Noname manuscript No.
(will be inserted by the editor)

Nutmeg: A MIP and CP Hybrid Solver Using Branch-and-Check

Edward Lam · Graeme Gange · Peter J. Stuckey · Pascal Van Hentenryck · Jip J. Dekker

Received: date / Accepted: date

Abstract This paper describes the implementation of Nutmeg, a solver that hybridizes mixed integer linear programming and constraint programming using the branch-and-cut style of logic-based Benders decomposition known as branch-and-check. Given a high-level constraint programming model, Nutmeg automatically derives a mixed integer programming master problem that omits global constraints with weak linear relaxations, and a constraint programming subproblem identical to the original model. At every node in the branch-and-bound search tree, the linear relaxation computes dual bounds and proposes solutions, which are checked for feasibility of the omitted constraints in the constraint programming subproblem. In the case of infeasibility, conflict analysis generates Benders cuts, which are appended to the linear relaxation to cut off the candidate solution. Experimental results show that Nutmeg's automatic decomposition outperforms pure constraint programming and pure mixed integer programming on problems known to have successful implementations of logic-based Benders decomposition, but performs poorly on general problems, which lack specific decomposable structure. Nonetheless, Nutmeg outperforms the standalone approaches on one problem with no known decomposable structure, providing preliminary indications that a hand-tailored decomposition for this problem could be worthwhile. On the whole, Nutmeg serves as a valuable tool for novice modelers to try hybrid solving and for expert modelers to quickly compare different logic-based Benders decompositions of their problems.

Edward Lam · Graeme Gange · Peter J. Stuckey · Jip J. Dekker
Department of Data Science and Artificial Intelligence, Faculty of Information Technology, Monash University
Melbourne, Victoria, Australia
E-mail: {edward.lam, graeme.gange, peter.stuckey, jip.dekker}@monash.edu

Pascal Van Hentenryck
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology
Atlanta, Georgia, United States of America
E-mail: pascal.vanhentenryck@isye.gatech.edu

Keywords mixed integer programming · constraint programming · hybridization · branch-and-check · logic-based Benders decomposition · conflict analysis · nogood · lazy clause generation · linear constraints · dual bound

1 Introduction

Combinatorial optimization problems are abstract mathematical problems with immense practicality. Therefore, solving them quickly is of interest to both academics and practitioners. Combinatorial optimization problems can be solved using any one of a variety of methods, including dynamic programming, Boolean satisfiability (SAT), mixed integer programming (MIP) and constraint programming (CP), to name just a few. Even though these methods have exponential worst-case time complexity, each has its own strengths and weaknesses, making some methods faster than others at particular problems. Usually, the speed difference arises from a method's ability or inability to exploit certain substructures within a problem. This paper studies a hybridization of MIP and CP that decomposes substructures to either MIP or CP in order to exploit their unique strengths and alleviate their weaknesses.

To solve a problem using either MIP or CP, the problem must be formally stated as a *model*. A model consists of unknowns, called *variables*, and relationships between the variables, called *constraints*. In an *optimization problem*, as opposed to a *satisfaction problem*, the model also contains an *objective function*, which computes a number to be minimized or maximized, called the *objective value*. The problem can then be solved by calling a *solver*, an implementation of an algorithm, on the model.

CP solvers typically run a tree search algorithm. At every node of the search tree, the solver maintains a set of values that each variable can take, called its *domain*, and calls a sequence of subroutines, called *propagators*, to remove inconsistent values from the domains. A solution is found whenever the domain of every variable is reduced to a singleton. Effective CP models typically use *global constraints* for declaring high-level substructures, such as network flow, bin packing or disjunctive scheduling. A major strength of CP is that solvers often implement specialized propagators for reasoning over the entire substructure of a global constraint, allowing domains to be reduced very quickly. The canonical example is cumulative scheduling, for which CP is state-of-the-art and has closed many difficult benchmarks [31].

MIP solvers also search a tree but take a different approach. At every node of the search tree, MIP solvers solve a *relaxation*; usually, but not always, a linear programming (LP) relaxation. A relaxation of a problem is identical to the originating problem but omits some of its constraints. Consequently, the optimal objective value of a relaxation provides a dual bound to the originating problem. This value also bounds the optimal objective value of all nodes in the subtree below the node. High-performance MIP models have a tight relaxation, i.e., one whose convex hull is close to its integer hull. Some models (e.g., those with a totally unimodular matrix and integer right-hand side) can be

theoretically proven that their LP relaxation is as tight as possible, i.e., its convex hull and integer hull coincide [10]. A consequence is that the MIP model is solved in one call to an LP solver, and hence, the problem can be solved in polynomial time.

