MDD Propagation for Disjunctive Scheduling

Tracking Number: 7

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 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, there are a number of problems for which constraint-based scheduling still suffers many shortcomings. This is particularly true in the presence of parameters that depend on how the activities are ordered in a schedule, such as in sequence-dependent setup times and in routing problems. A similar issue occurs for certain objective functions; it is not rare to find instances with less than 20 activities that require hours to be solved to optimality by state-of-the-art CSP solvers. In any case, the reason for this negative performance usually rests on the fact that the traditional *domain store*, in which variable domains are used to transmit the inferences from one constraint to another, fails to convey the necessary information for an effective propagation in the presence of such constraints.

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). The underlying idea is to use an MDD with restricted size to encode a relaxation of the feasible solution space of a CSP. The MDD would then be considered as part of the constraint store, also collecting the inferences performed by each constraint in a structured way. The restriction on its size is controlled by a parameter K that imposes a limit on the number of nodes in any of its layers (or *width*). In particular, for $K = 1$ the MDD would be equivalent to a domain store, while K sufficiently large corresponds to an exact representation of the feasible set.

The motivation to complement the domain store with limited-width 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 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 constraint (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 when the objective function is associated with sequence-dependent setup times.

We begin by introducing MDD-based CP and the formulation of our problem in terms of a `disjunctive` global constraint that our filtering techniques will be developed for. Next, we present the MDD encoding for `disjunctive` and discuss in detail the main operations related to the MDD processing, namely *filtering* and *refinement*. We then describe how to integrate the MDD-based reasoning with current domain filters. Finally, we present our computational results, which is followed by the concluding remarks.

Preliminaries and Notation

MDDs Let $C = (X, D, \mathcal{C})$ be a CSP with n variables $X = \{x_1, \dots, x_n\}$, discrete domains $D(x_i) \in D$ for each $x_i \in X$, and constraints $\mathcal{C}$. We assume the variables in X are ordered according to their indices. A *Multivalued Decision Diagram* (MDD) for C is a directed acyclic graph whose nodes are partitioned into $n+1$ sets $L_1, \dots, L_{n+1}$, denoted by *layers*. Layers L_1 and L_{n+1} consist of single nodes; the root $\mathbf{R}$ and the terminal $\mathbf{T}$, respectively. The width w of an MDD is given by $w := \max_{1 \leq i \leq n+1} |L_i|$.

All arcs of the MDD are directed from nodes in layer L_i to nodes in layer L_{i+1}, for some $i = 1, \dots, n$. The set of incoming and outgoing arcs at node v of the MDD are given by $\delta^+(v)$ and $\delta^-(v)$, respectively. Each arc (u, w) leaving layer L_i is labeled with an element $\phi_{u,w} = v \in D(x_i)$ and represents the assignment $x_i = v$. In addition, a label v is not allowed to appear more than once in the set of outgoing arcs of a node. Hence, a path from $\mathbf{R}$ to a node $u \in L_i$ can be denoted by the edge labels $v_1, \dots, v_{i-1}$ on the path and *identifies* a partial assignment $p := (x_1, \dots, x_{i-1}) = (v_1, \dots, v_{i-1})$. For the figures in this work, 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.

The set of paths from $\mathbf{R}$ to a node u is given by $\mathcal{P}_u^{\mathbf{R}}$, also used to denote all partial solutions identified by paths from $\mathbf{R}$ to a node u. Analogously, we define $\mathcal{P}_{\mathbf{T}}^u$ the set of paths from u to $\mathbf{T}$. In particular, a path $(v_1, \dots, v_n)$ from $\mathbf{R}$ to $\mathbf{T}$ is *feasible* for the constraint system $\mathcal{C}$ if the identified assignment is consistent with $\mathcal{C}$. A node v is *infeasible* if all paths from $\mathbf{R}$ to $\mathbf{T}$ passing through v are infeasible. An *exact* MDD contains all (and only) feasible paths for $\mathcal{C}$.

