendreau et al. (1998), as it represents the typical behavior of an MDD. It consists of a 20-vertex graph with an average time window width of 200 units. The tested approach was the MDD model with lex search. We used the following job ranking for the refinement strategy described in Section 7: The first job in the ranking, j_1^* , was set as the first job of the input. The i -th job in the ranking, j_i^* , is the one that maximizes the sum of the setup times to the jobs already ranked, i.e. $j_i^* = \arg \max_{p \in \mathcal{J} \setminus \{j_1^*, \dots, j_{i-1}^*\}} \{\sum_{k=1}^{i-1} t_{j_k^*, p}\}$ for the setup times t . The intuition is that we want jobs with largest travel distances to be exactly represented in \mathcal{M} .

The number of fails and total time to find the optimal solution for different MDD widths are presented in Figure 4. Due to the properties of the refinement technique in Theorem 1, we consider only powers of 2 as widths. We note from Figure 4a that the number of fails is decreasing rapidly as the width increases, up to a point where it becomes close to a constant (from 512 to 1024). This indicates that, at a certain point, the relaxed MDD is very close to an actual exact representation of the problem, and hence no benefit is gained from any increment of the width. The number of fails has a direct impact on the total solving time, as observed in Figure 4b. Namely, the times decrease accordingly as the width increases. At the point where the relaxed MDD is close to be exact, larger widths only introduce additional overhead, thus increasing the solving time.

To analyze the impact of the refinement, we generated 50 job rankings uniformly at random for the refinement strategy described in Section 7. These rankings were compared with the *structured* one for setup times used in the previous experiment. To make this comparison, we solved the MDD model with lex search for each of the 51 refinement orderings, considering widths from 4 to

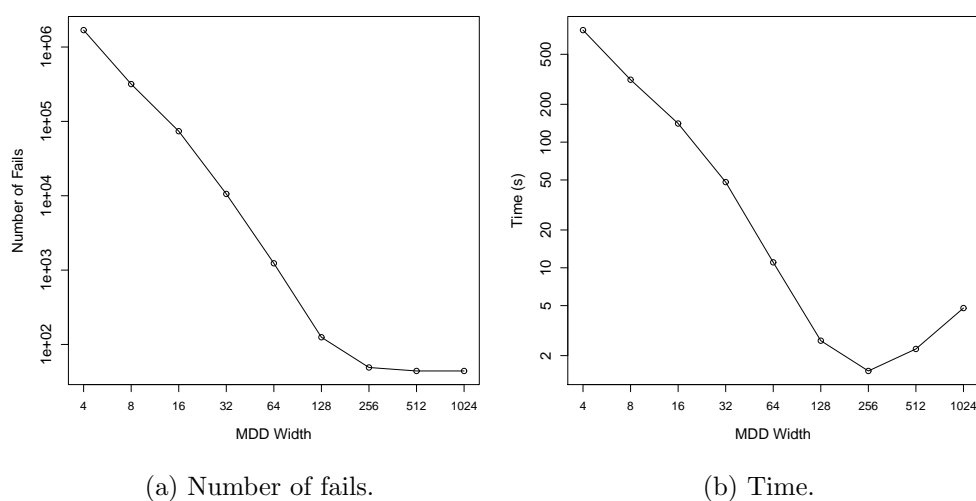


Figure 4 Impact of the MDD width on the number of fails and total time for the TSPTW instance $n20w200.001$ from the Gendreau class. The axes are in logarithmic scale.