Despite CP's impressive ability to reason within individual substructures using global constraints, reasoning across multiple substructures/constraints remains a challenge because each constraint has no knowledge about every other constraint. CP has considerable difficulty at reasoning across different constraints to optimize a linear objective function because it has no global view of the problem and because propagation of linear constraints is weak. In particular, CP performs especially poorly if many constraints are linear, i.e., the problem is or nearly is a pure MIP problem. Contrastingly, MIP problems are stated on a matrix, and solving its LP relaxation is equivalent to performing elementary row operations, essentially allowing for communication across constraints. Notably, reasoning over a polytope to optimize a linear objective function is trivial for LP.

Propagation and relaxation are complementary, and fully exploiting this complementarity should lead to speed improvements. Logic-based Benders decomposition (LBBD) formalizes this approach to hybridization [21]. Successful implementations of LBBD typically use CP to reason about individual substructures and MIP to aggregate this information across various substructures using linear constraints; also gaining the dual bound in the process.

A major disadvantage of LBBD in the past is that it is always problem-specific. Recently, two teams independently invented a generic form of LBBD, enabling LBBD of arbitrary problems with little effort [11,25]. The present paper, written by the two teams together, further develops these ideas into a publicly-available solver named Nutmeg. Given a high-level model of a problem, Nutmeg automatically builds a MIP master problem and a CP checking subproblem using the LBBD framework. Nutmeg then proceeds to solve the two problems side-by-side in a single branch-and-bound search tree. The MIP master problem optimizes a linear objective function while the CP checking subproblem propagates global constraints to reduce the variable domains. In addition, dual bounds in the MIP master problem are communicated to the CP subproblem for propagation, and infeasibilities in the CP subproblem are transferred to the MIP master problem as Benders cuts. The remainder of this paper details the method and empirically compares it against pure CP and pure MIP approaches on a large range of problems.

2 Related Work

Hybridization of MIP and CP is a highly active area of research. SCIP is a MIP solver that supports many features foundational to CP, such as global constraints, propagation and conflict analysis [2,1]. Even though SCIP natively supports global constraints, the key difference is that Nutmeg implements a decomposition; it does not simply extend a MIP solver with propagation and

conflict analysis (as in SCIP). Rather, Nutmeg moves complicating variables and constraints out of a monolithic model into a subproblem, in the hope that the master problem is smaller and can be solved faster.

The use of MIP methods in CP also has a long history. LP propagators have long been used within CP solvers [32] in order to better reason about conjunctions of linear constraints. Similarly, network flow algorithms [30] and general network flow propagators [33] have been used extensively in CP. However, the generic use of MIP techniques to find dual bounds is limited. Problem-specific CP solvers have implemented dual bounds using a variety of methods, including global optimization constraints [13], Lagrangian relaxation [14,6], Dantzig-Wolfe reformulation [22] and LBBD [21]. A summary of these techniques can be found in the short survey in [23].

The two main goals of Nutmeg are to provide better reasoning about a conjunction of linear constraints (i.e., a polytope) in a CP solver and to find tighter dual bounds via a linear relaxation. This is accomplished using LBBD. Classical Benders decomposition splits a monolithic MIP model into a MIP master problem and an LP subproblem, both of which contain different variables and constraints [7]. The method then iteratively solves the two problems. The master problem proposes a candidate solution. The subproblem checks the solution and communicates infeasibility or superoptimality to the master problem by adding rows, now known as Benders feasibility cuts and Benders optimality cuts respectively [34]. Benders cuts are found using LP duality theory, which has no equivalent for MIP subproblems. LBBD expands upon LP duality by defining an inference dual to any general optimization problem [21], enabling MIP subproblems or even general discrete subproblems. Unlike classical Benders decomposition, there is no exact form that applies to every problem due to its generality. Every implementation of LBBD requires the user to analyze the problem and precisely define the Benders cuts added to the master problem.

An automatic mechanism to generate Benders cuts applicable to all CP problems was independently invented by [11] and [25]. The method relies on conflict analysis of lazy clause generation CP solvers, which themselves are hybrids of SAT and traditional finite-domain CP solvers [28,12]. Conflict analysis generates a clause (i.e., a disjunction of Boolean variables) explaining an infeasibility. The clause is then translated into a Benders cut in the MIP master problem.

In [11], the method is implemented by adapting the MiniZinc modeling system [27]. The MiniZinc controller iterates between a MIP solver and a CP solver. It finds an optimal MIP solution, which is passed to the CP solver to check. Their results showed that the hybrid method outperformed pure MIP and pure CP models on a variety of problems with known successful LBBD implementations.

