MDD Propagation for Disjunctive Scheduling

Andre A. Cire, Willem-Jan van Hoeve
Tepper School of Business, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, U.S.A.
{acire,vanhoeve}@andrew.cmu.edu

Abstract

Disjunctive scheduling is the problem of scheduling activities that must not overlap in time. Constraint-based techniques, such as edge finding and not-first/not-last rules, have been a key element in successfully tackling large and complex disjunctive scheduling problems in recent years. In this work we investigate new propagation methods based on limited-width Multivalued Decision Diagrams (MDDs). We present theoretical properties of the MDD encoding and describe filtering and refinement operations that strengthen the relaxation it provides. Furthermore, we provide an efficient way to integrate the MDD-based reasoning with state-of-the-art propagation techniques for scheduling. Experimental results indicate that the MDD propagation can outperform existing domain filters especially when minimizing sequence-dependent setup times, in certain cases by several orders of magnitude.

Introduction

Disjunctive scheduling refers to a wide range of problems in which activities (or *jobs*) must be scheduled in a resource capable of processing only one activity at a time, without interruptions. Activities are usually associated with a number of constraints, such as release times, deadlines, or sequence-dependent setup times. This area has been subject to extensive research and comprises notoriously hard problem classes in both Artificial Intelligence and Operations Research (Pinedo 2008; Brucker 2007).

In this work we study techniques for disjunctive scheduling in the context of *constraint-based scheduling*, which investigates how scheduling problems can be formulated and solved as *Constraint Satisfaction Problems* (CSPs). It is currently regarded as one of the most successful generic techniques for tackling disjunctive scheduling (Baptiste, Le Pape, and Nuijten 2001). Most of its benefits derives from the fact that it enforces a clear separation between the problem definition, which includes the input parameters, constraints, and objective function, from the algorithms and search procedures responsible for solving it. This yields more general-purpose scheduling systems: Each constraint can exploit distinct algorithms and scheduling structures, while the system is flexible to allow for specialized search heuristics better suited to the given problem.

In spite of its relative success being a generic method, constraint-based scheduling still has shortcomings when applied to certain application domains. Most importantly, these include disjunctive scheduling problems with sequence-dependent parameters such as setup costs or transition times, and more complex objective functions that involve, e.g., minimizing the total setup costs. One reason for this negative performance is that the traditional propagation of variable domains (through the *domain store*) is relatively weak in the presence of, e.g., large summations (Baptiste et al. 2006). One of our goals is to strengthen the applicability of constraint-based scheduling to these important problem domains, which include transportation or routing problems (setup costs are associated with travel times) and industrial paintjob problems (setup costs due to changing paint colors).

Recently, *Multivalued Decision Diagrams* (MDDs) were introduced as an alternative to address the weaknesses of the domain store in CSPs (Andersen et al. 2007). MDDs are layered graphical representations of logic functions with a broad application in circuit design, simulation, and software synthesis (Wegener 2000). In this paper, we apply MDDs with a user-specified limited size to encode a *relaxation* of the feasible solution space of a CSP. The MDD is considered as part of the constraint store, also collecting the inferences performed by each constraint in a structured way.

The motivation to complement the domain store with limited-size MDDs is that the first provides a weak relaxation of the problem, simply defined by the Cartesian product of the domains. On the other hand, an MDD represents a more refined relaxation by exploiting non-trivial interactions among variables. Processing a constraint now is not limited to reducing variable domains; its semantic can be utilized to improve the MDD relaxation, namely by removing arcs associated with only infeasible solutions, or by adding nodes that might strengthen the MDD representation. This new framework is denoted by *MDD-based Constraint Programming*, or MDD-based CP (Andersen et al. 2007; Hoda, van Hoeve, and Hooker 2010).

Our contribution in this work is to approach some of the well-known deficiencies of state-of-the-art disjunctive propagators by exploring an MDD-based CP method for the problem. The techniques described here can be either applied to a general-purpose MDD-based solver, or entirely encapsulated in a typical *unary resource* global con-

straint (Baptiste, Le Pape, and Nuijten 2001). Hence, it can be readily implemented in any Constraint Programming solver. Furthermore, we provide a natural way to integrate our MDD-based reasoning with existing domain filters, such as edge-finding and not-first/not-last rules, by using the precedence relations that must hold in any feasible solution as a communication interface between techniques. Experimental results show that this combined approach is indeed very effective, especially when the objective function is associated with sequence-dependent setup times.

Preliminaries and Notation

MDDs Let $C = (X, D, \mathcal{C})$ be a CSP with n variables $X = \{x_1, \ldots, x_n\}$ in any arbitrary order, discrete domains $D(x_i) \in D$ for each $x_i \in X$, and constraints $\mathcal{C}$. A *Multivalued Decision Diagram* (MDD) for C is a directed acyclic multigraph $M = (U, R)$ that encodes the set of feasible solutions of the CSP in a particular form. Namely, the set of nodes U is partitioned into $n+1$ subsets $L_1, \ldots, L_{n+1}$, called *layers*. Layers L_1 and L_{n+1} consist of single nodes; the root $\mathbf{R}$ and the terminal $\mathbf{T}$, respectively. For the purpose of this work, all arcs in R are directed from nodes in layer L_i to nodes in layer L_{i+1}, for some $i = 1, \ldots, n$. Each arc $a = (u, w) \in R$ leaving layer L_k is labeled with a value $\phi_a \in D(x_k)$ and represents the assignment $x_k := \phi_a$. Moreover, no two outgoing arcs of a node are allowed to have the same label. As such, a path $p := (u_1, a_1, u_2, a_2, \ldots, u_{m-1}, a_{m-1}, u_m)$ from a node $u_1 \in L_k$ to a node $u_m \in L_{k'}$ for some $k < k'$ identifies a partial assignment $(x_k, \ldots, x_{k'-1}) = (\phi_{a_1}, \phi_{a_2}, \ldots, \phi_{a_{m-1}})$. (We need to explicitly list the arcs $a_1, \ldots, a_{m-1}$ in p, as there may exist parallel arcs between two nodes u_{i-1}, u_i.) In particular, a path from $\mathbf{R}$ to $\mathbf{T}$ represents a feasible solution of C, and conversely every feasible solution is identified by exactly one path from $\mathbf{R}$ to $\mathbf{T}$ in M.