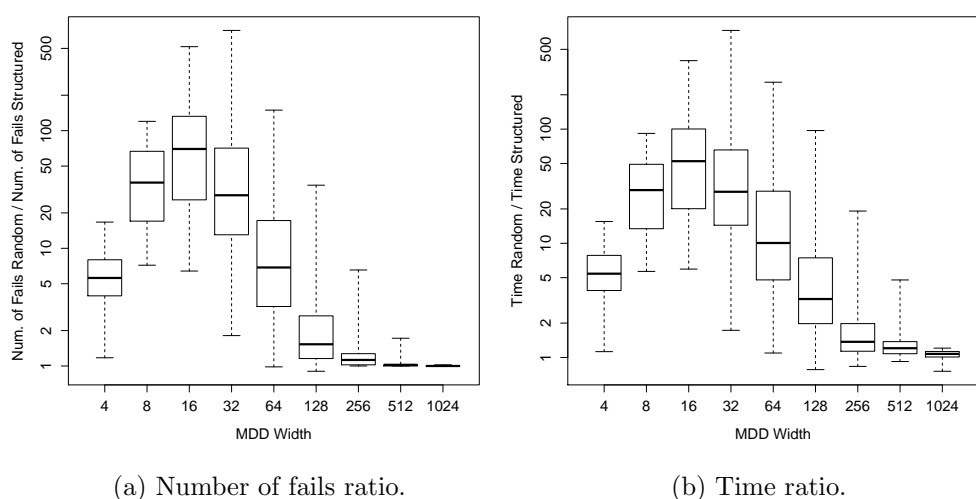


Figure 5 Performance comparison between random and structured refinement strategies for the TSPTW instance $n20w200.001$. The axes are in logarithm scale.

1024. For each random order, we divided the resulting number of fails and time by the ones obtained with the structured refinement for the same width. Thus, this ratio represents how much better the structured refinement is over the random strategies. The results are presented in the box-and-whisker plots of Figure 5. For each width the horizontal lines represent, from top to bottom, the maximum observed ratio, the upper quartile, the median ratio, the lower quartile, and the minimum ratio.

We interpret Figure 5 as follows. An MDD with very small width captures little of the jobs that play a more important role in the optimality or feasibility of the problem, in view of Theorem 1. Thus, distinct refinement strategies are not expected to differ much on average, as shown, e.g., in the width-4 case of Figure 5a. As the width increases, there is a higher chance that these crucial jobs are better represented by the MDD, leading to a good relaxation, but also a higher chance that little of their structure is captured by a random strategy, leading in turn to a weak relaxation. This yields a larger variance on the refinement performance. Finally, for sufficiently large widths, we end up with an almost exact representation of the problem and the propagation is independent of the refinement order (e.g., widths 512 and 1024 of Figure 5a). Another aspect we observe in Figure 5b is that, even for relatively small widths, the structured refinement can be orders of magnitude better than a random one. This emphasizes the importance of applying an appropriate refinement strategy for the problem at hand.

10.3. Traveling Salesman Problem with Time Windows

We first evaluate the relative performance of CPO and CPO+MDD on sequencing problems with time window constraints, and where the objective is to minimize the sum of setup times. We considered a set of well-known TSPTW instances defined by the Gendreau, Dumas, and Ascheuer benchmark classes, which were proposed by Gendreau et al. (1998), Dumas et al. (1995), and Ascheuer (1995), respectively. We selected all instances with up to 100 jobs, yielding 388 test cases in total. The CPO and the CPO+MDD models were initially solved with lex search, considering a maximum width of 16. A time limit of 1,800 seconds was imposed to all methods, and we used the structured job ranking described in Section 10.2.

The CPO approach was able to solve 26 instances to optimality, while the CPO+MDD approach solved 105 instances to optimality. The number of fails and solution times are presented in the scatter plots of Figure 6, where we only considered instances solved by both methods. The plots provide a strong indication that the MDD-based propagator can greatly enhance the CPO inference mechanism. For example, CPO+MDD can reduce the number of fails from over 10 million (CPO) to less than 100 for some instances.

In our next experiment we compared CPO and CPO+MDD considering a maximum width of 1024 and applying instead a dynamic search, so as to verify if we could still obtain additional gains with the general-purpose scheduling heuristics provided by CPO. A time limit of 1,800 seconds was imposed to all approaches.

With the above configuration, the CPO approach solved to optimality 184 out of the 388 instances, while the CPO+MDD approach solved to optimality 311 instances. Figure 7a compares the times for instances solved by both methods, while Figure 7b depicts the performance plot. In particular, the overhead introduced by the MDD is only considerable for small instances (up to 20 jobs). On the majority of the cases, the CPO+MDD is capable of proving optimality much quicker.

