Tight MDD Integration for Global Constraints: the Multi-Cumulatives Case

Hugo Apeloig, Nicolas Beldiceanu, Odile Bellenguez, Guillaume Massonnet, Gilles Simonin

Tight MDD Integration for Global Constraints: the Multi-Cumulatives Case

Hugo Apeloig[1], Nicolas Beldiceanu[1], Odile Bellenguez[1], Guillaume Massonnet[1], and Gilles Simonin[1]

IMT Atlantique, LS2N, UMR CNRS 6004, 44307, Nantes, France

Abstract. In problems with objects that have multiple feasible configurations, also known as multi-mode problems, a lack of communication often arises between global constraints and side constraints. To address this issue, we propose a novel approach where object configurations are represented using multi-valued decision diagrams (MDDs). Within this framework, global constraint filtering algorithms can directly query the MDDs to obtain a more accurate estimate of the bounds of a variable. We formally introduced this new method and a direct application is proposed through the Cumulative constraint. We propose two new constraints, named MDD-Cumulative and MDD-Multi-Cumulatives. The former is based on the state-of-the-art timetabling algorithm, while the latter takes advantage of the method for filtering mode assignments. We provide numerical evidence that demonstrates the effectiveness of these methods over the state-of-the-art multi-mode resource-constrained project scheduling problem (MM-RCPSP) benchmarks.

Keywords: constraint programming, multi-valued decision diagram, multi-mode, RCPSP, cumulative, timetabling, scheduling.

1 Introduction

Constraint Programming (CP) has proven effective in addressing scheduling problems with additional resources (*e.g.*, *resource constrained project scheduling problem* [4] (RCPSP)), notably in checking the feasibility of solutions wrt resource demands and capacities. Since its introduction in [1], the Cumulative constraint and its filtering algorithms have been extensively studied in the literature [8,25].

In this work, we focus on problems considering *objects* with *attributes*. For example the *multi-mode* RCPSP [10] (MM-RCPSP) where the execution of a task can be performed in alternative configurations, called *modes*, with a variety of constraints between its attributes. This work is originally motivated by an industrial use case modelled as an MM-RCPSP with additional constraints [17].

When considering such problems, two issues may arise. Firstly, a lack of communication can occur between the global constraints of the problem (*e.g* Cumulative) and the side constraints maintaining objects attributes consistency. Secondly, considering a CP solver library, variants of a problem may ne-

cessitate multiple variants of the same constraint, e.g. the original CUMULATIVE constraint of CHIP had up to 12 variants [21].

Modes are typically modelled (i) using *optional* tasks duplicates [26] or (ii) as a set of ELEMENT constraints over the attributes of each task [46]. However, case (i) cannot handle certain mode-specific conditions and may lead to an excessive number of duplicates. On the other hand, case (ii) risks underestimating values in the reasoning performed by CUMULATIVE, which can only be strengthened once assignments have already been made. Both approaches limit the reasoning capability of the solver, either by potentially introducing many optional tasks or by weakening filtering.

This motivates the use of alternative representations capable of capturing mode-dependent information in a more compact fashion, enabling stronger interactions between constraints. One such structure is the *Multi-valued Decision Diagram (MDD)*, a graph-based representation widely used in discrete optimisation, and particularly in CP [18]. To the best of our knowledge, MDDs have been mainly exploited either to solve a complete problem [23] or to represent a *global constraint* [38]. The idea of integrating them as parameters into global constraints has not yet been explored.

The remainder of this paper is organised as follows. We start by recalling in Section 2 useful notations and concepts of Constraint Programming and Multi-Valued Decision Diagrams, as well as the current state of the art over CUMULATIVE constraints and MDD application in CP. In Section 3 we discuss the advantages of integrating MDDs within global constraints. In Section 4, we present two novel global constraints. Firstly, the MDD-CUMULATIVE with its filtering algorithm, which is a straightforward application of the method. Secondly, the MDD-MULTI-CUMULATIVES, which takes advantage of the method to reason, furthermore, over the assignment variables, which is useful for industrial problems. Numerical results and their analysis are presented in Section 5. Finally, we conclude the paper in Section 6, discuss current limitations and propose possible extensions for future research.

2 Background and Limitations

2.1 Constraint Programming

Notations Given a CP model, let $\mathcal{X}$ be the set of its integer variables, taking values in finite domains $\mathcal{D}$, and $\mathcal{C}$ its set of constraints. Given a constraint $c \in \mathcal{C}$, $\mathcal{X}^c \subseteq \mathcal{X}$ is the set of variables of the constraint c, i.e. the *scope* of c. An assignment of each variable $x \in \mathcal{X}$ to a value $v \in \mathcal{D}$ is denoted $A(\mathcal{X})$. $A(\mathcal{X}^c)$ denotes the restriction of the assignment A over the scope of c. Given a variable $x \in \mathcal{X}$, we let $\underline{x}$ and $\overline{x}$ denote the smallest and largest possible value of x, respectively.

Constraint Programming Constraint Programming (CP) employs global constraints to enhance the reasoning capabilities beyond those achievable with

a collection of simple, individual constraints [12]. These constraints use specialised algorithms that leverage domain knowledge to improve propagation and pruning. In general, solving a CP problem relies on a *backtracked search tree*. At any node of the tree, several subtrees are created by partitioning the domain(s) of one or more variables of the problem (*e.g.* assignment of a value or domains splitting). A *propagation algorithm* is then applied to remove from the variable domains any value that is inconsistent with the constraints. The set of inconsistent values detected, referred to as reasoning, is derived from deductions generated by individual constraints and communicated indirectly to other constraints via domain reductions. Since this communication occurs only through value pruning, constraints exchange information only when their internal inferences are strong enough to eliminate domain values. This mechanism makes it straightforward to define generic global constraints and design efficient filtering algorithms. Determining whether a valid assignment exists for a certain CP problem is *NP-Complete* [44].

Several constraints, such as Cumulative, Alternative or DiffN, will be introduced when they first occur. The Element(I, T, V) constraint [28] defined on two variables I and V, and an array of variables $T[1..n]$. It is satisfied if and only if V is equal to the I-th entry of table T.

2.2 Scheduling in CP and the Cumulative Constraint

Consider the problem of scheduling a set of tasks $\mathcal{J}$ wrt a cumulative renewable resource with a finite capacity $B \in \mathbb{N}$. A task $i \in \mathcal{J}$ is characterised by its *starting* time S_i, *duration* P_i, *completion* time C_i, and *resource consumption* H_i. To be processed, a task must be executed wrt a set of *modes*. In multi-mode problems, it is also classic to consider an additional assignment variable whose value corresponds to the mode chosen. We denote it Y_i. Regardless of the other tasks, the modes of a task i correspond to the set of feasible assignments for variables Y_i, S_i, P_i, C_i, H_i, provided that the constraint $C_i = S_i + P_i$ holds. Each task must operate according to one of its available modes. Moreover, at any time point t, the total consumption of the renewable resource by all tasks that overlap t cannot exceed the resource's capacity B. We denote n the number of tasks and R the number of renewable resources.