In [25], a specialized branch-and-cut solver is built for the Vehicle Routing Problem with Time Windows. It passes every LP solution in the branch-and-bound tree to the CP solver. Borrowing terminology from earlier works [35,4], the algorithm was named branch-and-check with explanations. The solver found

cuts identical to problem-specific cuts previously proposed in the literature. Consequently, these cuts are lifted using existing techniques by recognizing the form of the cuts. The solver also found problem-specific cuts that have never before appeared in the literature.

Even though the results of [11, 25] are peer-reviewed, their codes are not intended for general use. The solver of [25] is problem-specific, and the system from [11] requires hand-tuning for each problem. The present paper, authored by both teams, develops these concepts into a fully-working solver available for public use. This paper does not present significant advances in the theory of hybridization, but rather, describes a new implementation in the definitive journal version of the earlier conference works.

3 Preliminaries

This section presents several concepts necessary for the discourse in the remainder of the paper.

3.1 Logic-based Benders Decomposition

LBBD is a method for solving combinatorial problems with substructures suited to MIP and substructures suited to CP. It is defined using an inference dual in the general case, but this paper only introduces a special case relevant to hybridizing MIP and CP.

An assumption of many CP approaches is that every variable takes integer values and has a finite domain. Hence, CP models are bounded. Consider a general bounded problem $\mathcal{P}$ that has variables partitioned into two groups $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_m)$, each with finite domains $D_{\mathbf{x}} \subseteq \mathbb{Z}^n$ and $D_{\mathbf{y}} \subseteq \mathbb{Z}^m$. Without loss of generality, the problem minimizes x_1 over the intersection of some linear constraints $A_1\mathbf{x} \leq \mathbf{b}_1$ on the $\mathbf{x}$ variables and some general constraints $S \subseteq D_{\mathbf{x}} \times D_{\mathbf{y}}$ on both the $\mathbf{x}$ and $\mathbf{y}$ variables. For example, S can contain global constraints or channeling constraints between multiple modelings. The problem $\mathcal{P}$ can be stated as follows:

$$\begin{aligned} \min \quad & x_1 \\ \text{subject to} \quad & A_1\mathbf{x} \leq \mathbf{b}_1, \\ & (\mathbf{x}, \mathbf{y}) \in S, \\ & \mathbf{x} \in D_{\mathbf{x}}, \\ & \mathbf{y} \in D_{\mathbf{y}}. \end{aligned}$$

Define the MIP master problem $\mathcal{M}$ of $\mathcal{P}$ as:

$$\begin{aligned} \min \quad & x_1 \\ \text{subject to} \quad & A_1\mathbf{x} \leq \mathbf{b}_1, \\ & A_2\mathbf{x} \leq \mathbf{b}_2, \\ & \mathbf{x} \in D_{\mathbf{x}}, \end{aligned}$$

1. **MIP master problem:** Solve $\mathcal{M}$ to optimality. If it is infeasible, report infeasibility and exit. Otherwise, retrieve its solution $\hat{\mathbf{x}}$.
2. **CP checking subproblem:** Temporarily fix $\mathbf{x}$ to $\hat{\mathbf{x}}$ in $\mathcal{C}$. Solve $\mathcal{C}$. If it is infeasible, create one or more of Constraint (1), add them to $\mathcal{M}$ and go back to Step 1.
3. **Optimal:** The problem is solved to optimality. Return the solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ from $\mathcal{C}$.

Fig. 1 The basic LBBD algorithm.

where $\{\mathbf{x} \in D_{\mathbf{x}} | \mathbf{y} \in D_{\mathbf{y}}, (\mathbf{x}, \mathbf{y}) \in S\} \subseteq \{\mathbf{x} \in D_{\mathbf{x}} | A_2\mathbf{x} \leq \mathbf{b}_2\}$. Then, the constraints $A_2\mathbf{x} \leq \mathbf{b}_2$ are said to be a *relaxation* of S. Since $D_{\mathbf{x}}$ is finite, $\mathcal{M}$ is bounded. Clearly, $\mathcal{M}$ is a MIP relaxation of $\mathcal{P}$. Next, define the CP checking subproblem $\mathcal{C}$ of $\mathcal{P}$ as the satisfiability version of $\mathcal{P}$:

$$\begin{aligned} \min \quad & 0 \\ \text{subject to} \quad & A_1\mathbf{x} \leq \mathbf{b}_1, \\ & (\mathbf{x}, \mathbf{y}) \in S, \\ & \mathbf{x} \in D_{\mathbf{x}}, \\ & \mathbf{y} \in D_{\mathbf{y}}. \end{aligned}$$

Notice that $A_1\mathbf{x} \leq \mathbf{b}_1$ also appears in $\mathcal{C}$, even though it already exists in $\mathcal{M}$. As explained later in Section 4.3, including it in $\mathcal{C}$ can lead to more propagation when dealing with fractional solutions from the LP relaxation of $\mathcal{M}$.