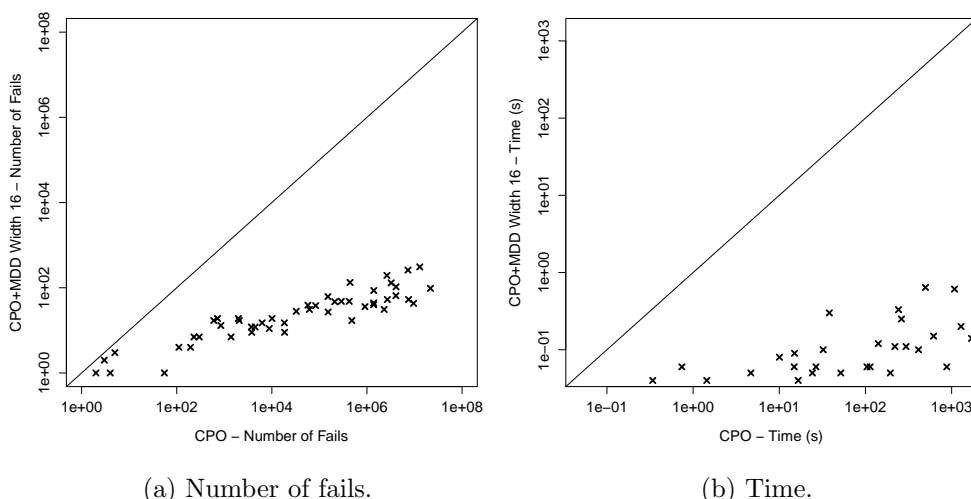


Figure 6 Performance comparison between CPO and CPO+MDD for minimizing sum of setup times on Dumas, Gendreau, and Ascheuer TSPTW classes with lex search. The vertical and horizontal axes are in logarithmic scale.

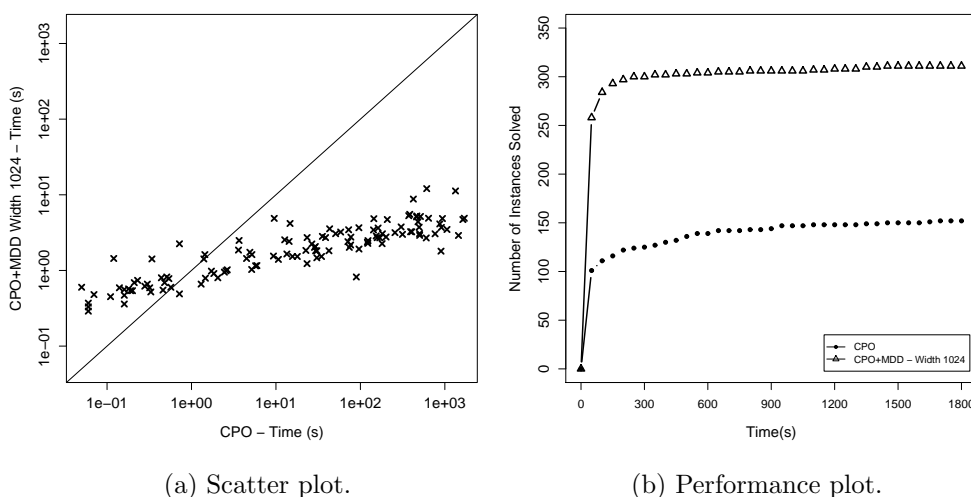


Figure 7 Performance comparison between CPO and CPO+MDD for minimizing sum of setup times on Dumas, Gendreau, and Ascheuer TSPTW classes using default depth-first CPO search. The horizontal and vertical axes in (a) are in logarithmic scale.