The Cumulative Constraint The Cumulative constraint is a well-known global constraint [1] whose filtering algorithms have been progressively enhanced over the last 30 years [5]. It is defined over a set of tasks, and ensures that the cumulated resource consumption never exceeds a maximum capacity at any time point. Multiple filtering procedures exist for Cumulative. The most popular are the energetic reasoning [49,14] and the *timetabling reasoning* [34,8,25]. Due to its simplicity, we will use timetabling and show how to tightly integrate it with MDD. To make the paper self-contained, we quickly review its main principles, as well as a state-of-the-art timetabling algorithm.

Timetabling Algorithm Before sketching the algorithm, we review two notions: the mandatory part of a task, and the resource profile for a set of tasks.

Consider a task i, where the values of P_i and H_i are constant. If $\overline{S_i} < \underline{C_i}$, then task i is active during the interval $[\overline{S_i}, \underline{C_i} - 1]$. This interval is referred to as the *mandatory part* of task i [25]. It follows that the residual capacity of the resource over the interval $[\overline{S_i}, \underline{C_i} - 1]$ is at most $B - H_i$. The resource profile corresponds to the aggregation of all such mandatory parts and their associated resource consumption across the entire set of tasks.

We now recall the filtering algorithm described in [25] as it is the simplest state-of-the-art algorithm for the cumulative constraint based on timetabling. The timetabling algorithm is based on the resource profile. It updates $\underline{S_i}, \underline{C_i}$ and $\overline{S_i}, \overline{C_i}$ for each task i. First, it computes the mandatory parts of each task and the corresponding profile, which consists of at most $2n+1$ rectangles. The height of the j-th rectangle in this profile is the sum of the heights of all tasks, whose mandatory portion spans the entire interval $[a_j, b_j]$ associated with rectangle j. We denote D_j the demand, or height, of rectangle j. A conflict arises when a task overlaps a rectangle, causing the total demand to exceed B. To begin, the filtering algorithm builds the resource profile. Then, for each task i that is not yet instantiated, it tries to adjust its earliest start and latest completion time based on the profile, excluding the mandatory part of task i. If task i is in a left conflict, *i.e.* $\exists a_j : \underline{S_i} \leq a_j \wedge a_j < \underline{S_i} + P_i \wedge D_j + H_i > B$, its earliest start time is pushed forward. Symmetrically, if task i is in a right conflict, *i.e.* $\exists a_j : \overline{C_i} \geq b_j \wedge b_j > \overline{C_i} - P_i \wedge D_j + H_i > B$, then the latest completion time of task i is pushed backward. The overall algorithm runs iteratively until a fixed point is reached, meaning a full pass is completed without creating any new mandatory parts. This timetabling algorithm will be extended later on to include queries to MDD to get tighter bounds for multi-mode problems.

2.3 Multi-valued Decision Diagrams

Given k variables with their associated domains, one can construct a Multi-valued Decision Diagram (MDD) as a rooted, directed acyclic graph with $k+1$ layers of nodes. Each non-terminal layer is associated with a specific variable, and the outgoing arcs from a node in layer ℓ correspond to a distinct value (or range of values) that the ℓth variable can take. The final layer contains the terminal nodes, which represent a complete assignment (or restriction) of all variables [39]. Let r denote the root node and t the leaves.

The *width* of an MDD is defined by the maximum number of nodes at each level. Two nodes are equivalent if the combinations of values encoded by their outgoing *labeled* path to the terminal nodes are the same. Equivalent nodes can then be merged to reduce the width of the MDD [39]. This reduction depends on the ordering of the variables (*i.e* layers), since different orderings can lead to different numbers of states [43]. MDDs can be viewed as *unfolded automata* [41]. In this work, as the maximum width stays tractable by creation, we only consider exact MDDs, i.e. MDDs in which any r-t path corresponds to a feasible solution.

Given a set of variables $\mathcal{X}$ and an MDD $\mathcal{M}$ over $\mathcal{X}$, we say that an assignment $A(\mathcal{X}) : A(X_1) = v_1, A(X_2) = v_2, \ldots, A(X_{|\mathcal{X}|}) = v_{|\mathcal{X}|}$ is *valid* if and only if there exists an r-t path in $\mathcal{M}$ such that for all $\ell \in \{1, \ldots, |\mathcal{X}|\}$, its ℓ-th layer-arc has value v_ℓ. [20,40] propose the constraint MDDC$(\mathcal{X}, \mathcal{M})$ to check the validity of variable assignments. We denote $\mathcal{P}_\mathcal{M}$ the set of all valid assignments over $\mathcal{X}$ (or, equivalently, the set of all valid r-t paths in $\mathcal{M}$). The constraint MDDC filters domains equivalently to a Regular constraint [41], and achieves arc-consistency when the associated MDD is exact.

2.4 Related Work

MDDs have already been used for the resolution of combinatorial problems by encoding the problem, either entirely or partially [23]. In the specific context of CP, MDDs have notably been exploited to represent constraints [3,19,27] or to enhance propagation [11,22,30,2].

Boolean satisfiability solving techniques such as *Lazy Clause Generation* (LCG) methods have already been applied with success for the Cumulative constraint [48] and multi-mode RCPSP problems [45,47]. However, LCG necessitates an additional integration within the CP solver. In the case of Multi-Mode Cumulative [45,47], the clause generation was based on a minimal mandatory mode. A possible extension would be to study the combination of MDD tight integration in a filtering algorithm and LCG methods. In such case, clauses could be augmented in order to take into account information obtained by the MDD and not only the fictitious minimal mandatory mode.

Another well-known method for scheduling problems with multi-mode configurations is the Alternative constraint SAT the CPOptimizer solver [33]. It is based on optional tasks that may or may not be present in the resulting schedule [32]. Given a task i, modes are defined as a collection of optional tasks and the Alternative constraint enforces i to be exactly one of those tasks. Optional tasks have proven efficient for variants of the RCPSP problems, see [29,37] among others. However, they face scalability issues when the number of modes becomes large.

3 MDDs Integration Method

We introduce a novel CP approach that uses MDDs to enhance CP reasoning for problems with multi-mode objects, i.e. objects whose attributes can be assigned different combinations of values based on their operational mode. The key idea is to encode each object's attributes as an MDD, which are then all integrated into a global constraint. This integration enables stronger domain reduction under specific assumptions. This section establishes the theoretical foundation for MDD-enhanced CP reasoning. We depict in Example 2 a running example to illustrate the method over the Cumulative constraint. In Section 4, we then demonstrate its practical applicability by presenting two new global constraints built upon this methodology.

3.1 Constraint Pattern Considered

We focus on problems defined over a collection of n *objects* $\{o_1, o_2 \ldots, o_n\}$, where each object is defined over a set of q *attributes*, fixed or variable. We consider the following conjunction of constraints: a single global constraint, GC, which restricts the set of objects and their attributes; and a set of side constraints, $\textsc{Ctr}_i$ (with $i \in [1, n]$), that restrict certain attributes of each object individually.

$$\left[\begin{array}{c} \mathrm{GC}([o_1, o_2 \ldots, o_n], \mathit{Parameters}) \wedge \\ \textsc{Ctr}_1(o_1) \wedge \textsc{Ctr}_2(o_2) \wedge \cdots \wedge \textsc{Ctr}_n(o_n) \\ \text{with } o_i = [a_{i1}, a_{i2}, \ldots, a_{iq}] \text{ for } i \in [1, n] \end{array}\right. \tag{1}$$

Two problems illustrating this pattern are, for instance, the *Multi-mode RCPSP* [10] or the *Orthogonal Packing Problem*, as illustrated by Example 1.