There may exist more than one possible MDD representing the solutions of C. Let M be one of such MDDs. Two nodes u, v in M are *equivalent* if the partial assignments identified by paths from u to $\mathbf{T}$ and by paths from v to $\mathbf{T}$ are the same. We say that M is *reduced* if no two nodes in any layer are equivalent. There is a unique reduced MDD for a given variable ordering $x_1, \ldots, x_n$, which is the most compact among all MDDs for C with respect to the number of nodes (Andersen et al. 2007).

We now introduce the main MDD notation used throughout the paper. The set of incoming and outgoing arcs at node v of the MDD are given by $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$, respectively. The width w of an MDD is the maximum number of nodes in a layer, given by $w := \max_{1 \le k \le n+1} |L_k|$. In our figures, a set of multiple arcs connecting nodes u and v is depicted as a single arc with an equivalent *arc domain* composed by the union of such labels, for clarity of presentation.

MDD-based Constraint Programming The Constraint Programming framework in which MDDs are used as a constraint store is referred to as *MDD-based Constraint Programming* (Andersen et al. 2007). The underlying idea is to use an MDD with restricted size to encode a relaxation of the feasible solution space of a CSP. More specifically, all feasible solutions of a CSP C (if any) are represented by some path in this limited-size MDD, but not necessarily all paths correspond to solutions that are consistent with the constraint set $\mathcal{C}$. This MDD is then passed from one constraint to the next, which will separately refine the representation by adding nodes or removing arcs.

The strength of the MDD representation can be controlled by increasing the maximum allowed width of the MDD, denoted by the parameter K. For instance, given a CSP C, one can construct an MDD of width $K = 1$ where each layer contains exactly one node and, for $k = 1, \ldots, n$, an arc with label v connects nodes $u \in L_k$ and $v \in L_{k+1}$ for each $v \in D(x_k)$ (see Figure 1a). Note this MDD is equivalent to the domain store, since its paths identify the solutions corresponding to the Cartesian product of the variables. At the other extreme, an unlimited (or large enough) width K allows to exactly represent all solutions to C.

Processing a constraint on an MDD amounts to a *filtering* and a *refinement* operation, which is performed by one constraint at a time. MDD filtering generalizes traditional domain filtering and consists of removing arcs that do not belong to any path that identifies a feasible solution. Refinement consists of adding nodes and arcs to the MDD, so as to strengthen the relaxation it represents without violating the maximum width K. This can be accomplished through an iterative procedure named *incremental refinement* (Hadzic et al. 2008), which is based on *splitting* nodes according to their set of incoming arcs. We note that the filtering and refinement operations can be performed iteratively until a fixpoint is reached. A systematic scheme for MDD propagation is presented in (Hoda, van Hoeve, and Hooker 2010). Here, we specialize those procedures to the case of disjunctive scheduling problems.

We henceforth distinguish between an *exact MDD* and *limited-width MDD* for a CSP C: The first exactly represents all feasible solutions of C, while the second has a limited width of at most K and might contain infeasible solutions.

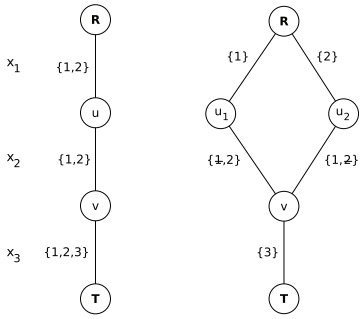
Finally, let $f : X \to \mathbb{R}$ be a separable objective function defined on X. If arc weights are appropriately set in an MDD M, then the shortest path from $\mathbf{R}$ to $\mathbf{T}$ corresponds to the minimum value of f if M is exact, and to a lower bound of f if M is a limited-width MDD (Bergman, van Hoeve, and Hooker 2011).

Example 1 We present an example of MDD processing for the `alldifferent` constraint, as proposed in (Andersen et al. 2007). Consider the CSP defined by $X = \{x_1, x_2, x_3\}$, $D(x_1) = D(x_2) = \{1, 2\}$, $D(x_3) = \{1, 2, 3\}$, and $\mathcal{C}$ composed by a single $\texttt{alldifferent}(x_1, x_2, x_3)$.

The `alldifferent` constraint receives the MDD of Figure 1a as input and carries out filtering and refinement operations. For the filtering, notice first that the set of arc labels occurring in some path from $\mathbf{R}$ to v is $\{1, 2\}$. Since each of these paths corresponds to an assignment of x_1 and x_2, both values will be necessarily taken by these variables, as they must be pairwise distinct. Hence, we can remove arcs with labels $\{1, 2\}$ connecting v and $\mathbf{T}$. Now, suppose our refinement operation splits node u into nodes u_1, u_2, parti-

tioning its incoming arcs as showed in Figure 1b. Since the value 1 belongs to all paths from $\mathbf{R}$ to u_1, it cannot be assigned to any other subsequent variable, so we can remove the arc in $\delta^{\text{out}}(u_1)$ with label 1. We analogously remove the arc labeled with 2 that connects u_2 and v.

The MDD in Figure 1b, after filtering, yields a stronger relaxation than the one represented by its projection onto the variable domains. For instance, suppose we wish to minimize a function $f(x) = x_1 + x_2 + x_3$. The lower bound provided by the MDD corresponds to the shortest path with weights defined by the arc labels, which yields a value of 6. On the other hand, the projection onto the domains is given by $D(x_1) = D(x_2) = \{1, 2\}$, $D(x_3) = \{3\}$, and yields a lower bound value of 5. □



(a) Input (b) After refinement

Figure 1: `alldifferent` processing for Example 1.

Disjunctive Scheduling We consider a constraint-based scheduling system in which the tasks to be scheduled are represented by a finite set of *activities* $\mathcal{A} = \{1, \ldots, n\}$. With each activity $i \in \mathcal{A}$ we associate a processing time p_i, a release time r_i (minimum start time of i), and a deadline d_i (maximum end time of i). Let $s = \{s_1, \ldots, s_n\}$ be a set of variables $s_i \in [r_i, d_i - p_i]$ representing the *start time* of activity $i \in \mathcal{A}$. Furthermore, let $t = \{t_{ij} \geq 0 : i, j \in \mathcal{A}\}$ be a set of *sequence-dependent setup times*. We are interested in developing propagation algorithms for the global constraint

$$\texttt{disjunctive}(s, \mathcal{A}, t), \tag{1}$$

which enforces that activities in $\mathcal{A}$ must not overlap in time and $t_{i,j}$ time units must elapse between the end of activity i and the start time of activity j, when j is the first to succeed i with respect to the other activities. Hence, a feasible assignment of variables s, denoted by *feasible schedule*, implies that activities can be ordered as a *sequence* $\pi = (\pi_1, \ldots, \pi_n)$, where $\pi_i \in \mathcal{A}$ is the i-th activity in the ordering and π satisfies $s_{\pi_{i-1}} + p_{\pi_{i-1}} + t_{\pi_{i-1},\pi_i} \leq s_{\pi_i}$ for $i = 2, \ldots, n$. The `disjunctive` constraint is also referred to as a *unary resource* constraint, since it represents a non-preemptive resource with a capacity of one, and each activity requires one unit of the resource during its execution (Baptiste, Le Pape, and Nuijten 2001). We assume that this constraint is given in conjunction with an objective function with respect to s that must be minimized. Moreover, additional *precedence relations* might also be specified; i.e., if activity i must precede j, which we write $i \ll j$, then $s_i + p_i \leq s_j$ must hold in any feasible solution.