10.4. Asymmetric Traveling Salesman Problem with Precedence Constraints

We next evaluate the performance of CPO and CPO+MDD on sequencing problems with precedence constraints, while the objective is again to minimize the sum of setup times. As benchmark problem, we consider the asymmetric traveling salesman problem with precedence constraints (ATSP), also known as *sequential ordering problem*. The ATSP is a variation of the asymmetric TSP where

instance	vertices	bounds	CPO		CPO+MDD, width 2048	
			best	time (s)	best	time (s)
br17.10	17	55	55	0.01	55	4.98
br17.12	17	55	55	0.01	55	4.56
ESC07	7	2125	2125	0.01	2125	0.07
ESC25	25	1681	1681	TL	1681	48.42
p43.1	43	28140	28205	TL	28140	287.57
p43.2	43	[28175, 28480]	28545	TL	28480	279.18
p43.3	43	[28366, 28835]	28930	TL	28835	177.29
p43.4	43	83005	83615	TL	83005	88.45
ry48p.1	48	[15220, 15805]	18209	TL	16561	TL
ry48p.2	48	[15524, 16666]	18649	TL	17680	TL
ry48p.3	48	[18156, 19894]	23268	TL	22311	TL
ry48p.4	48	[29967, 31446]	34502	TL	31446	96.91
ft53.1	53	[7438, 7531]	9716	TL	9216	TL
ft53.2	53	[7630, 8026]	11669	TL	11484	TL
ft53.3	53	[9473, 10262]	12343	TL	11937	TL
ft53.4	53	14425	16018	TL	14425	120.79

Table 1 Results on ATSP instances. Values in bold represent instances solved for the first time.

precedence constraints must be observed. Namely, given a weighted digraph $D = (V, A)$ and a set of pairs $P = V \times V$, the ATSP is the problem of finding a minimum-weight Hamiltonian tour T such that vertex v precedes u in T if $(v, u) \in P$.

The ATSP has been shown to be extremely challenging for exact methods. In particular, a number of instances with less than 70 vertices from the well-known TSPLIB (2012) benchmark, proposed initially by Ascheuer et al. (2000), are still open. We refer to the work of Anghinolfi et al. (2011) for a more detailed literature review of exact and heuristic methods for the ATSP.

We applied the CPO and CPO+MDD model with dynamic search and a maximum width of 2048 for 16 instances of the ATSP from the TSPLIB benchmark. A time limit of 1,800 seconds was imposed, and we used the structured job ranking described in Section 10.2. The results are reported in Table 1. The column *Best* corresponds to the best solution found by the method and the column *Time* corresponds to the time that the solution was proved optimal. A value TL indicates that the time limit was reached. Since the TSPLIB results are not updated, we report updated bounds obtained from Hernádvölgyi (2003), Gouveia and Pesneau (2006), and Anghinolfi et al. (2011).

We were able to close three of the unsolved instances with our generic approach, namely p43.2, p43.3, and ry48p.4. In addition, instance p43.4 was solved before with more than 22 hours of CPU time by Hernádvölgyi (2003) (for a computer approximately 10 times slower than ours), and by more than 4 hours by Gouveia and Pesneau (2006) (for an unspecified machine), while we could solve it in less than 90 seconds. The presence of more precedence constraints (indicated for these instances by a larger suffix number) is more advantageous to our MDD approach, as shown in Table 1. On the other hand, less constrained instances are better suited to MILP-based approaches; instances p43.1 and ry48p.1 are solved by a few second in Ascheuer et al. (2000).

As a final observation, we note that the bounds for the p43.1-4 instances reported in the TSPLIB are inconsistent. They do not match any of the bounds from existing works we are aware of and the