Example 1. In orthogonal packing problems, the objects correspond to rectangles, GC to the non-overlapping constraint DiffN, and $\textsc{Ctr}_i$ to constraints modelling the orientation of each rectangle. Each rectangle is represented by the coordinates X_i, Y_i of its bottom left corner, and by its width L_i and height H_i in $\{s_i, t_i\}$.

$$\left[\begin{array}{c} \textsc{DiffN}([o_1, \ldots, o_n]) \\ \bigwedge\limits_{i \in [1,n]} (L_i = s_i \wedge H_i = t_i) \vee (L_i = t_i \wedge H_i = s_i) \\ \text{where } o_i = [X_i, Y_i, L_i, H_i] \text{ (with } i \in [1, n]) \end{array}\right. \tag{2}$$

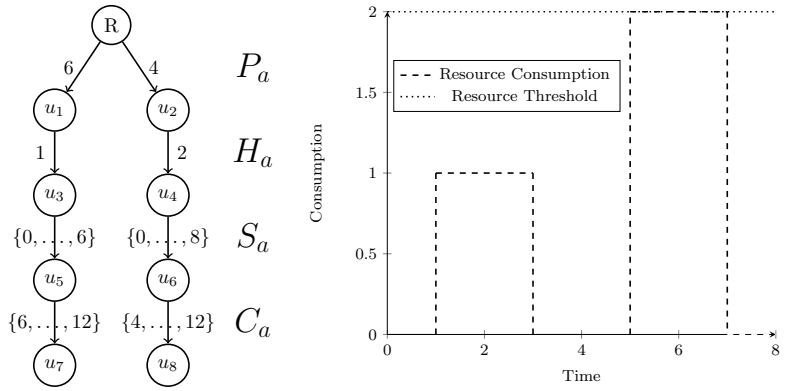
Example 2 (Running Example). In multi-mode RCPSP, the objects correspond to tasks, GC stands for a cumulative constraint, and $\textsc{Ctr}_i$ represents constraints that model the multi-mode aspect, the conjunction of two element constraints here: the duration P_i and the demand H_i of a task i may depend on its mode Y_i, corresponding to

$$\left[\begin{array}{c} \textsc{Cumulative}([o_1, o_2, \ldots, o_n], B) \wedge \\ \bigwedge\limits_{i \in [1,n]} \textsc{Element}(Y_i, [p_{i1}, \ldots, p_{im}], P_i) \wedge \textsc{Element}(Y_i, [h_{i1}, \ldots, h_{im}], H_i) \\ \text{where } o_i = [Y_i, S_i, P_i, C_i, H_i] \text{ (with } i \in [1, n]) \end{array}\right. \tag{3}$$

Consider a cumulative resource with a capacity of $B = 2$ and the consumption profile given in Figure 1b. Consider a task a with the following five attributes Y_a, S_a, C_a, P_a, H_a and a dependency between Y_a, P_a and H_a. Moreover, $\mathcal{D}(Y_a) = \{1, 2\}$, $\mathcal{D}(S_a) = \{0, \ldots, 8\}$, $\mathcal{D}(C_a) = \{4, \ldots, 12\}$, $\mathcal{D}(P_a) = \{4, 6\}$, $\mathcal{D}(H_a) = \{1, 2\}$ and the following restrictions must hold $Y_a = 1 \implies (P_a = 4 \wedge H_a = 2)$ and $Y_a = 2 \implies (P_a = 6 \wedge H_a = 1)$.

3.2 Analysing the Lack of Filtering

Communication Issue An issue may arise when addressing such problems, as the global constraint GC may underestimate (*resp.* overestimate) the value of certain attributes without considering the restrictions implied by the side



(a) Profile of the resource considered. (b) Profile of the resource considered.

Fig. 1: The MDD $\mathcal{M}_a$ (1a) and a resource's profile (1b) of Example 2 .

constraints. Given an object o_i, it is typical that GC considers all of its attributes simultaneously set to their minimal (*resp.* maximal) value, which is impossible wrt constraint $\textsc{Ctr}_i$. This issue is illustrated in Part (1) of Example 3. While this communication is typically established indirectly via domain reduction, a filtering procedure may greatly benefit from a stronger communication between the global and side constraints.

Implicit Assumptions To obtain reasoning over the attributes of an object o_i, GC filtering algorithm considers a subset of other objects. It implicitly assumes that certain attribute values of o_i are in interaction with this subset. In other words, it *makes implicit assumptions* over the value of a subset of attributes of o_i. However, those assumptions are not taken into account when GC asks for the minimal or maximal value of an attribute of o_i. This is illustrated in (2).

Example 3 (Continuation of Example 2). (1) When the timetabling algorithm is applied, minimum values of the attributes of a are considered. However, due to the modes of a, $P_a = \underline{P_a} \wedge H_a = \underline{H_a}$ cannot hold. In other words, it is not a valid support for $\textsc{Element}(Y_a, [4,6], P_a) \wedge \textsc{Element}(Y_a, [2,1], H_a)$. (2) In the timetabling algorithm, three attributes of a are checked over the profile. The starting time (S_a) and duration (P_a), to determine the rectangles of the profile that are in conflict, and the demand (H_a) to ensure it does not exceed the resource threshold. The timetabling algorithm implicitly assumes that a overlaps certain rectangles in the profile when filtering a task attribute against them. In this example, it assumes first that a only overlaps with the first rectangle; i.e. it assumes that $\underline{S_a} + \underline{P_a} \leq 5$. Then, it also considers $H_a = \underline{H_a}$ without taking this assumption into account. Finally, as the height of the rectangle is 1 and $B - 1 \geq \underline{H_a}$, the Cumulative constraint does not filter.

Objective We seek a method to bridge the gap between the implicit hypothesis made by the filtering algorithm of GC and the external constraints Ctr.

3.3 Novel Proposed Method

We propose a novel framework based on (1) MDDs and (2) a request mechanism on the MDDs accessing the minimal or maximal value of an object attribute wrt a set of assumptions.

Firstly, we propose to use MDDs to encode attribute relations, constructing one MDD for each object of the problem. It enables the usage of existing procedures maintaining consistency over MDDs [30] to obtain efficient reasoning over object attributes. Secondly, we propose an interface such that MDDs can be embedded as parameters inside global constraints. It restricts each MDD considered to have a certain subset of variables called the *core*. Thirdly, we discuss explicit assumptions. Finally, those assumptions and the MDDs are used to define the request mechanism.