Propagation algorithms for `disjunctive` aim at deducing new valid constraints among activities so as to strengthen the non-overlapping condition. In the traditional domain store, this consists of tightening the domains of the start time variables as much as possible. The most effective propagation techniques for this purpose, such as *edge-finding* and *not-first/not-last rules* (Baptiste, Le Pape, and Nuijten 2001), achieve this domain reduction by relying on a higher dimensional representation of the solution space based on the *precedence relations* between activities. Namely, they are efficient algorithms that deduce all possible precedences between an activity $i \in \mathcal{A}$ and a group of activities $S \subseteq \mathcal{A}$ under particular rules, deriving the tightest time bounds obtainable from these relations (Vilím 2004).

In MDD-based CP, a propagation algorithm for the `disjunctive` must provide a filtering and a refinement operation for a particular encoding of the constraint as an MDD. This MDD is a global structure which may be shared and filtered by other constraints as well. We highlight that this propagation is complementary to the one performed by domain filters; in particular, the MDD approach could be entirely encapsulated in a global constraint and only used to tighten the time variable bounds. Nonetheless, we demonstrate that stronger filtering can be achieved by exploiting interactions between our MDD encoding and the precedence relations derived by existing filters.

MDD Encoding for Disjunctive Scheduling

Our MDD encoding for the `disjunctive` constraint, similar to the description in (Hoda, van Hoeve, and Hooker 2010), is based on a reformulation of (1) into simpler constraints that explicitly represent the sequence of activities in a feasible schedule. Namely, let π_k now be a variable representing the k-th activity to be performed on the schedule, and let e_i be a variable representing the end time of activity $i \in \mathcal{A}$. Constraint (1) can be equivalently written as follows:

$$\texttt{alldifferent}(\pi_1, \ldots, \pi_n), \tag{2}$$

$$e_{\pi_{k-1}} + t_{\pi_{k-1},\pi_k} + p_{\pi_k} \leq e_{\pi_k}, \quad k = 2, \ldots, n, \tag{3}$$

$$D(e_{\pi_k}) = [r_{\pi_k} + p_{\pi_k}, d_{\pi_k}], \quad k = 1, \ldots, n, \tag{4}$$

$$D(\pi_k) = \mathcal{A}, \quad k = 1, \ldots, n. \tag{5}$$

Constraint (2) enforces π to represent a permutation. Constraints (3) state that sequence π must yield non-overlapping time intervals that are consistent with the activity processing times and sequence-dependent setup times. Finally, constraints (4) and (5) describe the domains of variables π and e. In particular, domains in (4) are intentionally written with variables as subscripts. The linking between start and end times variables is done by setting $s_i = e_i - p_i$ for all $i \in \mathcal{A}$, which we assume for ease of notation to be implicit.

A valid MDD representation can be naturally derived from the reformulation above. The layers are defined such that they refer only to variables $\pi_1, \pi_2, \ldots, \pi_n$ in that order; equivalently, layer L_k corresponds to the k-th activity to be scheduled, $k = 1, \ldots, n$. The paths from $\mathbf{R}$ to $\mathbf{T}$ therefore

identify all feasible sequences π that satisfy constraints (2) to (5) if the MDD is exact. We will refer to this encoding as *permutation MDD*, since any path corresponds to a permutation of $\mathcal{A}$. The end time variables are not explicitly represented in the MDD, but rather implied from any particular MDD path under consideration.

Let M be a limited-width permutation MDD for an instance of the `disjunctive` constraint. We define a *state* for each node of M, which will explore the structure of the permutation MDD for the purpose of filtering and refinement. With each node $v \in L_k$, associate a tuple

$$I_v := (A_v^{\downarrow}, S_v^{\downarrow}, E_v), \tag{6}$$

where each element is described as follows. The states $A_v^{\downarrow}$ and $S_v^{\downarrow}$ are the set of activities that belong to *all* and *some* paths from $\mathbf{R}$ to v, respectively. They stem from the `alldifferent` constraint (2) following (Andersen et al. 2007). The state E_v is primarily derived from constraints (3) and (4). It represents the *minimum end time* at v; more specifically, the minimum end time over all partial sequences that are identified by paths from $\mathbf{R}$ to v.

The state (6) can be computed for all nodes by a single top-down pass in M. Namely, let $(A_{\mathbf{R}}^{\downarrow}, S_{\mathbf{R}}^{\downarrow}, E_{\mathbf{R}}) = (\emptyset, \emptyset, 0)$ and assume the states for all nodes in layers $L_2, \ldots, L_k$ were already calculated. Suppose we wish to compute I_v for a node $v \in L_{k+1}$. The `alldifferent` states are obtained by applying the following rules from (Andersen et al. 2007) (recall that the arc labels $\phi_{(u,v)}$ refer to activities in $\mathcal{A}$):

$$A_v^{\downarrow} := \bigcap \{A_u^{\downarrow} \cup \{\phi_{(u,v)}\} : (u,v) \in \delta^{\text{in}}(v)\}, \tag{7}$$

$$S_v^{\downarrow} := \bigcup \{S_u^{\downarrow} \cup \{\phi_{(u,v)}\} : (u,v) \in \delta^{\text{in}}(v)\}. \tag{8}$$

Lastly, E_v can be computed using the following Lemma:

Lemma 1 *For a node $v \in U$,*

$$E_v = \min \left\{ E'_{(u,v)} : (u,v) \in \delta^{\text{in}}(v) \right\} \tag{9}$$

where E'_a for an arc $a = (u,v)$ is given by

$$E'_a = \max \left\{ r_{\phi_a}, E_u + \min_{b \in \delta^{\text{in}}(u)} \{t_{\phi_b, \phi_a}\} \right\} + p_{\phi_a}. \tag{10}$$

Proof. Let p be any path from $\mathbf{R}$ to a node v in the MDD. We define $\mathit{eet}^{(p)}(v)$ as the earliest end time at v when processing the activities in the sequence represented by p, while respecting constraints (2) to (5). Likewise, we define $\mathit{let}^{(p)}(v)$ as the latest end time at v following the sequence given by p. Thus, for any complete path $p = (v_1, a_1, v_2, a_2, \ldots, v_n, a_n, v_{n+1})$ where $v_1 = \mathbf{R}$ and $v_{n+1} = \mathbf{T}$, we can retrieve consistent end (and start) times by appropriately setting $e_i \in [\mathit{eet}^{(p)}(v_k), \mathit{let}^{(p)}(v_k)]$, such that $\phi_{a_{k-1}} = i$, for $k = 2, \ldots, n+1$.