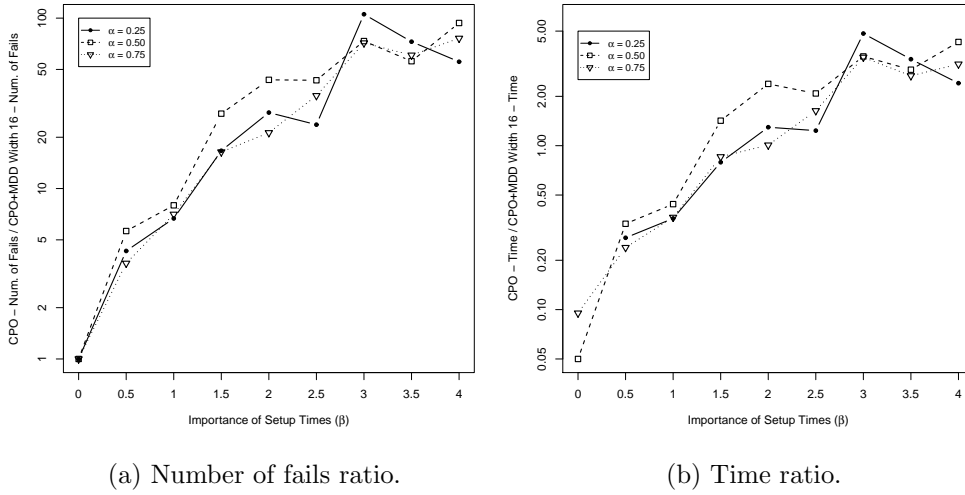


Figure 8 Comparison between CPO and CPO+MDD for minimizing makespan on three instances with randomly generated setup times. The vertical axes are in logarithmic scale.

ones provided by Ascheuer et al. (2000), from where these problems were proposed. This includes the instance p43.1 which was solved in that work.

10.5. Makespan Problems

Constraint-based solvers are known to be particularly effective when the objective function is makespan, which is greatly due to specialized domain propagation techniques that can be used in such cases; see, e.g., Baptiste et al. (2001).

In this section we evaluate the performance of CPO and CPO+MDD on sequencing problems with time window constraints and where the objective is to minimize makespan. Our goal is to test the performance of such procedures on makespan problems, and verify the influence of setup times on the relative performance. In particular, we will empirically show that the MDD-based propagator makes schedulers more robust for makespan problems especially when setup times are present.

To compare the impact of setup times between methods, we performed the following experiment. Using the scheme from Chang et al. (1995), we first generated three random instances with 15 jobs. The processing times p_i are selected uniformly at random from the set $\{1, 100\}$, and release dates are selected uniformly at random from the set $\{0, \dots, \alpha \sum_i p_i\}$ for $\alpha \in \{0.25, 0.5, 0.75\}$. No deadlines are considered. For each of the three instances above, we generated additional random instances where we add a setup time for all pairs of jobs i and j selected uniformly at random from the set $\{0, \dots, (50.5)\beta\}$, where $\beta \in \{0, 0.5, 1, \dots, 4\}$. In total, 10 instances are generated for each β . We computed the number of fails and total time to minimize the makespan using CPO and CPO+MDD models with a maximum width of 16, applying a lex search in both cases. We then divided

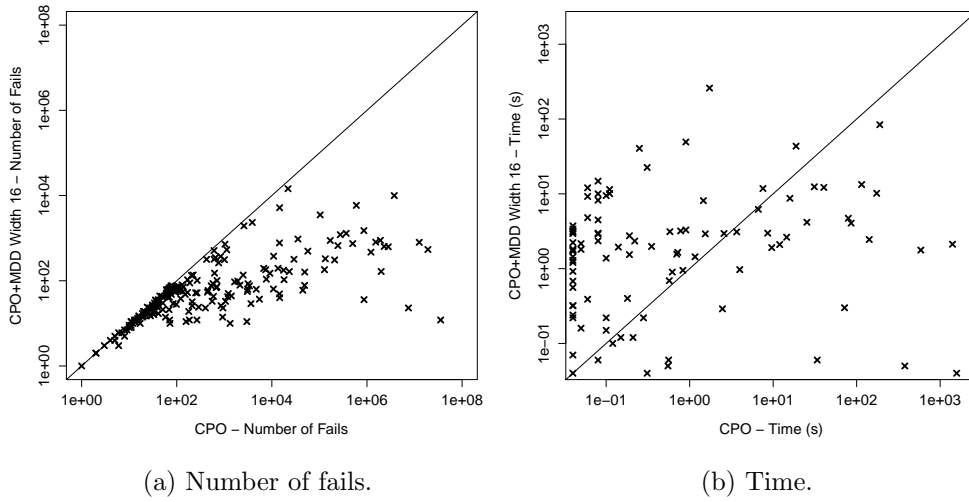


Figure 9 Performance comparison between CPO and CPO+MDD for minimizing makespan on Dumas and Gendreau TSPTW classes. The vertical and horizontal axes are in logarithmic scale.

the CPO results by the CPO+MDD results, and computed the average ratio for each value of β . The job ranking for refinement is done by sorting the jobs in decreasing order according to the value obtained by summing their release dates with their processing times. This forces jobs with larger completion times to have higher priority in the refinement.