MDD Representation Considering an object o, we construct the corresponding MDD $\mathcal{M}_o$. Each attribute of o is represented by a variable in the CP model, corresponding to a layer in the MDD. A mode can then be equivalently encoded, either as a feasible *tuple* of attribute values, or as the corresponding r-t path in the MDD. See (1) of Example 4. Since the number of objects and the number of feasible modes per object are fixed problem parameters, the number of MDDs involved and their respective sizes (or number of r-t paths) remain bounded, making the use of exact MDDs practical.

Remarks Note that, in some cases, certain constraints are a by-product of object definitions and may result in inefficient representations. For the sake of generality, we consider in this work arithmetic constraints embedded in the MDD, although in practice it may be more computationally effective to handle some of them externally, especially when a variable domain contains many values. Problem-specific tuning may thus be necessary to ensure that the size of the MDD involved remains tractable.

Global Constraint Interface We propose the following interface

$$\text{MDD-GC}([o_1, o_2 \ldots, o_n], \mathit{Parameters}, \mathcal{T}) \tag{4}$$

with $\mathcal{T} = [\mathcal{M}_{o_1}, \mathcal{M}_{o_2}, \ldots, \mathcal{M}_{o_n}]$ the set of objects MDDs. This interface is generic, as it enables using the same MDD-GC in multiple variants of problems, as the side constraints are directly encoded through MDDs. Moreover, in order to be generic, the constraint must provide a *core*. This core is a common subset of attributes of all objects assigned by the constraint. It corresponds to a minimal subset of variables (*i.e.* levels in the MDD) that must be present for each object. See (2) of Example 4 for the MDD-Cumulative definition and its core.

Explicitly Annotated Algorithm Our method leverages the efficient global constraint algorithms that have been actively developed in the CP literature over the past years. They are augmented by explicitly annotating the assumptions made. We consider *unary assumptions*, which are *tuples* of the form $< Y, op, v >$ where Y is a variable, op is a binary relation $(\leq, <, =, \neq, >, \geq)$, and v is a value. The term *unary* refers to the fact that such an assumption can be checked in constant time. See Part (3) of Example 4.

Interactions Given a variable X, a multi-valued decision diagram $\mathcal{M}$ and a set of unary assumptions Φ, we define the function $\text{MINVALUE}(X, \mathcal{M}, \Phi)$ that returns the minimal value that X can take wrt its domain, MDD-based restrictions from $\mathcal{M}$ and all the assumptions in Φ. See (4). Clearly, $\mathcal{M}$ can only strengthen this minimal value if X is a variable that corresponds to a layer of $\mathcal{M}$. This observation is also valid for all variables Y that appear in the assumptions of Φ. Note that the assumptions in Φ may involve variable X directly as well as other variables $Y \neq X$. In the latter case, and whenever Y is not in $\mathcal{M}$, the assumption can be considered as *true* by default. The complexity of $\text{MINVALUE}(X, \mathcal{M}, \Phi)$ is linear wrt the size of $\mathcal{M}$. This function extends directly to $\text{MAXVALUE}(X, \mathcal{M}, \Phi)$ with $\text{MAXVALUE}(X, \mathcal{M}, \Phi) = -\text{MINVALUE}(-X, \mathcal{M}, \Phi)$ and returns the maximum value that X can take wrt Φ. We also abuse the notation in what follows by defining $\text{MINVALUE}(\mathcal{X}, \mathcal{M}, \Phi)$, which returns the minimum feasible values for all variables $X \in \mathcal{X}$ that correspond to a layer of $\mathcal{M}$.

From the perspective of the constraint, a call to the MDD is treated as any other operation (*i.e.* made in constant time); however, its computational cost strongly depends on the size of the MDD and is therefore likely to differ from one problem to another.

Remarks In general, filtering algorithms make more than one assumption. The reasoning obtained results from the conjunction of multiple assumptions detecting unfeasible variable assignments. However, the possible interdependencies between the variables at play require to carefully manage their processing order to guarantee consistent reasoning.

Example 4 (Continuation of Example 2).

(1) Figure 1a depicts the MDD $\mathcal{M}_a$ associated with task a. It is composed of four layers, each one corresponding to one of the four variables S_a, C_a, P_a and H_a, and encodes the two constraints $P_a = 4 \Leftrightarrow H_a = 2$ and $P_a = 6 \Leftrightarrow H_a = 1$ through its $r - t$ paths.
(2) The parameters of the CUMULATIVE constraint extended with MDDs $\mathcal{T}$ are

$$\text{MDD-CUMULATIVE}([o_1, o_2, \ldots, o_n], B, \mathcal{T})$$

with its corresponding core variables being $[S_i, P_i, C_i, H_i]$ for each object o_i.
(3) Consider the timetabling algorithm recalled in Section 2.2. Algorithm 1 presents the left filtering of a task i with annotations for each assumption

made. The procedure starts by assigning variable S_i to its lowest current feasible value denoted s^* (line 4). Then, it fixes P_i to its lower bound (line 5). Finally, the threshold test (line 6) sets H_i to $\underline{H_i}$ and updates s^* accordingly.

(4) Firstly, using those three assumptions and setting S_a to 0, a call to $\mathcal{M}_a$ reveals that task a cannot be processed wrt $P_a = \underline{P_a}$ and $H_a = \underline{H_a}$. Thus, not only rectangles between $\underline{S_a}$ and $\underline{P_a}$ will be considered but also those in the interval $[\underline{P_a}, \overline{P_a}]$. Recall that, firstly, only the first rectangle of the profile is checked (with the corresponding assumption $s^* + \underline{P_a} \leq 5$) with a height of 1. The value considered for $\underline{H_a}$ must be consistent with both assumptions $S_a = 0$ and $P_a \leq 5$. This value can be obtained directly using $h^* = \text{minValue}(H_a, \mathcal{M}_a, [S_a = 0, P_a \leq 5]) = 2$. However, as $h^* + 1 = 3 > B = 2$, it means that no feasible assignment for a exist, such that it only overlaps the first rectangle and respects the capacity. Thus, the second rectangle can be checked, *i.e.* $\underline{P_a} > 5 \wedge \underline{P_a} \leq \overline{P_a} = 6$. Again, the corresponding minimum value for H_a is given by $h^* = \text{minValue}(H_a, \mathcal{M}_a, [S_a = 0, P_a > 5, P_a \leq 6]) = 1$. As $h^* > 0$, it raises a resource conflict with the second block of the profile. Thus, reasoning can be made to update $\underline{S_a}$ to 7.

Algorithm 1 Cumulative: Left *Sweep* filtering algorithm for one task, from [25] and annotated.

Require: A resource capacity B, a profile TT_Ω and a task i: variables S_i, C_i, P_i and H_i
Ensure: Return **FAIL** if it is unfeasible; otherwise, updates bounds on S_i

1: compute j_1 such that $\underline{S_i} \in [a_{j_1}; b_{j_1}[$
2: compute j_2 such that $\overline{C_i} - 1 \in [a_{j_2}; b_{j_2}[$
3: $j \leftarrow j_1$
4: $s^* \leftarrow \underline{S_i}$ ▷ $\phi_1 : S_i = s^*$
5: **while** $j \leq j_2 \wedge a_j < \min(s^* + \underline{P_i}, \overline{S_i})$ **do** ▷ $\phi_2 : P_i = \underline{P_i}$
6: **if** $B - \underline{H_i} < D_j$ **then** ▷ $\phi_3 : H_i = \underline{H_i}$
7: $s^* \leftarrow \min(b_j, \overline{S_i})$
8: $j \leftarrow j + 1$
9: **if** $s^* > \underline{S_i}$ **then** $\underline{S_i} \leftarrow s^*$

4 MDD-Cumulatives and MDD-Multi-Cumulatives Constraints

We now use the method in two ways. Firstly, we apply it to the Cumulative constraint and describe the MDD-based timetabling algorithm for this constraint. Then, we expand Cumulative by considering an assignment variable, effectively generating a new version of timetabling that filters that assignment variable.