Assume wlog that $v \in L_{k+1}$ for some k. By definition,

$$E_v := \min\{\mathit{eet}^{(p)}(v) : p \in \mathcal{P}_v^{\mathbf{R}}\} \tag{11}$$

where $\mathcal{P}_v^{\mathbf{R}}$ is the set of paths from $\mathbf{R}$ to v. To compute E_v, note that inequalities (3) and (4) impose

$$e_{\pi_k} \geq e_{\pi_{k-1}} + t_{\pi_{k-1}, \pi_k} + p_{\pi_k}, \tag{12}$$

$$e_{\pi_k} \geq r_{\pi_k} + p_{\pi_k} \tag{13}$$

which implies, for any path $p = (\mathbf{R}, a_1, \ldots, u_k, a_k, v)$ representing the partial assignment $\pi_i^* = \phi_{a_i}$ $(i = 1, \ldots, k)$,

$$\mathit{eet}^{(p)}(v) = \max \left\{ r_{\pi_k^*},\ \mathit{eet}^{(p)}(u_k) + t_{\pi_{k-1}^*, \pi_k^*} \right\} + p_{\pi_k^*}. \tag{14}$$

The equality 9 now follows from a dynamic programming argument using the relations (11) and (14). ■

activity i	r_i	d_i	p_i
activity 1	3	15	4
activity 2	5	12	3
activity 3	0	6	2

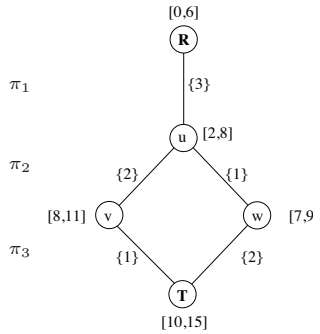


Figure 2: Reduced MDD for Example 2. Next to each node v, the interval $[E_v, F_v]$ is given.

Example 2 Let $\mathcal{A} = \{1,2,3\}$, $(r_1, d_1, p_1) = (3, 15, 4)$, $(r_2, d_2, p_2) = (5, 12, 3)$, and $(r_3, d_3, p_3) = (0, 6, 2)$. Suppose $t_{i,j} = 0$ for all $i, j \in \{1,2,3\}$. The reduced permutation MDD is depicted in Figure 2. Next to each node v we depict the interval $[E_v, F_v]$, reflecting the earliest, respectively latest, ending time at v (F_v is formally introduced in Section 'Filtering'). For each $\mathbf{R}$-$\mathbf{T}$ path, we can determine the earliest and latest ending times for the activities. In particular, path $p = (\mathbf{R}, u, w, \mathbf{T})$ yields $e_1 \in [7,9]$, $e_2 \in [10,12]$, and $e_3 \in [2,6]$. □

Refinement of the Permutation MDD

In this section we develop an incremental refinement technique for the limited-width permutation MDD. We remark that the refinement operation is a fundamental component of the MDD-Based CP framework, since it may help filters to remove a significant part of the infeasible solution space and dramatically improve the propagation effectiveness.

First, it is in general desirable that the resulting new nodes are non-equivalent, which prevents unnecessary recomputations and yields more fine-grained MDDs. We first assess the complexity of identifying this condition with respect to the permutation MDD, which is formalized in Theorem 2.

Theorem 2 *Let M be an MDD representing an arbitrary* `disjunctive` *instance. Deciding if two nodes u, v are equivalent is NP-Hard.*

Proof. Consider the n partial assignments composed of a single activity that are identified by paths with a single arc $(\mathbf{R}, v)$, $v \in \delta^{\text{out}}(\mathbf{R})$: $(1), (2), \ldots, (n)$. Two partial assignments (i) and (j) with $i \neq j$ have the same completions only if their set of completions is the empty set. Hence, we

can decide if the problem is infeasible by performing $\mathcal{O}(n^2)$ node equivalence tests, one for each pair of the n assignments. But this is the same as deciding if a single machine problem with arbitrary release times and deadlines has a solution, which is NP-Hard (Garey and Johnson 1979). ■

In spite of the result in Theorem 2, we can still provide necessary node equivalence conditions by taking the conjunction of individual tests for each problem constraint (Hadzic et al. 2008). We provide one of such necessary conditions in Lemma 3.

Lemma 3 *Let M be a limited-width MDD representing an arbitrary* `disjunctive` *instance. For a node u and an activity $i \in \mathcal{A}$, let $T_u(i)$ be the maximum time activity i could start according to the partial sequences identified by paths from $\mathbf{R}$ to u, defined by $T_u(i) := \max_{(w,u)\in\delta^{\text{in}}(u)} \{d_{\phi_{(w,u)}} + t_{\phi_{(w,u)},i}\}$. Given two nodes $u, v \in L_k$, the conditions*

$$S_v^{\downarrow} = S_u^{\downarrow} \ \wedge \ |S_v^{\downarrow}| = k-1, \tag{15}$$

$$r_i \geq \max\{T_u(i), T_v(i)\}, \ \ \forall i \in \mathcal{A} \setminus A_v^{\downarrow} \tag{16}$$

imply that u and v are equivalent.