Two nodes $u, v \in L_j$ are *equivalent* in a MDD if $\mathcal{P}_{\mathbf{T}}^v = \mathcal{P}_{\mathbf{T}}^u$, that is, the set of *completions* for the partial assignments identified by the paths from $\mathbf{R}$ to nodes u and v is exactly the same for both nodes. An MDD is *reduced* if no two nodes in any layer are equivalent; thus, reduced MDDs are the most compact MDD representation of the feasible set of a CSP for a particular variable ordering $x_1, \dots, x_n$. There is a unique reduced MDD M for a given variable ordering, which can be found in polynomial time in the size of M in several relevant cases (Andersen et al. 2007; Bergman, van Hoeve, and Hooker 2011). We henceforth differentiate between exact and reduced MDD: Both encode exactly the set of feasible solutions of a CSP, but the latter does not contain any two equivalent nodes.

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

MDD-based Constraint Programming The Constraint Programming framework in which limited-width MDDs are used as a constraint store is referred to as *MDD-based Constraint Programming*. The propagation process is based on a *filtering* and a *refinement* operation that are performed by each constraint one at a time.

MDD Filtering is a generalization of domain filtering and consists of removing arcs that do not belong to any feasible path with respect to $\mathcal{C}$. It can be systematically approached as follows (Hoda, van Hoeve, and Hooker 2010). With each MDD node u we associate a *state* I_u, which represents a local information that is dependent on the constraint type under consideration. The decision as to whether to delete an arc (u, v) is then based on some inference performed over I_u and I_v according to the constraint semantics. This state must be a function of the labels in $\delta^+(u)$ and on the previous states $\{I_v : (v, u) \in \delta^+(u)\}$; as such, all states can be computed efficiently through a single top-down pass (from L_1 to L_{n+1}). We can analogously define the states with respect to a bottom-up perspective, in which the state of u depends only on the labels of $\delta^-(u)$ and $\{I_v : (u, v) \in \delta^-(u)\}$.

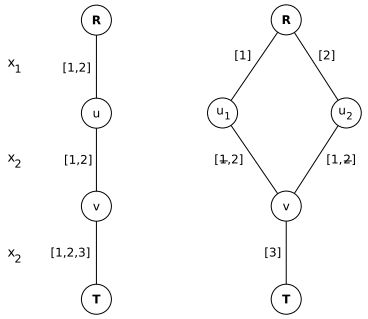
Refinement is the process of adding nodes and arcs to the MDD, so as to strengthen the relaxation it represents without violating the maximum width K. This can be accomplished by partitioning the set of incoming arcs of a node, which is performed by observing node equivalence tests or heuristic criteria (Hadzic et al. 2008). In this paper we develop a refinement procedure specific for disjunctive problems that can be easily generalized to other constraints.

Example 1 We present an example of MDD processing for the `alldifferent` constraint, first 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 `alldifferent`(x_1, x_2, x_3). Typically, the first MDD considered in this framework has a width of 1 and one arc for each domain value, which yields the same relaxation as the domain store. This is presented in Figure 1a.

The `alldifferent` constraint receives the MDD of Figure 1a as input and carries out filtering and refinement op-

erations. For the filtering, notice first that all the paths in $\mathcal{P}_v^{\mathbf{R}}$ are labeled with values $\{1, 2\}$. Since each of these paths corresponds to an assignment of $\{x_1, x_2\}$, both values will be necessarily taken by these variables. Hence, the filtering mechanism could deduce that no more variables can take any of these values, and 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, partitioning its incoming arcs as showed in Figure 1b. Since the value 1 belongs to all paths in $\mathcal{P}_{u_1}^{\mathbf{R}}$, it cannot be assigned to any other variable, so we can remove the arcs in $\delta^-(u_1)$ with label 1. By an analogous reasoning, we remove the label 2 from arcs (u_2, 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 completion time of i). Let $s = \{s_1, \ldots, s_n\}$ be such that each $s_i \in [r_i, d_i - p_i]$ is a variable representing the *start time* of activity $i \in \mathcal{A}$. Also, 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 a 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 $i = 1, \ldots, n-1$. The `disjunctive` constraint is also referred to as *unary resource*, since it represents non-preemptive resources with a capacity of one, and each activity requires one unit of the resource during its execution (Baptiste, Le Pape, and Nuijten 2001). We consider that this constraint might be 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 equivalently 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 suggestion 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_j now be a variable representing the j-th activity to be performed on the schedule, and e_i the completion time of activity $i \in \mathcal{A}$. Constraint (1) can be equivalently written as follows.