4.1 Timetabling Algorithm for the MDD-Cumulative

In this section we extend the basic timetabling algorithm, i.e. Algo. 1 presented in Section 2, in order to handle the MDD-Cumulative introduced in Example 4. We first define two functions that we will use in our extended algorithm Algo. 2.

Notations We introduce the function IndexRectangle(v) where v is a time point. This function returns the index j of the interval of the profile such that v belongs to it. In other words, $v \in [a_j, b_j[$. Let δ denote the difference between the threshold and the profile demand, and let δ_j denote the difference at rectangle j. δ is computed as the minimum over all δ_j for all j between the start and the current checked completion time. More precisely, if i has a mandatory part (*i.e.* $\overline{S_i} < \underline{C_i}$), δ does not consider $\underline{H_i}$. The MinGapOnTopOfProfile(j, k, i) function performs this computation; here, j and k are the first and last intervals taken into account, respectively, and i is the current task.

Intuition of the new filtering algorithm Alg. 2 To ensure validity wrt the feasible modes of a task, certain modifications are made. Given a task i, the idea is as follows. As duration (P_i) and demand (H_i) may depend on the value of S_i, we iterate over each interval of the profile within $\underline{S_i}$ and $\overline{S_i}$. For each interval j considered, we explicitly assume $S_i \in [a_j, b_j[$. Then, each interval corresponding to the last use of the resource, i.e. $C_i - 1$, wrt the assumption $S_i \in [a_j, b_j[$ must be tested. To do so, we iterate over each interval k and assume that $C_i - 1 \in [a_k, b_k[$. Finally, rather than getting the minimum value of H_i ($\underline{H_i}$), we compute its minimal value wrt those assumptions and the MDD $\mathcal{M}_i$; and check if it is valid. In other words, we check if it exists an assignment such that $S_i \in [a_j, b_j[$, $C_i - 1 \in [a_k, b_k[$ and $H_i \leq \delta$.

Description of Alg. 2 The overall **for** loop, lines 2–15, corresponds to the iterations over the feasible intervals of S_i. wrt S_i within rectangle j, bounds for P_i and C_i are computed in lines 3–6. Then, the inner **for** loop, lines 9–14, iterates over the feasible completion times of i. If a feasible assignment is found, we update, if necessary, the lower bound of S_i and exit with true, lines 12–14. Otherwise, we check the next feasible starting interval. After checking all feasible start intervals for S_i, we return false. We depict only the filtering for the start variables, since the same idea applies to the completion variables.

Validity and Worst-Case Complexity First, note that, for configurations without modes, a call to the MDD returns the only value of the domain. Therefore, the algorithm is equivalent to the one in [25]. Second, in the general case, once a fixed point has been reached, let $\underline{S_i} = s$ and j be the rectangle of the profile such that $s \in [a_j, b_j - 1]$. This implies that there must exist an assignment of task i wrt (*1*) $\mathcal{M}_i$ (modes of i) (*2*) the current profile (*3*) $S_i \in [s, b_j - 1]$. Moreover, this indicates that s is not necessarily a feasible value for S_i, as we only filter rectangle by rectangle. In other words, there may be inconsistent values in the domain of S_i as granularity depends on the profile.

Functions IndexRectangle and MinGapOnTopOfProfile may need to iterate over the entire profile. Thus, their complexity is in $O(n)$. In the worst case, every rectangle of the profile must be checked for S_i and again for C_i. Thus, the complexity is in $O(n^2)$. Finally, this algorithm is called for each task to filter the starting and completion time variables, resulting in an overall complexity of $O(n^3)$ for the MDD-Cumulative timetabling algorithm. Note that as introduced

in Section 3.3, as the size of $\mathcal{M}_i$ is considered bounded by the number of modes and the number of attributes (which is constant), a call to the MINVALUE function is considered to be either in constant time or in M_i, the number of modes of i. Formally, the overall complexity is thus bounded by $O(n^3 + n^2 \cdot \max_i M_i)$.

Algorithm 2 CUMULATIVE: Left *Sweep* algorithm with assumptions and call to the MDD. It is denoted SWEEP$(B, TT, \mathcal{X}_i, \mathcal{M}_i)$

Require: A resource capacity B, a profile TT, variables $\mathcal{X}_i$ and the MDD $\mathcal{M}_i$
Ensure: Return *FALSE* if it is unfeasible; otherwise, updates bounds on S_i

$j_1 \leftarrow$ INDEXRECTANGLE$(\underline{S_i})$; $j_2 \leftarrow$ INDEXRECTANGLE$(\overline{S_i})$; $s^* \leftarrow \underline{S_i}$;
for $j \in [j_1, j_2]$ **do** ▷ $\phi_1 : S_i \in [s^*, b_j - 1]$
  $p^*_{\min} \leftarrow$ MINVALUE$(P_i, \mathcal{M}_i, [S_i \geq s^*, S_i < b_j])$
  $p^*_{\max} \leftarrow$ MAXVALUE$(P_i, \mathcal{M}_i, [S_i \geq s^*, S_i < b_j])$
  $c^*_{\min} \leftarrow s^* + p^*_{\min}$ ▷ $\phi_2 : P_i \in [p^*_{min}, p^*_{\max}]$
  $c^*_{\max} \leftarrow \min(\overline{C}, \min(\overline{S}, b_j - 1) + p^*_{\max})$
  $k_1 \leftarrow$ INDEXRECTANGLE$(c^*_{\min} - 1)$; $k_2 \leftarrow$ INDEXRECTANGLE$(c^*_{\max} - 1)$;
  $\delta \leftarrow$ MINGAPONTOPOFPROFILE(j, k_1, i)
  for $k \in [k_1, k_2]$ **do**
    $h^* \leftarrow$ MINVALUE$\left(H_i, \mathcal{M}_i, \begin{bmatrix} S_i \geq s^*, S_i < b_j, \\ P_i \geq p^*_{\min}, P_i \leq p^*_{\max}, \\ C_i - 1 \geq a_k, C_i - 1 < b_k \end{bmatrix}\right)$
    $\delta \leftarrow \min(\delta, \delta_k)$
    if $h^* \leq \delta$ **then** ▷ $\phi_3 : H_i = h^*$
      if $s^* > \underline{S_i}$ **then** $\underline{S_i} \leftarrow s^*$
      return *TRUE*
  $s^* \leftarrow b_j$
return *FALSE*

4.2 The MDD-MULTI-CUMULATIVES Constraint

In order to benefit as much as possible from the method, we modify the MDD-CUMULATIVE to filter the assignment variable Y_i. Moreover, we also generalise the constraint to handle more than one resource at a time. For the sake of generality, a mode is defined by the value of the assignment variable Y_i and other restrictions between the variables of $\mathcal{X}_i$. Consider the multi-valued decision diagram $\mathcal{M}_i$ based on the set of variables $\langle Y_i, S_i, P_i, C_i, H_{ik} | k = 1, \ldots, R \rangle$ for task i.