Proof. The proof follows from contradiction. Suppose, without loss of generality, that there exists a completion $(\pi_k, \ldots, \pi_n)$ such that the sequence $(\pi_1^u, \ldots, \pi_{k-1}^u, \pi_k, \ldots, \pi_n)$ is feasible for some partial sequence $(\pi_1^u, \ldots, \pi_{k-1}^u)$ identified by a path from $\mathbf{R}$ to u, but the sequence $(\pi_1^v, \ldots, \pi_{k-1}^v, \pi_k, \ldots, \pi_n)$ is infeasible for some $(\pi_1^v, \ldots, \pi_{k-1}^v)$ identified by a path from $\mathbf{R}$ to v. We will see in the filtering section that the conditions (15) will force the filters to remove all arcs with labels in $S_v^{\downarrow}$, since they represent the fact that the first $k-1$ variables are taking the corresponding $k-1$ values in $S_v^{\downarrow}$. Thus, the `alldifferent` is not the reason for infeasibility, and therefore the deadline of some activity $\pi_k, \ldots, \pi_n$ is violated in the completion for v. Since $\{\pi_{k-1}^u, \pi_{k-1}^v\} \subseteq \{\phi_{(w,t)} : (w,t) \in \delta^{\text{in}}(u) \cup \delta^{\text{in}}(v)\}$, condition (16) implies $r_{\pi_k} \geq \max\{d_{\pi_{k-1}^u} + t_{\pi_{k-1}^u,\pi_k},\ d_{\pi_{k-1}^v} + t_{\pi_{k-1}^v,\pi_k}\}$ and hence the minimum start time of activity π_k in both sequences is the same, which is a contradiction. ■

A number of actively studied problems in scheduling theory may only consider deadlines in the objective function, not as a constraint (Pan and Shi 2008; Pinedo 2008). In this case Lemma 4 can be directly applied.

Lemma 4 *Suppose the* `disjunctive` *instance M represents is such that $d_i = +\infty$ for all $i \in \mathcal{A}$. Two nodes $u, v \in L_k$ for $j = 2, \ldots, n$ are equivalent if and only if conditions (15) and (16) are satisfied.*

Proof. Any sequence corresponding to a permutation of $\mathcal{A}$ is feasible, and the conditions above are necessary and sufficient for the `alldifferent` constraint (2). ■

The node equivalence conditions presented so far are not expected to be frequently satisfied, since exact MDDs representing permutations have exponentially large widths in general. Hence, the decision on how to choose and split nodes should be mainly heuristic, in a way that explores the particular structure of the constraints.

Based on this assumption, we now present a refinement technique for the permutation MDD. It builds on a previous method called *incremental refinement* (Hadzic et al. 2008), and can be easily generalized to other constraints. The procedure is outlined in Algorithm 1 and consists of two phases.

In the *node processing phase*, we first compute the state of the nodes that would be created if we were to fully split a node $v \in L_k$ into nodes $v_1, \ldots, v_{|\delta^{\text{in}}(v)|}$ containing a single incoming arc. This corresponds to one 'temporary' state for each arc $a = (u, v) \in \delta^{\text{in}}(v)$, denoted by $I'_a := (A_a^{\downarrow\prime}, S_a^{\downarrow\prime}, E'_a)$, where E'_a has the same meaning as in Lemma 1. The arc a and its respective state I'_a are then stored in a data structure $\mathcal{R}$. We also apply our filtering rules, which means we do not add infeasible arcs to $\mathcal{R}$. The node processing phase is depicted in steps 4 to 15 in Algorithm 1.

In the *refinement phase*, we partition the set $\mathcal{R}$ into at most K groups according to some heuristic criteria. These groups will represent the final nodes in the refined layer. Namely, for each group V_i, a new node v will be created. The incoming arcs at v correspond to the arcs in V_i, and the outgoing arcs of v correspond to the outgoing arcs of all the target nodes in V_i. This is represented in steps 17 to 24 in Algorithm 1. We replace a layer L_k by a new layer composed by these new nodes, which represents the refinement.

The partitions should be defined by taking the state $I'_{u,v}$ into account, for instance using particular heuristic criteria. The set $\mathcal{R}$ gives a *global* view of the layer in that it allows us to infer the state of the resulting new nodes before actually performing the splits. That is, given any subset $V \subseteq \mathcal{R}$, the new node v to be created from V is such that $E_v = \min_{(u,w)\in V}\{E'_{(u,w)}\}$, $A_v^{\downarrow} = \bigcap_{(u,w)\in V}\{A_{(u,w)}^{\downarrow}\}$, and $S_v^{\downarrow} = \bigcup_{(u,w)\in V}\{S_{(u,w)}^{\downarrow}\}$, which follows from the definition of the states. We note that special care must be taken to avoid creating partitions that would force a node in a previous layer to have two outgoing arcs with the same label.

The heuristic criterion used to partition $\mathcal{R}$ in this work ensures that the $K-1$ new nodes are associated with the lowest $K-1$ values possible for E_v, breaking ties according to the size of the $A_v^{\downarrow}$ set (i.e,. larger first). This was verified to be very effective in practice. Intuitively, smaller values for E_v force paths with small makespan to be present in the MDD, while larger $A_v^{\downarrow}$ countermeasures this procedure by enforcing that sequences must be pairwise distinct.

Filtering

This section presents our filtering rules that can be applied to a permutation MDD M. We first extend the state definition to also consider a bottom-up perspective of M. Namely, for each node v in M, we now redefine I_v as

$$I_v := (A_v^{\downarrow}, S_v^{\downarrow}, E_v, A_v^{\uparrow}, S_v^{\uparrow}, F_v). \tag{17}$$

The state elements $A_v^{\downarrow}$, $S_v^{\downarrow}$, and E_v are the same as defined in (6). The states $A_v^{\uparrow}$ and $S_v^{\uparrow}$ are bottom-up versions of the first two, and can be analogously computed within a bottom-

Algorithm 1: MDD Refinement (Input: MDD M)

```
1  begin
2      for layer indices k = 2, ..., n do
3          // 1. Node processing phase
4          R := ∅
5          foreach node v ∈ L_k do
6              foreach arc (u, v) ∈ δ^in(v) do
7                  A'↓_(u,v) := A↓_u ∩ {φ_(u,v)}
8                  S'↓_(u,v) := S↓_u ∪ {φ_(u,v)}
9                  Compute E'_(u,v) according to (10)
10                 // filtering
11                 if a rule (19)-(22) is violated then
12                     Remove arc (u, v)
13                 else
14                     I'_{u,v} := (A'↓_(u,v), S'↓_(u,v), E'_(u,v))
15                     R := R ∪ {(I'_(u,v), (u, v))}
16         // 2. Refinement phase
17         Partition R into (at most) V_1, ..., V_K groups
           according to E'_{u,v}
18         L'_k := ∅
19         foreach group V_i do
20             Create a new node v, add it to L'_k
21             δ+(v) := {(u, v) : ∃ (u, w) ∈ V_i}
22             δ−(v) := ⋃{δ^out(w) : ∃(u, w) ∈ V_i}
23             Compute I_v
24         Replace L_k by L'_k
25 end
```

up pass by setting $A^{\uparrow}_{\mathbf{T}} = S^{\uparrow}_{\mathbf{T}} = \emptyset$ and for any $v \in L_k$, $k \geq 2$,

$$A^{\uparrow}_v := \bigcap \{A^{\uparrow}_u \cup \{\phi_{(v,u)}\} : (v,u) \in \delta^{\text{out}}(v)\},$$
$$S^{\uparrow}_v := \bigcup \{S^{\uparrow}_u \cup \{\phi_{(v,u)}\} : (v,u) \in \delta^{\text{out}}(v)\},$$

which stem from (7) and (8), respectively (Andersen et al. 2007). The state F_v represents the *maximum end time* at $v \in L_k$; more specifically, the maximum end time over all partial sequences identified by paths from $\mathbf{R}$ to v. It can be computed analogous to Lemma 1, as formalized in the following Lemma (presented here without proof):