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

$$e_{\pi_{j-1}} + t_{\pi_{j-1},\pi_j} + p_{\pi_j} \leq e_{\pi_j}, \qquad j = 2, \ldots, n, \tag{3}$$

$$D(e_{\pi_j}) = [r_{\pi_j} + p_{\pi_j}, d_{\pi_j}], \qquad j = 1, \ldots, n, \tag{4}$$

$$D(\pi_j) = \mathcal{A}, \qquad j = 1, \ldots, n. \tag{5}$$

Constraint (2) enforces π to represent a valid activity execution sequence. 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 using `element` constraints. 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 in our approach.

A valid MDD representation can be directly 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_j corresponds to the j-th activity to be scheduled, $j = 1, \ldots, n$. 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 denote this encoding by *permutation MDD*, since any path corresponds to a permutation of $\mathcal{A}$. The completion time variables are not explicitly represented in the MDD, but rather implied from any particular MDD path under consideration. To this end, let $p = (\pi_1^*, \ldots, \pi_{j-1}^*)$ be any path from $\mathbf{R}$ to a node $v \in L_j$. We define $e_i^{(p)}$ and $f_i^{(p)}$ as the minimum and maximum completion time, respectively, of activity i in path p if we set $\pi_l = \pi_l^*, l = 1, \ldots, j-1$, and observe constraints (2) to (5).

The properties and filtering techniques developed here strongly rely on this particular variable ordering for the MDD representation, which might not be advantageous in all problem domains. Nevertheless, we argue that the chronological ordering of activities, as represented by our encoding, provides a natural and efficient way of tackling a large number of scheduling problems; it is usually explored in scheduling data structures in both search and filtering (Baptiste, Le Pape, and Nuijten 2001).

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 for this disjunctive problem is depicted in Figure 2. In particular, the path $p = (\mathbf{R}, u, w, \mathbf{T})$ yields $e_1^{(p)} = 7$, $e_2^{(p)} = 10$, $e_3^{(p)} = 2$, $f_1^{(p)} = 9$, $f_2^{(p)} = 12$, and $f_3^{(p)} = 6$. □

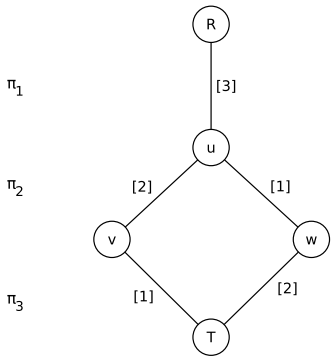


Figure 2: Reduced MDD for Example 2.

Let M be a limited-width permutation MDD for an instance of the `disjunctive` constraint. We now define a *state* for each node of M, which will explore the structure of the permutation MDD for the purposes of filtering and refinement. With each node $v \in L_j$, 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 completion time* at v:

$$E_v := \min\{e^{(p)}_{\pi^*_{j-1}} : p = (\pi_1^*, \ldots, \pi_{j-1}^*) \in \mathcal{P}_v^{\mathbf{R}}\}. \tag{7}$$

The intuition behind E_v is that it *accumulates* the time up to the end of the $(j-1)$-th activity, thus allowing us to make inferences on the completion times variables without representing them explicitly through additional layers in M.

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 $L_2, \ldots, L_j$ were already calculated. Suppose we wish to compute I_v for a node $v \in L_{j+1}$. The `alldifferent` states are obtained by applying the rules from (Andersen et al. 2007):

$$A_v^{\downarrow} := \bigcap \{A_u^{\downarrow} \cup \{\phi_{u,v}\} : (u, v) \in \delta^+(v)\}, \tag{8}$$

$$S_v^{\downarrow} := \bigcup \{S_u^{\downarrow} \cup \{\phi_{u,v}\} : (u, v) \in \delta^+(v)\}. \tag{9}$$

To compute E_v, notice first that inequalities (3) and (4) impose a lower bound on e_{π_j}:

$$e_{\pi_j} \geq e_{\pi_{j-1}} + p_{\pi_j} + t_{\pi_{j-1}, \pi_j}, \tag{10}$$

$$e_{\pi_j} \geq r_{\pi_j} + p_{\pi_j} \tag{11}$$

which implies, for any path $p = (\pi_1^*, \ldots, \pi_j^*) \in \mathcal{P}_v^{\mathbf{R}}$,

$$e^{(p)}_{\pi^*_j} = \max\left\{r_{\pi^*_j},\ e^{(p)}_{\pi^*_{j-1}} + t_{\pi^*_{j-1}, \pi^*_j}\right\} + p_{\pi^*_j}. \tag{12}$$