We define a new constraint over the following interface:

$$\text{MDD-MULTI-CUMULATIVES}(\mathcal{X}, \mathcal{T}, B)$$

with $\mathcal{X} = \{\mathcal{X}_i | \forall i \in \mathcal{J}\}$ the set of tasks variables such that $\mathcal{X}_i = \{Y_i, S_i, P_i, C_i, \{H^r_{ik} | k = 1, \ldots, R\}\}$ and $\mathcal{T} = \{\mathcal{M}_i | \forall i \in \mathcal{J}\}$ the set of tasks MDDs. $B = \{B_k | \forall k \in \mathcal{K}\}$ denotes the resource thresholds vector. $\langle Y_i, S_i, P_i, C_i, H_{ik} | k = 1, \ldots, R \rangle$ is the core of the MDD-MULTI-CUMULATIVES constraint.

The constraint holds if the following conditions are true all together:

$$\forall i \in \mathcal{J} : S_i + P_i = C_i \tag{5}$$

$$\forall k \in \mathcal{K}, \forall t \in \mathbb{N} : \sum_{i|S_i \leq t < C_i} H_{ik} \leq B_k \tag{6}$$

$$\forall i \in \mathcal{J} : \exists P \in \mathcal{P}_{\mathcal{M}_i} : \forall X \in \mathcal{X}_i : X = x, x \in P \tag{7}$$

First (5) ensures the coherence of each task wrt its start, duration and completion time. Then (6) enforces that at each time point where at least one task is active, for each resource the threshold is respected. With (7), for each task, its mode (variable assignment) must respect its corresponding MDD. Due to the MDDs definitions, this constraint can enforce any mode configuration.

Consider a task i, for each variable in $\mathcal{X}_i$ the level of filtering made by the MDD-Multi-Cumulatives constraint is the following. The complete domain of Y_i is filtered. In other words, all values belonging to $\mathcal{D}(Y_i)$ are consistent with the constraint. On the other hand, for variables S_i and C_i, the same level of consistency is reached as for the MDD-Cumulative. In other words, consistency is enforced only wrt the interval of the profile. It may exist inconsistent values inside the domain (*i.e* $t \in [\underline{S_i}, \overline{S_i}]$).

Filtering Algorithm In the MDD-Cumulative, Algorithm 2 is called twice for each task. However, in the MDD-Multi-Cumulatives, it is called for each mode $m \in \mathcal{D}(Y_i)$ of each task i. At each call of $\mathcal{M}_i$, the assumption $Y_i = m$ is added. Moreover, whenever the algorithm returns *FALSE*, m is filtered. Note that, as the assumption $Y_i = m$ is added, there is no more guarantee that $c^*_{\min} \leq c^*_{\max}$ and thus, a test must be added after line 6.

We also consider multiple resources at the same time. The profile is built the same way; however, for each event, all resources are considered. For each of those resources, the event is mapped to a corresponding height. Even if the number of resources is higher than one, the overall profile stays the same with the same number of rectangles and events. Moreover, in practice, rectangle j is mapped to a set of corresponding resources for which a change appears at a_j and b_j in order to avoid iterating on resources with no demands.

The test made in line 12 must be performed over all resources. Thus the complexity becomes $O(n^2 \cdot R)$. Finally, for each task i, it is called $2 \cdot |\mathcal{D}(Y_i)|$. Let $M^{\max} = \max |\mathcal{D}(Y_i)|$ be the maximum number of assignments for each task. The overall complexity of MDD-Multi-Cumulative is $O(n^3 \cdot R \cdot M^{\max})$.

5 Experiments

Both constraints have been implemented in Java on open-source *ChocoSolver* [42]. Evaluations have been made on a computer with a *3.80 GHz Intel Core Ultra 7 165H CPU* and *32 GB of RAM*. The version used of the *ChocoSolver* is the *5.0.0-beta*. A time limit was fixed at 1800 seconds (30 minutes).

We have tested the MDD-Multi-Cumulatives on the PSPlib [31]. It is the state-of-the-art instances for the MM-RCPSP problem. In the MM-RCPSP, a mode must be assigned to each task and a Cumulative constraint must be respected for each renewable resource. Moreover, the problem also considers

precedences between tasks and non-renewable resources. We use as baseline a decomposed model (Dec) with a set of Element constraints ensuring mode consistency for each task and a Cumulative constraint for each renewable resource. In the second model (MC), they are replaced, respectively, by an MDDC constraint for each task and a unique Multi-Cumulatives for renewable resources.

Firstly, the (Dec) has been compared with both a CPOptimizer [33] and a MiniZinc [36] model. For space reasons, these results are not reported in detail here, but it is important to note that it is comparable with state-of-the-art methods. Letting the solver define the strategy itself (using heuristics and problem recognition) has led to a faster resolution. Secondly, both Choco models have been compared. Allowing the solver to define its own strategy (using heuristics and problem recognition) has led to a faster resolution. However, in order to compare efficiency, the search strategy must be fixed. It ensures that decisions are the same for both models, and the reasoning is correctly compared. Two classical search strategies have been implemented. A *dichomotic* search (denoted Dicho) which splits the makespan and tasks starting (*resp.* completion) time into two smaller windows [15] and one that creates mandatory parts for tasks (denoted MP). Fifty instances were tested for each benchmark with both strategies. Also note that in these benchmarks, tasks have only three possible modes.

Numerical results Observe the results in Table 1. In all cases (MC) solves and proves optimality for more instances. We also present detailed results for the instances solved by both models. Average time in seconds and number of nodes are reported. Finally, we present detailed results for the instances solved only by (MC). Note that all instances solved by (Dec) have also been solved by (MC). The same is true for the optimally solved instances. Time and nodes are only reported when an optimal solution has been found and proved.

Consider instances solved by both models. It can be observed that on average (Dec) is faster than (MC). However, (MC) always goes through fewer nodes to reach optimality, and all instances solved by (MC) where (Dec) times out are not taken into account. The time difference can be explained by (1) the longer procedure made by the MDD-Multi-Cumulatives constraint, and (2) the current implementation. For the latter case, it is made as proof of concept and thus, an overlay of the MDDs is built to make the implementation of the minValue function easier. However, it can be noted that (1) the search space is significantly reduced (more reasoning is obtained), (2) more instances are solved or optimally solved, and (3) there are certain cases where the time gap is almost leverage (*e.g.* benchmark R5 with dichotomic search). Finally, another test has been performed to compare filtering. It consists of letting (MC) solve the instance to optimality and retrieve the corresponding decision path. The domains are observed after applying this path to (Dec). It shows that, on average, more than 30% of the assignment variables are still not fixed. Both these results tend to show the efficiency of the novel constraint even over instances with a small number of modes per task.