The results are presented in Figure 8. For each value of α , we plot the ratio of CPO and CPO+MDD in terms of the number of fails (Figure 8a) and time (Figure 8b). The plot in Figure 8a indicates that the CPO+MDD inference becomes more dominant in comparison to CPO for larger values of β , that is, when setup times become more important. The MDD introduces a computational overhead in comparison to the CPO times (around 20 times slower for this particular problem size). This is compensated as β increases, since the number of fails for the CPO+MDD model becomes orders of magnitude smaller in comparison to CPO. The same behavior was observed on average for other base instances generated under the same scheme.

To evaluate this on structured instances, we consider the TSPTW instances defined by the Gendreau and Dumas benchmark classes, where we changed the objective function to minimize makespan instead of the sum of setup times. We selected all instances with up to 100 jobs, yielding 240 test cases in total. We solved the CPO and the CPO+MDD models with lex search, so as to compare the inference strength for these problems. A maximum width of 16 was set for CPO+MDD and a time limit of 1,800 was imposed to both cases. The job ranking is the same as in the previous experiment.

The CPO approach was able to solve 211 instances to optimality, while the CPO+MDD approach solved 227 instances to optimality (including all the instances solved by CPO). The number of fails

and solving time are presented in Figure 9, where we only depict instances solved by both methods. In general, for easy instances (up to 40 jobs or with a small time window width), the reduction of the number of fails induced by CPO+MDD was not significant, and thus did not compensate the computational overhead introduced by the MDD. However, we note that the MDD presented a better performance for harder instances; the lower diagonal of the Figure 9b is mostly composed by instances from the Gendreau class with larger time windows, for which the number of fails was reduced by five and six orders of magnitude. We also note that the result for the makespan objective is less pronounced than for the sum of setup times presented in Section 10.3.

10.6. Total Tardiness

Constraint-based schedulers are usually equipped with specific filtering techniques for minimizing total tardiness, which are based on the propagation of a piecewise linear function as described by Baptiste et al. (2001). For problems without any constraints, however, the existing schedulers are only capable of solving small instances, and heuristics end up being more appropriate as the propagators are not sufficiently strong to deduce good bounds.

In this section we evaluate the performance of CPO and CPO+MDD on sequencing problems where the objective is to minimize the total tardiness. Since we are interested in evaluating the inference strength of the objective function bounding mechanism, we do not take into account any additional side constraints and we limit our problem size to 15 jobs. Moreover, jobs are only subject to a release date, and no setup time is considered.

We have tested the total tardiness objective using random instances, again generated with the scheme of Chang et al. (1995). The processing times p_i are selected uniformly at random from the set $\{1, 10\}$, the release dates r_i are selected uniformly at random from the set $\{0, \dots, \alpha \sum_i p_i\}$, and the due dates are selected uniformly at random from the set $\{r_i + p_i, \dots, r_i + p_i + \beta \sum_i p_i\}$. To generate a good diversity of instances, we considered $\alpha \in \{0, 0.5, 1.0, 1.5\}$ and $\beta \in \{0.05, 0.25, 0.5\}$. For each random instance generated, we create a new one with the same parameters but where we assign tardiness weights selected uniformly at random from the set $\{1, \dots, 10\}$. We generated 5 instances for each configuration, hence 120 instances in total. A time limit of 1,800 seconds was imposed to all methods. The ranking procedure for refinement is based on sorting the jobs in decreasing order of their due dates.

We compared the CPO and the CPO+MDD models for different maximum widths, and lex search was applied to solve the models. The results for unweighted total tardiness are presented in Figure 10a, and the results for the weighted total tardiness instances are presented in Figure 10b. We observe that, even for relatively small widths, the CPO+MDD approach was more robust than CPO for unweighted total tardiness; more instances were solved in less time even for a width of 16,