Applying a dynamic programming argument using equality (12), one can show that

$$E_v = \min\left\{E'_{u,v} : (u, v) \in \delta^+(v)\right\} \tag{13}$$

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

$$E'_a = \max\left\{r_{\phi_a}, E_u + \min_{e \in \delta^+(u)}\{t_{\phi_e, \phi_a}\}\right\} + p_{\phi_a} \tag{14}$$

and represents the minimum completion time if the label in arc (u, v) is necessarily considered in the assignment.

Refinement of the Permutation MDD

In this section we develop an incremental refinement technique for the permutation MDD, in which nodes are selected to be split so as to strengthen the quality of the relaxation the diagram provides. 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 1.

Theorem 1 *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 $(\mathbf{R}, v)$, $v \in \delta^-(\mathbf{R})$: $(1), (2), \ldots, (n)$. 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 $\mathbf{R}$ is infeasible (and therefore the problem as well) by performing $\mathcal{O}(n^2)$ node equivalence tests, one for each pair of the n assignments. But checking node infeasibility 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). ■

Hence, we are not able to identify node equivalence with a polynomial-size state for nodes (if $P \neq NP$). Nevertheless, we can identify necessary conditions for node equivalence by taking the conjunction of individual equivalence tests for constraints (2)-(5) in the reformulation (Hadzic et al. 2008). We provide one of such necessary conditions in Lemma 2.

Lemma 2 *Let M be a limited-width MDD representing an arbitrary* `disjunctive` *instance. For a node u, define $T_u(k) := \max_{(w,u)\in\delta^+(u)} \{d_{\phi_{w,u}} + t_{\phi_{w,u},k}\}$. Given two nodes $u, v \in L_j$, the conditions*

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

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

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 $j-1$ variables are taking the corresponding $j-1$ 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 $\{\pi_{k-1}^u, \pi_{k-1}^v\} \subseteq \{\phi_{w,t} : (w,t) \in \delta^+(u) \cup \delta^+(v)\}$, 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 impose particular restrictions on the input parameters, which could be exploited to develop necessary and sufficient equivalence tests. For instance, problems that involve minimizing tardiness or weighted flow (Pan and Shi 2008; Pinedo 2008) usually do not take deadlines into account. In this case Lemma 3 can be directly applied.

Lemma 3 *Suppose the* `disjunctive` *instance M represents is such that $d_i = +\infty$ for all $i \in \mathcal{A}$ and has arbitrary release and setup times. Two nodes $u, v \in L_j$ 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 idea is to partition nodes based on some global information that represents the best possible partitioning of a layer. 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 split a node $v \in L_j$ into nodes $v_1, \ldots, v_{|\delta^+(v)|}$ containing a single incoming arc. This corresponds to one state for each arc $(u, v) \in \delta^+(v)$, denoted by $I'_{u,v} := (A^{\downarrow}_{u,v}, S^{\downarrow}_{u,v}, E'_{u,v})$. The elements in $I'_{u,v}$ are such that $E'_{u,v}$ is given in (14), $A^{\downarrow}_{u,v} := A^{\downarrow}_u \cap \{\phi_{u,v}\}$, and $S^{\downarrow}_{u,v} := S^{\downarrow}_u \cup \{\phi_{u,v}\}$; the last two equalities are derived from (8) and (9), respectively. The arc (u, v) and its respective state $I'_{u,v}$ are then stored in a data structure $\mathcal{R}$. It is already possible to apply filtering at this point of the algorithm, which would be equivalent to not adding infeasible arcs in $\mathcal{R}$.

In the *refinement phase*, we partition the set $\mathcal{R}$ into at most K groups V_i. These partitions will define the new nodes in the layer when the procedure terminates; namely, for each V_i, a new node v will be created with incoming arcs given by the arcs in V_i. This is represented in steps 12 to 16 in Algorithm 1. We replace a layer L_j by a new layer composed by these new nodes, which represents the result of the split.