Bench	Search	Dec		MC		Solved by both				Solved only by MC			
		Sol	Opt	Sol	Opt	Time		Node		Sol	Opt	Time	Node
						Dec	MC	Dec	MC				
R4	Dicho	24	23	42	39	**7.22**	90.79	470 197	74 491	14	10	194.55	176 128
	MP	28	27	43	41	**11.66**	58.75	474 260	40 746	15	14	251.99	84 470
R5	Dicho	22	22	46	34	**19.78**	**20.47**	1 001 989	16 263	24	12	209.75	85 031
	MP	26	25	38	37	**11.70**	98.73	449 005	109 978	12	12	95.27	105 787
C15	Dicho	30	30	43	39	**3.06**	18.02	182 824	17 422	13	9	90.27	46 076
	MP	33	32	41	40	**10.12**	22.43	945 754	31 714	8	8	71.15	58 903
J20	Dicho	30	28	41	35	**8.68**	24.68	76 928	24 281	11	7	163.01	86 682
	MP	32	29	40	35	**8.79**	84.47	383 766	112 538	8	6	337.78	91 698
M5	Dicho	30	30	44	40	**1.08**	13.66	93 404	10 436	14	10	252.25	243 661
	MP	33	33	38	38	**2.06**	17.29	104 784	8 802	5	5	270.97	44 184

Table 1: Results of the decomposed (Dec) and MDD-Multi-Cumulatives (MC) models over the PSPLib instances. Average time (in seconds) and number of nodes are reported over instances optimally solved.

Discussion The novel constraints can handle problems with more task configurations. To remain generic, mode values are checked wrt the starting time of the task. To show the effectiveness of the method while staying accessible, tests were conducted on PSPlib instances. In these benchmarks, modes depend solely on the assignment variable, where optional task representation can be very efficient. However, when considering dependencies with the execution time, the number of modes may become very large. In our proposed method, only the number of paths in the MDD increases, and those dependencies are directly taken into account by the constraint. Conversely, classical models generate an optional task for each of these modes. Therefore, it is expected that the method will be able to handle instances where classical models time out. This behaviour has been observed empirically in specific instances, and preliminary results from more general instances suggest that this expectation remains consistent.

6 Conclusion

The novel usage of MDDs presented here opens several avenues for future work.
Expanded Applications and Constraints The integration of MDDs could lead to more efficient reasoning within multiple constraints. In the packing domain, constraints like DiffN [7,35] and Geost [6,50] would be prime candidates. Other relevant constraints that consider objects with multiple attributes would be Cycles [13], LexChain [16] or StableKeysort [9].
Enhancements for Cumulative Constraints This work offers new insights into Cumulative constraints. The method could be enhanced for specific scheduling scenarios (e.g., problems with setup tasks [24]) by integrating specialised assumptions directly into the MDD structure. Moreover this work only considers the timetabling algorithm [25], but the MDD-Cumulative constraint could also benefit from integrating energetic reasoning [14] with MDDs. Ultimately, designing new algorithms based on the principle of integrated, tighter assumptions within MDDs could lead to even greater efficiency.

Acknowledgments

This publication has emanated from research conducted with the Tandem financial support of Region Pays de La Loire.

References

1. Aggoun, A., Beldiceanu, N.: Extending chip in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling **17**(7), 57–73 (1993). https://doi.org/https://doi.org/10.1016/0895-7177(93)90068-A, https://www.sciencedirect.com/science/article/pii/089571779390068A
2. Amilhastre, J., Fargier, H., Niveau, A., Pralet, C.: Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. International Journal on Artificial Intelligence Tools **23**(4), 1460015 (2014). https://doi.org/10.1142/S021821301460015X, https://www.worldscientific.com/doi/abs/10.1142/S021821301460015X
3. Andersen, H.R., Hadzic, T., Hooker, J.N., Tiedemann, P.: A constraint store based on multivalued decision diagrams. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming – CP 2007. pp. 118–132. Springer (2007). https://doi.org/10.1007/978-3-540-74970-7_11
4. Artigues, C.: The Resource-Constrained Project Scheduling Problem, chap. 1, pp. 19–35. John Wiley & Sons, Ltd (2008). https://doi.org/https://doi.org/10.1002/9780470611227.ch1, https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470611227.ch1
5. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling: applying constraint programming to scheduling problems, vol. 39. Springer Science & Business Media (2001)
6. Beldiceanu, N., Carlsson, M., Poder, E., Sadek, R., Truchet, C.: A generic geometrical constraint kernel in space and time for handling polymorphic k-dimensional objects. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming – CP 2007. pp. 180–194. Springer (2007). https://doi.org/10.1007/978-3-540-74970-7_15
7. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling **20**(12), 97–123 (1994). https://doi.org/10.1016/0895-7177(94)90127-9, https://www.sciencedirect.com/science/article/pii/0895717794901279
8. Beldiceanu, N., Carlsson, M.: A new multi-resource cumulatives constraint with negative heights. In: Van Hentenryck, P. (ed.) Principles and Practice of Constraint Programming - CP 2002. pp. 63–79. Springer (2002). https://doi.org/10.1007/3-540-46135-3_5
9. Beldiceanu, N., Carlsson, M., Flener, P., Lorca, X., Pearson, J., Petit, T., Prud'Homme, C.: A modelling pearl with sortedness constraints. In: Global conference on artificial intelligence. vol. 36, pp. 27–41 (2015)
10. Bellenguez-Morineau, O., Néron, E.: Multi-Mode and Multi-Skill Project Scheduling Problem, chap. 9, pp. 149–160. John Wiley & Sons, Ltd (2008). https://doi.org/https://doi.org/10.1002/9780470611227.ch9, https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470611227.ch9