Lemma 5 *For a node $v \in U$,*

$$F_v = \min \left\{ \max_{a \in \delta^{\text{in}}(v)} d_{\phi_a}, \ \max_{a \in \delta^{\text{out}}(v)} \{F'_a\} \right\}, \qquad (18)$$

where F'_a for an arc $a = (v, w)$ is given by

$$F'_a := F_w - \min_{b \in \delta^{\text{in}}(v)} \{t_{\phi_b, \phi_a}\} - p_{\phi_a}.$$

The first term of the min function in (18) as well as $\min_{b \in \delta^{\text{in}}(v)} \{t_{\phi_b, \phi_a}\}$ are computed during a top-down pass, which can be already used as an upper bound of F_v for filtering purposes. The remaining terms are computed during a bottom-up pass.

The filtering rules used to remove inconsistent arcs are now described. They can be applied during both top-down and bottom-up passes, or simultaneously with refinement (step 7 in Algorithm 1). Consider nodes u, v with $u \in L_k$ and $v \in L_{k+1}$. From the `alldifferent` constraint (2), arc (u, v) can be removed if any of the conditions below hold (Hoda, van Hoeve, and Hooker 2010):

$$\phi_{(u,v)} \in A^{\downarrow}_u \cup A^{\uparrow}_v, \qquad (19)$$
$$|S^{\downarrow}_u| = k - 1 \ \ \wedge \ \ \phi_{(u,v)} \in S^{\downarrow}_u, \qquad (20)$$
$$|S^{\uparrow}_v| = n - k + 1 \ \ \wedge \ \ \phi_{(u,v)} \in S^{\uparrow}_v. \qquad (21)$$

Observe now that E_v and F_v represent a lower and upper bound, respectively, of variable $e_{\pi_{k-1}}$ in the reformulation (2)-(5) of the `disjunctive` constraint. Hence, if we fix $\pi_{k-1} = \phi_{(u,v)}$ for an arc (u, v), we need only to check if this violates the maximum end time at v. This is summarized in the following rule, in which an arc (u, v) for $u \in L_k$ and $v \in L_{k+1}$ is removed if

$$\max \left\{ r_{\phi_{(u,v)}}, \ E_u + \min_{(w,u) \in \delta^{\text{in}}(u)} \{t_{\phi_{(w,u)}, \phi_{(u,v)}}\} \right\} + p_{\phi_{(u,v)}} > \min \left\{ d_{\phi_{(u,v)}}, F_v \right\}. \qquad (22)$$

As a further note, state values are monotonically increasing or decreasing. Hence, they can be preserved along the refinement or after some search decision that does not necessarily remove the node, possibly yielding extra propagation since a valid state, even if relaxed, is readily available without an additional top-down or bottom-up pass.

Optimization A limited-width MDD can provide a lower bound for any separable scheduling objective function by assigning appropriate arc weights and computing the shortest path from $\mathbf{R}$ to $\mathbf{T}$. For instance, to minimize the sum of the setup times, we assign a weight $w_{(u,v)} = \min_{a \in \delta^{\text{in}}(u)} t_{a, \phi_{(u,v)}}$ to each arc (u, v). To minimize makespan, the lower bound is directly given by $E_{\mathbf{T}}$.

The value v^* of the shortest path can be used in different ways for filtering purposes. If a *cost* variable is associated with the `disjunctive`, we update its domain according to v^*. Moreover, if the shortest path from $\mathbf{R}$ to a node is above the upper bound of this cost variable, we may remove the corresponding arcs from the MDD as well. Note that, by doing so, Algorithm 1 remains valid — except for the equivalence tests — and it can be improved by taking the objective function into account for its heuristic component.

Integration with Domain Filters

The inferences performed by state-of-the-art scheduling domain filters and the permutation MDD can be shared so as to increase the effectiveness of both propagation techniques. The key insight is that existing filters, as discussed previously, tighten variable domains by deducing *precedence relations* among activities according to specific rules. We show in this section that there is an intrinsic connection between the permutation MDD and the precedence relations of a `disjunctive` instance. Namely, given a limited-width permutation MDD M, we can efficiently deduce all precedence relations that are satisfied by the solutions encoded in M. Conversely, given a set of precedence relations that

must hold in any feasible sequence, we can apply additional filtering rules to M to further strengthen the relaxation it provides. Hence, precedence relations can be used as a communication interface between traditional scheduling propagators and the MDD-based filters.

To formally state our result, we assume that all considered MDDs henceforth associate a state I_v, as defined in (17), with each of its nodes v. We have the following Theorem.

Theorem 6 *Let M be an exact permutation MDD for an arbitrary* `disjunctive` *instance. An activity i must precede activity j in any feasible solution if and only if*

$$(j \notin A_u^{\downarrow}) \text{ or } (i \notin A_u^{\uparrow})$$

for all nodes u in M.

Proof. Suppose there exists a node u in layer L_k, $k \in \{1, \ldots, n+1\}$, such that $j \in A_u^{\downarrow}$ and $i \in A_u^{\uparrow}$. By definition, there exists a path $(\mathbf{R}, \ldots, u, \ldots, \mathbf{T})$ that identifies a sequence where activity j starts before activity i. This can only be true if and only if activity i may not precede j in some feasible solution. ■

Corollary 7 *The set of all precedence relations that must hold in a* `disjunctive` *instance can be efficiently extracted from its exact MDD $M = (U, R)$ in $\mathcal{O}(n^2|U|)$.*

Proof. Construct a digraph $G^* = (\mathcal{A}, E^*)$ by adding an arc (i, j) to E^* if and only if there exists a node u in M such that $j \in A_u^{\downarrow}$ and $i \in A_u^{\uparrow}$. Checking this condition for all pairs of activities takes $O(n^2)$ for each node in M, and hence the time complexity to construct G^* is $O(n^2|U|)$. According to Theorem 6 and the definition of G^*, the complement graph of G^* contains an edge (i, j) if and only if $i \ll j$. ■

As we are mainly interested in limited-width MDDs, we derive an additional Corollary of Theorem 6.