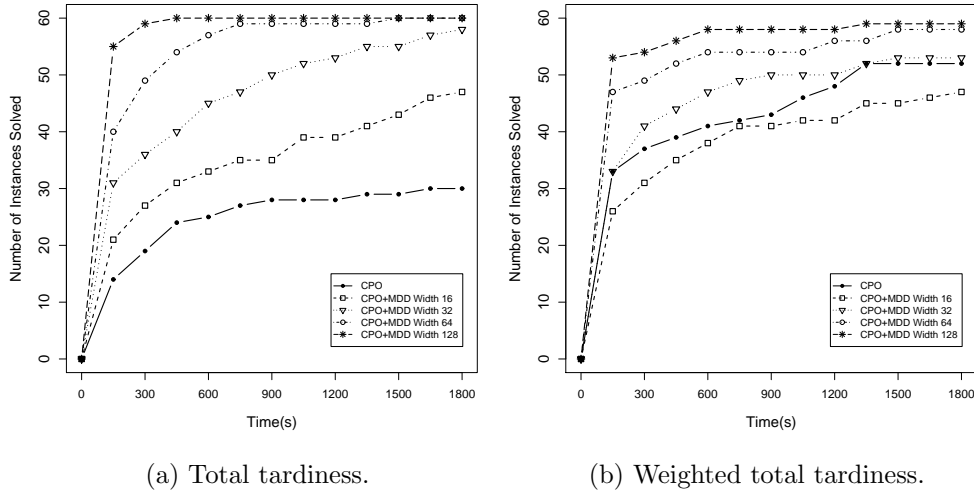


Figure 10 Performance comparison between CPO and CPO+MDD for minimizing total tardiness on randomly generated instances with 15 jobs.

which is a reflection of a great reduction of the number of fails. On the other hand, for weighted total tardiness CPO+MDD required larger maximum widths to provide a more significant benefit with respect to CPO. We believe that this behavior may be due to a weaker refinement for the weighted case, which may require larger widths to capture the set of activities that play a bigger role in the final solution cost.

In all cases, a minimum width of 128 would suffice for the MDD propagation to provide enough inference to solve all the considered problems.

11. Conclusion

We presented a novel generic approach to solving sequencing problems using multivalued decision diagrams (MDDs). We introduced relaxed MDDs to represent an over-approximation of all feasible solutions of a sequencing problem. We showed how these can be used to provide bounds on the objective function value or to derive structured sequencing information, such as precedence relations that must hold in any feasible solution. To strengthen the relaxed MDDs, we proposed a number of techniques for a large class of scheduling problems where precedence and time window constraints are imposed. We also showed that, for a TSP problem introduced by Balas (1999), the MDD that exactly represents all its feasible solutions has a polynomial size in the number of cities. Lastly, we have applied our MDD relaxations to constraint-based scheduling, and we showed that we can improve the performance of a state-of-the-art solver by orders of magnitude. In particular, we were able to close three open TSPLIB instances for the TSP with precedence constraints. Relaxed MDDs thus provide a strong addition to existing generic approaches for solving constrained sequencing problems.

References

- Andersen, H. R., T. Hadzic, J. N. Hooker, P. Tiedemann. 2007. A constraint store based on multivalued decision diagrams. *Proceedings of the 13th international conference on Principles and practice of constraint programming*. CP'07, Springer-Verlag, Berlin, Heidelberg, 118–132.
- Anghinolfi, Davide, Roberto Montemanni, Massimo Paolucci, Luca Maria Gambardella. 2011. A hybrid particle swarm optimization approach for the sequential ordering problem. *Computers & Operations Research* **38**(7) 1076 – 1085.
- Ascheuer, Norbert. 1995. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Ph.D. thesis, Technische Universitt Berlin, Germany.
- Ascheuer, Norbert, Michael Jnger, Gerhard Reinelt. 2000. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications* **17** 2000.
- Balas, E. 1999. New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research* **86** 529–558.
- Balas, Egon, Neil Simonetti. 2000. Linear time dynamic-programming algorithms for new classes of restricted tsp: A computational study. *INFORMS J. on Computing* **13** 56–75.
- Baptiste, Philippe, Claude Le Pape, Wim Nuijten. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. International Series in Operations Research and Management Science, Kluwer.
- Becker, B., M. Behle, F. Eisenbrand, R. Wimmer. 2005. BDDs in a branch and cut framework. S. Nikolettseas, ed., *Experimental and Efficient Algorithms, Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA 05), Lecture Notes in Computer Science*, vol. 3503. Springer, 452–463.
- Behle, M., F. Eisenbrand. 2007. 0/1 vertex and facet enumeration with BDDs. *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*. 158–165.
- Bergman, David, Andre A. Cire, Willem-Jan van Hoeve, John N. Hooker. 2012. Variable ordering for the application of bdds to the maximum independent set problem. *Proceedings of the 9th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. CPAIOR'12, Springer-Verlag, Berlin, Heidelberg, 34–49.
- Bergman, David, Willem-Jan van Hoeve, John Hooker. 2011. Manipulating MDD relaxations for combinatorial optimization. Tobias Achterberg, J. Beck, eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science*, vol. 6697. Springer Berlin / Heidelberg, 20–35.
- Chang, S., Q. Lu, G. Tang, W. Yu. 1995. On decomposition of the total tardiness problem. *Operations Research Letters* **17**(5) 221 – 229.