The partitions should be defined by taking the state $I'_{u,v}$ into account, for instance using particular heuristic criteria and the equivalence tests described in this section. The set $\mathcal{R}$ gives a *global* view of the layer in that it allow 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 $E_v = \min_{(u,w)\in V}\{E'_{u,w}\}$, $A^{\downarrow}_v = \bigcap_{(u,w)\in V}\{A^{\downarrow}_{u,w}\}$, and $S^{\downarrow}_v = \bigcup_{(u,w)\in V}\{S^{\downarrow}_{u,w}\}$, 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 partitions in this work were created in a way that $K-1$ new nodes were associated with the lowest $K-1$

Algorithm 1: MDD Refinement (Input: MDD M)

```
begin
    for layer indices j = 2, ..., n do
        // 1. Node processing phase
        R := ∅
        foreach node v ∈ L_j do
            foreach arc (u, v) ∈ δ⁺(v) do
                Compute I'_{u,v} and filter (u, v)
                R := R ∪ {(I'_{u,v}, (u, v))}
        // 2. Refinement phase
        Partition R into (at most) V_1, ..., V_K groups
        L'_j := ∅
        foreach group V_i do
            Create a new node v, add it to L'_j
            δ⁺(v) := {(u, v) : ∃ (u, w) ∈ V_i}
            δ⁻(v) := ⋃{δ⁻(w) : ∃(u, w) ∈ V_i}
            Compute I_v
        Replace L_j by L'_j
end
```

values possible for E_v, breaking ties by giving priority to the largest new $A_v^{\downarrow}$. This was verified to be very effective in practice, since intuitively smaller values for E_v forces paths with small makespan to be present in the MDD, while larger $A_v^{\downarrow}$ countermeasures this procedure by preventing infeasible paths in the MDD with too many repeated activities.

Filtering

This section presents the 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-up pass by setting $A_{\mathbf{T}}^{\uparrow} = S_{\mathbf{T}}^{\uparrow} = \emptyset$ and for any $v \in L_j$, $j \geq 2$,

$$A_v^{\uparrow} := \bigcap \{A_u^{\uparrow} \cup \{\phi_{v,u}\} : (v,u) \in \delta^-(v)\},$$
$$S_v^{\uparrow} := \bigcup \{S_u^{\uparrow} \cup \{\phi_{v,u}\} : (v,u) \in \delta^-(v)\},$$

which stem from (8) and (9), respectively (Hoda, van Hoeve, and Hooker 2010). The state F_v represents the *maximum completion time* at $v \in L_j$, defined by

$$F_v := \max\{f^{(p)}_{\pi^*_{j-1}} : p = (\pi^*_1, \ldots, \pi^*_{j-1}) \in \mathcal{P}_v^{\mathbf{R}}\}. \tag{18}$$

The state F_v can be computed analogous to E_v. Namely, let $t_{\min}(v,u) = \min_{(w,v)\in\delta^+(v)}\{t_{\phi_{w,v},\phi_{v,u}}\}$. From inequalities (3) and (4), we have an upper bound on variable $e_{\pi_{j-1}}$ given by $e_{\pi_{j-1}} \leq \min\{d_{\pi_{j-1}}, e_{\pi_j} - t_{\pi_{j-1},\pi_j} - p_{\pi_j}\}$. Applying a dynamic programming argument, we obtain

$$F_v = \min\left\{\max_{(u,v)\in\delta^+(v)} d_{\phi_{u,v}},\ \max_{(v,u)\in\delta^-(v)}\{F'_{v,u}\}\right\}, \tag{19}$$

where $F'_{v,u} := F_u - t_{\min}(v,u) - p_{\phi_{v,u}}$.

The first term of the max in (19) and $t_{\min}(v,u)$ 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 to remove inconsistent values from the arc domains 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 $u \in L_j$ and $v \in L_{j+1}$. From the `alldifferent` constraint (2), arc (u,v) can be removed if any of the conditions below hold (Hoda, van Hoeve, and Hooker 2010).

$$\phi_{u,v} \in A_u^{\downarrow} \cup A_v^{\uparrow}, \tag{20}$$
$$|S_u^{\downarrow}| = j - 1 \ \ \wedge \ \ i \in S_u^{\downarrow}, \tag{21}$$
$$|S_v^{\uparrow}| = n - j + 1 \ \ \wedge \ \ i \in S_v^{\uparrow}. \tag{22}$$