Corollary 8 *Given a limited-width MDD M for a* `disjunctive` *instance, an activity i must precede activity j in any feasible solution (in case one exists) if*

$$(j \notin S_u^{\downarrow}) \text{ or } (i \notin S_u^{\uparrow})$$

for all nodes u in M.

Proof. It follows from the state definitions that $A_u^{\downarrow} \subseteq S_u^{\downarrow}$ and $A_u^{\uparrow} \subseteq S_u^{\uparrow}$. Hence, if the conditions for the relation $i \ll j$ from Theorem 6 are satisfied by $S_u^{\downarrow}$ and $S_v^{\uparrow}$, they must be also satisfied by a reduced MDD containing only the feasible solutions of the instance. ■

By Corollary 8, the precedence relations implied by the solutions of a limited-width MDD M can be extracted by applying the algorithm in Corollary 7 to the states $S_v^{\downarrow}$ and $S_v^{\uparrow}$. These precedences can then be used to tighten start time variables, as shown in Example 3 below, or be provided directly to constraint-based solvers that may benefit from them. Since M has at most $\mathcal{O}(nK)$ nodes, this algorithm has a worst-case complexity of $\mathcal{O}(n^3K)$.

Conversely, suppose we collect the set of precedences deduced by domain filters in a set $\mathcal{P} \in \mathcal{A} \times \mathcal{A}$, such that $(i, j) \in \mathcal{P}$ if $i \ll j$. Then, Theorem 6 and the state definitions immediately yield the following filtering rules. Arc (u, v) with $u \in L_k$, $v \in L_{k+1}$ can be removed if any of the conditions below holds, where $i \in \mathcal{A}$ is an activity.

$$i \in A_u^{\downarrow}, \qquad \exists\, (\phi_{(u,v)}, i) \in \mathcal{P}, \quad (23)$$

$$|S_u^{\downarrow}| = k - 1 \;\wedge\; i \in S_u^{\downarrow}, \qquad \exists\, (\phi_{(u,v)}, i) \in \mathcal{P}, \quad (24)$$

$$i \in A_v^{\uparrow}, \qquad \exists\, (i, \phi_{(u,v)}) \in \mathcal{P}, \quad (25)$$

$$|S_v^{\uparrow}| = n - k + 1 \;\wedge\; i \in S_v^{\uparrow}, \quad \exists\, (i, \phi_{(u,v)}) \in \mathcal{P}. \quad (26)$$

These conditions can be efficiently checked during a top-down and a bottom-up pass, or in step 7 in Algorithm 1.

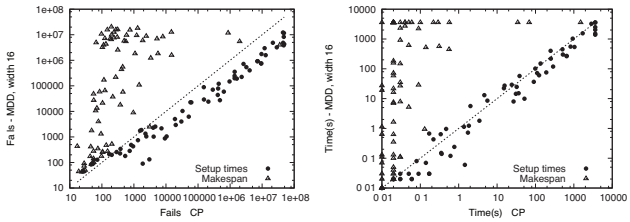
Example 3 The propagation of the precedence relations inferred by the MDD can lead to a stronger domain filtering than edge-finding and not-first/not-last rules, even for smaller widths. Consider the following instance from Vilím (2004): $\mathcal{A} = \{1, 2, 3\}$, $(r_1, d_1, p_1) = (0, 25, 11)$, $(r_2, d_2, p_2) = (1, 27, 10)$, and $(r_3, d_3, p_3) = (14, 35, 5)$, with zero setup time for any pair of activities.

Edge-finding and not-first/not-last rules deduce $1 \ll 3$ and $2 \ll 3$, which does not suffice to change the start time bounds of any variables $i \in \mathcal{A}$. However, starting with an MDD of width 1 and applying our filters and refinement operations, we obtain the same MDD presented in Figure 1b (where $x_i = \pi_i$, for $i = 1, \ldots, 3$). If the precedences are kept in a set $\Omega_i := \{j : j \in \mathcal{A}, j \ll i\}$ for each $i \in \mathcal{A}$, the filtering rules will set $\Omega_3 := \{1, 2\}$. This triggers the propagation $s_3 \geq 10 + 11 = 21$, which can be used to update the start time variable of activity 3. □

Computational Experiments

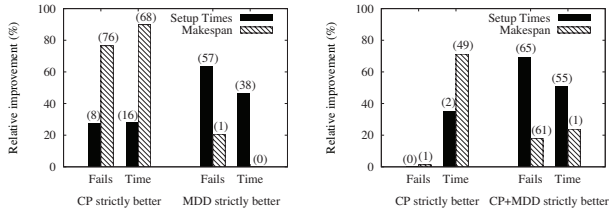
We have implemented our MDD propagation algorithm as a global constraint in IBM-ILOG CP Optimizer that is packaged within CPLEX Academic Studio 12.4. We evaluate our methods on problem instances arising from the Traveling Salesman Problem with Time Windows, which can be modelled by a single `disjunctive`. Even though specialized CP algorithms have been previously applied to this particular problem (Pesant et al. 1996; Focacci, Lodi, and Milano 2002), existing domain filters are known to be less effective when setup times are involved. We consider two different objective functions: one minimizes makespan, while the other minimizes the sum of setup times. The makespan is a traditional objective function studied in scheduling, while the sum of setup times represents an important component of many real-world scheduling problems, as discussed before.

We compared three techniques: a formulation with the unary resource constraint from CP Optimizer (`CP`), *IloNoOverlap*, set with *extended* filtering; the standalone MDD filtering (`MDD`); and a combined approach (`CP+MDD`). However, a relatively loose integration is considered for `CP+MDD`, since the precedences inferred by CP Optimizer are only partially available. Specifically, the precedences inferred by the MDD propagation are communicated to CP Optimizer by restricting a variable *IlcIntervalSequenceVar*, which specifies the activities that may be next in the sequence. Conversely, the domain of this variable is used to prune arcs in the first and previous to the last layer of the



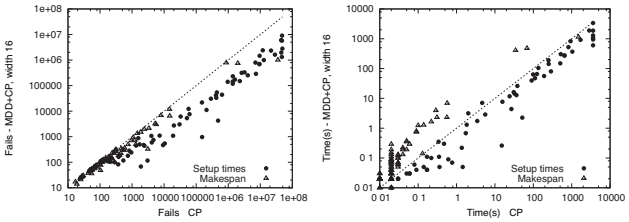
(a) Fails (b) Time

Figure 3: Performance between CP and MDD



(a) CP vs. MDD (b) CP vs. CP+MDD

Figure 4: Relative performance of CP versus MDD (a), and CP versus CP+MDD (b), for minimizing setup times and makespan. The number of instances that were used to produce each bar is indicated between parentheses.