- Christofides, Nicos, A. Mingozzi, P. Toth. 1981. State-space relaxation procedures for the computation of bounds to routing problems. *Networks* **11**(2) 145–164.
- Cire, Andre A., Willem-Jan van Hoeve. 2012. MDD propagation for disjunctive scheduling. *Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS*. AAAI Press, 11–19.
- Dumas, Yvan, Jacques Desrosiers, Eric Gelinas, Marius M. Solomon. 1995. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research* **43**(2) 367–371.
- Freuder, G., M. Wallace. 2000. Constraint technology and the commercial world. *Intelligent Systems and their Applications, IEEE* **15**(1) 20–23.
- Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman.
- Gendreau, Michel, Alain Hertz, Gilbert Laporte, Mihnea Stan. 1998. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research* **46**(3) 330–335.
- Gouveia, L., P. Pesneau. 2006. On extended formulations for the precedence constrained asymmetric traveling salesman problem. *Networks* **48**(2) 77–89.
- Hadzic, Tarik, John N. Hooker, Barry O’Sullivan, Peter Tiedemann. 2008. Approximate compilation of constraints into multivalued decision diagrams. *Proceedings of the 14th international conference on Principles and Practice of Constraint Programming*. CP ’08, Springer-Verlag, Berlin, 448–462.
- Hernádvölgyi, István T. 2003. Solving the sequential ordering problem with automatically generated lower bounds. *Proceedings of Operations Research 2003*. Springer Verlag, 355–362.
- Hoda, Samid, Willem-Jan Van Hoeve, J. N. Hooker. 2010. A systematic approach to MDD-based constraint programming. *Proceedings of the 16th international conference on Principles and practice of constraint programming*. CP’10, Springer-Verlag, Berlin, Heidelberg, 266–280.
- ILOG. 2012. *CPLEX Optimization Studio V12.4 Manual*.
- Lee, C. Y. 1959. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal* **38** 985–999.
- Lopes, Tony, Andre A. Cire, Cid de Souza, Arnaldo Moura. 2010. A hybrid model for a multiproduct pipeline planning and scheduling problem. *Constraints* **15** 151–189.
- Pinedo, M. 2008. *Scheduling: Theory, Algorithms and Systems*. Third ed. Prentice Hall.
- Rendl, Andrea, Matthias Prandtstetter, Gerhard Hiermann, Jakob Puchinger, Günther Raidl. 2012. Hybrid heuristics for multimodal homecare scheduling. *Proceedings of the 9th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. CPAIOR’12, Springer-Verlag, Berlin, Heidelberg, 339–355.
- TSPLIB. 2012. Retrieved at <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/> on December 10, 2012.

- Vilím, Petr. 2004. $O(n \log n)$ filtering algorithms for unary resource constraint. Jean-Charles Régin, Michel Rueher, eds., *Proceedings of CP-AI-OR 2004, Lecture Notes in Computer Science*, vol. 3011. Springer-Verlag, Nice, France, 335–347.
- Wegener, I. 2000. *Branching programs and binary decision diagrams: theory and applications*. SIAM monographs on discrete mathematics and applications, Society for Industrial and Applied Mathematics.