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

$$\max\left\{r_{\phi_{u,v}},\ E_u + \min_{(w,u)\in\delta^+(u)}\{t_{\phi_{w,u},\phi_{u,v}}\}\right\} + p_{\phi_{u,v}} > \min\left\{d_{\phi_{u,v}}, F_v\right\}. \tag{23}$$

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 stronger 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. This is done by assigning appropriate arc weights and computing the shortest path from $\mathbf{R}$ to $\mathbf{T}$. For instance, to minimize sum of setup times, we assign a weight $w_{u,v} = \min_{a\in\delta^+(u)} t_{a,\phi_{u,v}}$. To minimize makespan, no shortest path is required; the lower bound is $E_{\mathbf{T}}$ by definition.

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 also 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 4 *Let M be an exact 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 $(\mathbf{R}, \ldots, v, \ldots, \mathbf{T})$ 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 5 *The set of all precedence relations that must hold in a* `disjunctive` *instance can be efficiently extracted from its exact MDD M in $\mathcal{O}(n^2|M|)$.*

Proof. Construct a digraph $G^* = (\mathcal{A}, E^*)$ by adding an arc (i, j) to E^* 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 pair of activities takes $O(n^2)$ for each node in M, and hence the time complexity to construct G^* is $O(n^2|M|)$. According to Theorem 4 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 4.

Corollary 6 *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 4 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, which is a subset of the solutions represented by M. ■

By Corollary 6, the precedence relations implied by the solutions of a limited-width MDD M can be extracted by applying the algorithm in Corollary 5 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, or be provided directly to constraint-based solvers that may benefit from them. Since M has at most $\mathcal{O}(nK)$ nodes and $\mathcal{O}(nK^2)$ arcs, the time to extract the precedences has a worst-case complexity of $\mathcal{O}(n^3K^2)$ by this particular algorithm.

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 4 and the state definitions immediately yield the following filtering rules. Arc (u, v) with $u \in L_j$, $v \in L_{j+1}$ can be removed if any of the conditions below holds:

$$j \in A_u^{\downarrow}, \qquad \exists\,(\phi_{u,v}, j) \in \mathcal{P}, \quad (24)$$

$$|S_u^{\downarrow}| = j - 1 \;\wedge\; j \in S_u^{\downarrow}, \qquad \exists\,(\phi_{u,v}, j) \in \mathcal{P}, \quad (25)$$

$$j \in A_v^{\uparrow}, \qquad \exists\,(j, \phi_{u,v}) \in \mathcal{P}, \quad (26)$$

$$|S_v^{\uparrow}| = n - j + 1 \;\wedge\; j \in S_v^{\uparrow}, \qquad \exists\,(j, \phi_{u,v}) \in \mathcal{P}. \quad (27)$$

The conditions above 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 in 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 $(r_3, d_3, p_3) = (14, 35, 5)$.

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 the filters and refinement presented here, we derive the MDD presented in Figure 3. If the precedences are kept in a set $\Omega_i := \{j \ll i : j \in \mathcal{A}\}$ 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 might be used to update the start time variable of activity 3. These same precedences could also be inferred by analyzing the activity time bounds; nonetheless, the MDD could derive much more complicated relations that may depend on how several subsets of activities interact among themselves. □

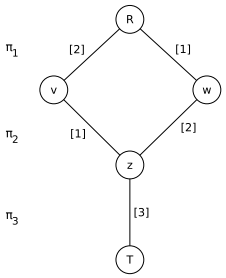


Figure 3: MDD for Example 3.

Computational Experiments

We have implemented our MDD propagation and refinement algorithms as a global constraint in ILOG CP Optimizer 2.3

(the source code will be made available at the author's website). We evaluate our methods on problem instances arising from the TSP 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.

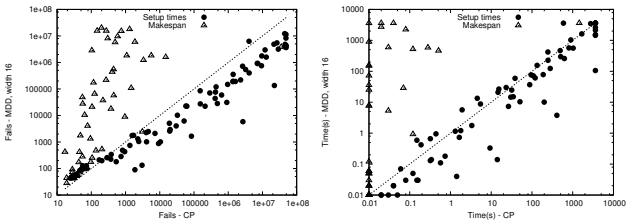
We compared three techniques: a formulation with the unary resource constraint from Ilog CP Optimizer (`CP`), *IloNoOverlap*, set with *extended* filtering; the standalone MDD filtering (`MDD`); and a combined approach (`MDD+CP`). However, a relatively loose integration is considered for `MDD+CP`, since the precedences inferred by the Ilog solver are only partially available. Tests were implemented in C++ and ran on an Intel Xeon E5345 with 8 GB RAM. The MDD width was set to 16 in both MDD approaches.