(a) Fails (b) Time

Figure 5: Performance between CP and CP+MDD.

MDD. Tests were implemented in C++ and ran on an Intel Xeon E5345 with 8 GB RAM. For exposition convenience, we set our maximum MDD width to 16, which was chosen based on manual parameter tuning. Generally, however, this value can be dynamically determined based on the problem size or other parameters (e.g., memory).

We selected 85 instances from the Dumas (Dumas et al. 1995) and AFG (Ascheuer 1995) benchmarks for which at least one of the methods would not exceed the time limit. These instances contain between 10 and 60 activities, and have a wide range of time windows, i.e., $[d_i - r_i]$ for activity i. In order to make a proper qualitative comparison (in terms of filtering and propagation strength) between the three methods CP, MDD, and CP+MDD, all use the same lexicographic search, in which activities were recursively assigned to be first in the schedule according to a previously fixed order. We impose a time limit of 3600 seconds. All reported averages correspond to geometric means.

CP vs. MDD We first compare CP to the pure MDD approach. Figure 3 shows a scatter plot for the number of fails (the number of dead ends encountered during the backtrack search), and the solving time in seconds. For these instances, the average relative improvement (as a percentage) of CP and MDD is presented in Figure 4.a. Here, the relative improvement between values a and b is computed as $(b-a)/b$ for those data cases where $a < b - 0.01$, i.e., the method for a was strictly better than the method for b. The figure also shows the number of instances for which one method was better than the other between parentheses for each class. Together, these figures indicate that the pure MDD approach can substantially improve CP Optimizer when the objective is to minimize the sum of the setup times. For example, regarding the number of fails, MDD was better than CP for 65 instances (these are all instances that were solved), with an average of 69% less fails than CP. It is also important to note that the CP performs much better than MDD when minimizing the makespan. Most likely, this behavior is due to the very well-developed filtering techniques for makespan that are not being explored by the MDD.

CP vs. CP+MDD We next compare CP and the combined approach CP+MDD. This is perhaps the most relevant analysis, since CP solvers usually run a portfolio of filtering techniques, to which the MDD-based approach could be added. The results are shown in Figure 5 as a scatter plot, while the average relative performance is shown in Figure 4.b. These figures indicate that the combined approach yields more robust results, often outperforming both CP and MDD alone. For some instances, we observed orders of magnitude improvement; e.g., n60w20.003 from Dumas, with 60 activities and average time window width of 20, was solved in 30 seconds by CP+MDD, but could not be solved within 1 hour by CP. Again, the improvement of the combined approach manifests itself mainly when minimizing setup times, while CP remains the better method (in terms of time) for minimizing makespan.

Dynamic Search As a last comment, we also tested all three methods with a dynamic search strategy, where activities are sequentially added to the schedule according to their propagated earliest start time. We found that the results were almost unchanged, except that the combined approach CP+MDD could now solve three more instances while the total solved instances remained the same for CP and MDD.

Conclusion

In this paper we studied propagation techniques based on MDDs for disjunctive problems. We presented refinement and filtering operations to strengthen the relaxation it represents, and showed how it can be integrated with existing filters such as edge-finding. Experimental results demonstrated that MDD propagation can yield stronger filtering that existing domain propagation methods, especially when minimizing sequence-dependent setup times.

Acknowledgements This work was supported by NSF under grant CMMI-1130012 and a Google Research Grant.

References

Andersen, H. R.; Hadzic, T.; Hooker, J. N.; and Tiedemann, P. 2007. A constraint store based on multivalued decision diagrams. In *Proceedings of the 13th international conference on Principles and practice of constraint programming*, CP'07, 118–132. Berlin, Heidelberg: Springer-Verlag.

Ascheuer, N. 1995. *Hamiltonian path problems in the online optimization of flexible manufacturing systems*. Ph.D. Dissertation, Technische Universität Berlin.

Baptiste, P.; Laborie, P.; Le Pape, C.; and Nuijten, W. 2006. *Constraint-Based Scheduling and Planning*. Elsevier. chapter 22, 761–799.

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. International Series in Operations Research and Management Science. Kluwer.

Bergman, D.; van Hoeve, W.-J.; and Hooker, J. 2011. Manipulating MDD relaxations for combinatorial optimization. In Achterberg, T., and Beck, J., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6697 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 20–35.

Brucker, P. 2007. *Scheduling algorithms*. Springer Verlag.

Dumas, Y.; Desrosiers, J.; Gelinas, E.; and Solomon, M. 1995. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research* 367–371.

Focacci, F.; Lodi, A.; and Milano, M. 2002. A hybrid exact algorithm for the tsptw. *INFORMS Journal on Computing* 14(4):403–417.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

Hadzic, T.; Hooker, J. N.; O'Sullivan, B.; and Tiedemann, P. 2008. Approximate compilation of constraints into multivalued decision diagrams. In *Proceedings of the 14th international conference on Principles and Practice of Constraint Programming*, CP '08, 448–462. Berlin, Heidelberg: Springer-Verlag.

Hoda, S.; van Hoeve, W.-J.; and Hooker, J. N. 2010. A systematic approach to MDD-based constraint programming. In *Proceedings of the 16th international conference on Principles and practice of constraint programming*, CP'10, 266–280. Berlin, Heidelberg: Springer-Verlag.

Pan, Y., and Shi, L. 2008. New hybrid optimization algorithms for machine scheduling problems. *IEEE Transactions on Automation Science and Engineering* 5(2):337–348.

Pesant, G.; Gendreau, M.; yves Potvin, J.; and Rousseau, J.-M. 1996. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science* 32:12–29.

Pinedo, M. 2008. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, third edition.

Vilím, P. 2004. $O(n \log n)$ filtering algorithms for unary resource constraint. In Régin, J.-C., and Rueher, M., eds., *Proceedings of CP-AI-OR 2004*, volume 3011 of *Lecture Notes in Computer Science*, 335–347. Nice, France: Springer-Verlag.

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.