11. Bergman, D., Cire, A.A., Hoeve, W.v.: MDD propagation for sequence constraints. Journal of Artificial Intelligence Research **50**, 697–722 (2014). https://doi.org/10.1613/jair.4199, https://www.jair.org/index.php/jair/article/view/10895
12. Bessière, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit complexity and decompositions of global constraints (2009), https://arxiv.org/abs/0905.3757
13. Bourreau, E.: Traitement de contraintes sur les graphes en programmation par contraintes. Ph.D. thesis, Paris 13 (1999)
14. Carlier, J., Pinson, E., Sahli, A., Jouglet, A.: An o(n2) algorithm for time-bound adjustments for the cumulative scheduling problem. European Journal of Operational Research **286**(2), 468–476 (2020). https://doi.org/https://doi.org/10.1016/j.ejor.2020.03.079, https://www.sciencedirect.com/science/article/pii/S037722172030312X
15. Carlier, J., Latapie, B.: Une méthode arborescente pour résoudre les problèmes cumulatifs. RAIRO-Operations Research **25**(3), 311–340 (1991)
16. Carlsson, M., Beldiceanu, N.: Arc-Consistency for a Chain of Lexicographic Ordering Constraints. Swedish Institute of Computer Science (2002), https://urn.kb.se/resolve?urn=urn:nbn:se:ri:diva-21992
17. Carvin, B.: Optimisation d'un ordonnancement de production de produits périssables dans un contexte multi-lignes. Ph.D. thesis, Ecole nationale supérieure Mines-Télécom Atlantique (2025)
18. Castro, M.P., Cire, A.A., Beck, J.C.: Decision diagrams for discrete optimization: A survey of recent advances. INFORMS Journal on Computing **34**(4), 2271–2295 (2022). https://doi.org/10.1287/ijoc.2022.1170, https://pubsonline.informs.org/doi/10.1287/ijoc.2022.1170
19. Cheng, K.C.K., Yap, R.H.C.: An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. Constraints **15**(2), 265–304 (2010). https://doi.org/10.1007/s10601-009-9087-y, https://doi.org/10.1007/s10601-009-9087-y
20. Cheng, K.C., Yap, R.H.: Maintaining generalized arc consistency on ad hoc r-ary constraints. In: International conference on principles and practice of constraint programming. pp. 509–523. Springer (2008)
21. CHIP team: CHIP, finite domain constraints reference manual. Tech. rep., COSYTEC (2003)
22. Cire, A., Van Hoeve, W.j.: MDD propagation for disjunctive scheduling. Proceedings of the International Conference on Automated Planning and Scheduling **22**, 11–19 (2012). https://doi.org/10.1609/icaps.v22i1.13521, https://ojs.aaai.org/index.php/ICAPS/article/view/13521
23. Cire, A.A., Van Hoeve, W.J.: Multivalued decision diagrams for sequencing problems. Operations Research **61**(6), 1411–1428 (2013). https://doi.org/10.1287/opre.2013.1221, https://pubsonline.informs.org/doi/10.1287/opre.2013.1221
24. Fanjul-Peyro, L.: Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. Expert Systems with Applications: X **5**, 100022 (2020). https://doi.org/10.1016/j.eswax.2020.100022, https://linkinghub.elsevier.com/retrieve/pii/S2590188520300019
25. Gay, S., Hartert, R., Schaus, P.: Simple and scalable time-table filtering for the cumulative constraint. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming. pp. 149–157. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-23219-5_11

26. Gedik, R., Kalathia, D., Egilmez, G., Kirac, E.: A constraint programming approach for solving unrelated parallel machine scheduling problem. Computers & Industrial Engineering **121**, 139–149 (2018). https://doi.org/10.1016/j.cie.2018.05.014, https://www.sciencedirect.com/science/article/pii/S0360835218302158
27. Gentzel, R., Michel, L., van Hoeve, W.J.: HADDOCK: A language and architecture for decision diagram compilation. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming. pp. 531–547. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-58475-7_31
28. Hentenryck, P.V., Carillon, J.P.: Generality versus specificity: an experience with ai and or techniques. In: Proceedings of the Seventh AAAI National Conference on Artificial Intelligence. p. 660–664. AAAI'88, AAAI Press (1988)
29. Hill, A., Ticktin, J., Vossen, T.W.M.: A computational study of constraint programming approaches for resource-constrained project scheduling with autonomous learning effects. In: Stuckey, P.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 26–44. Springer International Publishing, Cham (2021)
30. Hoda, S., van Hoeve, W.J., Hooker, J.N.: A systematic approach to MDD-based constraint programming. In: Cohen, D. (ed.) Principles and Practice of Constraint Programming – CP 2010. pp. 266–280. Springer (2010). https://doi.org/10.1007/978-3-642-15396-9_23
31. Kolisch, R., Sprecher, A.: Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program. European Journal of Operational Research **96**(1), 205–216 (1997). https://doi.org/https://doi.org/10.1016/S0377-2217(96)00170-1, https://www.sciencedirect.com/science/article/pii/S0377221796001701
32. Laborie, P., Rogerie, J.: Reasoning with conditional time-intervals. In: FLAIRS. pp. 555–560 (2008)
33. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: Ibm ilog cp optimizer for scheduling: 20+ years of scheduling with constraints at ibm/ilog. Constraints **23**(2), 210–250 (2018)
34. Lahrichi, A.: Scheduling: the Notions of Hump, Compulsory Parts and their Use in Cumulative Problems. C.R. Acad. Sci., Paris **294**, 209–211 (February 1982)
35. Mirza, D.: Investigating diffn constraints in lazy clause generation solvers (2025)
36. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: International conference on principles and practice of constraint programming. pp. 529–543. Springer (2007)
37. Okubo, H., Miyamoto, T., Yoshida, S., Mori, K., Kitamura, S., Izui, Y.: Project scheduling under partially renewable resources and resource consumption during setup operations. Computers & Industrial Engineering **83**, 91–99 (2015). https://doi.org/10.1016/j.cie.2015.02.006, https://www.sciencedirect.com/science/article/pii/S0360835215000637
38. Patt-Shamir, B., Rawitz, D.: Vector bin packing with multiple-choice. Discrete Applied Mathematics **160**(10-11), 1591–1600 (2012)
39. Perez, G.: Decision diagrams: constraints and algorithms. Ph.D. thesis, COMUE Université Côte d'Azur (2015-2019) (2017)
40. Perez, G., Régin, J.C.: Improving gac-4 for table and mdd constraints. In: International Conference on Principles and Practice of Constraint Programming. pp. 606–621. Springer (2014)

41. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: International conference on principles and practice of constraint programming. pp. 482–495. Springer (2004)
42. Prud'homme, C., Fages, J.G.: Choco-solver: A java library for constraint programming. Journal of Open Source Software **7**(78), 4708 (2022). https://doi.org/10.21105/joss.04708, https://doi.org/10.21105/joss.04708
43. Rice, M., Kulhari, S.: A survey of static variable ordering heuristics for efficient bdd/mdd construction. University of California, Tech. Rep **130** (2008)
44. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the tenth annual ACM symposium on Theory of computing. pp. 216–226 (1978)
45. Schnell, A., Hartl, R.F.: On the efficient modeling and solution of the multi-mode resource-constrained project scheduling problem with generalized precedence relations. OR Spectrum **38**(2), 283–303 (2016). https://doi.org/10.1007/s00291-015-0419-6, https://doi.org/10.1007/s00291-015-0419-6
46. Schnell, A., Hartl, R.F.: On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs. Operations Research Perspectives **4**, 1–11 (2017). https://doi.org/https://doi.org/10.1016/j.orp.2017.01.002, https://www.sciencedirect.com/science/article/pii/S2214716016300458
47. Schnell, A., Hartl, R.F.: On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs. Operations Research Perspectives **4**, 1–11 (2017). https://doi.org/10.1016/j.orp.2017.01.002, https://linkinghub.elsevier.com/retrieve/pii/S2214716016300458
48. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints **16**, 250–282 (2011)
49. Tesch, A.: Improving energetic propagations for cumulative scheduling. In: International conference on principles and practice of constraint programming. pp. 629–645. Springer (2018)
50. Ågren, M., Beldiceanu, N., Carlsson, M., Sbihi, M., Truchet, C., Zampelli, S.: Six ways of integrating symmetries within non-overlapping constraints. In: van Hoeve, W.J., Hooker, J.N. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 11–25. Springer (2009). https://doi.org/10.1007/978-3-642-01929-6_3