We selected 75 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. Among these, we had 10 instances containing between 10 and 19 activities, 48 instances with 20 activities, 10 instances with 40 activities, and 7 instances with 60 activities. The average time window width for the `Dumas` instances varied between 20 and 180 for instances containing up to 20 activities and 20 and 40 for larger instances.

The results comparing the `CP` and `MDD` approaches are shown in Figure 4. We applied a lexicographic search, in which activities were recursively assigned to be first in the schedule according to an arbitrary order. A time limit of 3600 seconds was imposed. For the sum of setup times, we can observe in Figure 4a that the `MDD` approach provided a stronger propagation than `CP`, even when Ilog was set to extended filtering. In total, `CP` exceeded time limit for 6 instances for which the `MDD` was able to prove optimality. The average relative improvement on the number of fails between `CP` and `MDD`, where the relative improvement between values a and b is given by $(b-a)/b$, was 0.48 (hence, the number of fails decreased by half on average). We observe in Figure 4b that in about 55% of the cases the `MDD` performed better than `CP`; that is, in 45% of the cases the reduction on the number of fails was not sufficient to compensate the overhead introduced by the MDD. The average relative time improvement between `CP` and `MDD` when `CP` performed better was 0.32, and the average relative time difference between `CP` and `MDD` when `MDD` was better was 0.54.

Comparing now the minimization of makespan for these instances, the `CP` filtering had a far superior performance than the standalone `MDD`, both with respect to number of fails and time. Most likely, this is due to the very well-developed filtering techniques for makespan that are not being explored by the `MDD`.

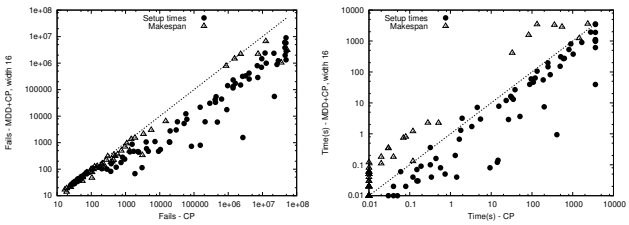
The results comparing the `CP` and `MDD+CP` approaches are shown in Figure 5. This is the most relevant analysis, since CP solvers usually run a portfolio of filtering techniques, to which the MDD-based approach could be added. For minimizing setup times, we observe in Figure 5a that



(a) Fails (b) Time

Figure 4: Performance between `CP` and `MDD`

the interaction between the filters yielded a stronger filtering than `MDD` and `CP` alone. The `MDD+CP` could solve 5 instances for which the `MDD` approach exceeded the time limit (and 11 instances that `CP` could not solve), and the average relative improvement of the number of fails between `CP` and `MDD` was 0.75. This had a significant effect on the times, as showed in Figure 5b; for all but three instances, the combined `MDD+CP` approach performed better than `CP` alone. The average time improvement was 0.45; hence, on average the time to prove optimality reduced by 45%. We observed orders of magnitude improvement for some instances; e.g., `n60w20.003` from `Dumas`, with 60 activities and average time window width of 20, was solved in 30 seconds by `MDD+CP`, but exceeded 1 hour for `CP`.



(a) Fails (b) Time

Figure 5: Performance between `CP` and `CP+MDD`

As for makespan, the `CP` approach generally had a better performance than `MDD+CP`, since the reduction on the number of fails was not sufficient to reduce the total solving time. Nevertheless, for only 10 cases tested (mainly instances with 40 activities and large time window width), the performance loss between both techniques was more than 10 seconds. Notice also that we have not yet explored the full integration between the `MDD` and `CP` approaches

Conclusion

In this paper we studied propagation techniques based on MDDs for disjunctive problems. We provided 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 especially when minimizing sequence-dependent setup times, usually a deficiency in CP solvers.

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.; 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. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: 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.