In some sense, this decomposition delegates the optimization parts to the MIP problem and the satisfiability parts to the CP problem. Such a decomposition – one that includes all the original constraints in the CP subproblem – is rare in the literature.

The LBBD methodology iterates between the master problem $\mathcal{M}$ and the subproblem $\mathcal{C}$, but exactly when to solve them is a choice in the implementation. The basic LBBD algorithm is summarized in Figure 1. (Nutmeg implements the branch-and-cut form of LBBD, called branch-and-check, as described later.) Step 1 solves the MIP master problem optimally. If it is infeasible, the algorithm immediately exits. Otherwise, an $\mathcal{M}$-feasible solution $\hat{\mathbf{x}}$ exists since the problem is bounded. In Step 2, $\hat{\mathbf{x}}$ is fed into the CP checking subproblem to check for feasibility of the omitted constraints S. If the subproblem finds that $\hat{\mathbf{x}}$ is infeasible with respect to S, one or more linear constraints

$$A_{\hat{\mathbf{x}}}\mathbf{x} \leq \mathbf{b}_{\hat{\mathbf{x}}} \tag{1}$$

are added to the master problem to remove $\hat{\mathbf{x}}$. The LBBD algorithm then solves the master problem again, iterating between the two problems until the subproblem declares that $\hat{\mathbf{x}}$ is feasible. The first feasible solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ in the checking subproblem is optimal for the original problem $\mathcal{P}$.

As in standard CP models, the objective function is equated to a variable. Hence, there is no need to distinguish between feasibility cuts and optimality cuts. Constraint (1) is simply referred to as a Benders cut. Unlike classical Benders decomposition, a general template of Constraint (1) does not exist

in LBBD; its exact form is problem-specific. The remainder of this section presents an example.

Example 1 (Cumulative Scheduling with Optional Tasks) The global constraint CumulativeOptional optionally schedules tasks on machines. Machines can be thought of as replenishing resources. Given a set $\mathcal{I} = \{1, \ldots, I\}$ of $I \in \mathbb{Z}_+$ tasks, the constraint CumulativeOptional$(\mathbf{x}, \mathbf{s}, \mathbf{d}, \mathbf{r}, C)$ takes a vector $\mathbf{x} = (x_1, \ldots, x_I) \in \{0,1\}^I$ of binary variables indicating whether a task is scheduled or ignored, a vector $\mathbf{s} = (s_1, \ldots, s_I) \in \mathbb{Z}_+^I$ of integer variables representing the starting time of the active tasks (i.e., every task $i \in \mathcal{I}$ with $x_i = 1$), a constant vector $\mathbf{d} = (d_1, \ldots, d_I) \in \mathbb{Z}_+^I$ of the duration of each task, a constant vector $\mathbf{r} = (r_1, \ldots, r_I) \in \mathbb{Z}_+^I$ for the number of machines simultaneously required for each task, and a scalar $C \in \mathbb{Z}_+$ representing the total number of machines available. The constraint attempts to schedule the tasks $\{i \in \mathcal{I} | x_i = 1\}$ on C machines, where each task i uses r_i machines from time s_i to time $s_i + d_i - 1$ (inclusive).

Consider this constraint as S in the definition of $\mathcal{P}$ and $\mathcal{C}$. A possible linear relaxation of S is the constraint

$$\sum_{i \in \mathcal{I}} r_i \cdot d_i \cdot x_i \leq C \cdot T,$$

where $T = \max_{i \in \mathcal{I}}(\max(D_{s_i}) + d_i - 1)$ is the time before which all tasks must be completed and D_{s_i} is the domain of s_i. This constraint is known as the *energy relaxation* because it reasons about the "area under the curve". Figure 2 illustrates this relaxation for a CumulativeOptional constraint with three machines and two tasks. Each task $i \in \{1, 2\}$ requires $r_i = 2$ machines, has a duration of $d_i = 3$ and must start between time 1 and 2, i.e., $s_i \in \{1, 2\}$. Then, the time horizon is $T = 4$. The full constraint, shown on the left, permits at most one task to be scheduled. The energy relaxation, shown on the right, only bounds the total number of squares in use.

Example 2 (Planning and Scheduling) The Planning and Scheduling problem assigns jobs to facilities and then schedules the jobs at each facility on a number of machines [19,20]. Let $T \in \mathbb{Z}_+$ be the time horizon before which all jobs must be completed. Define $\mathcal{J} = \{1, \ldots, J\}$ as the set of jobs and $\mathcal{F} = \{1, \ldots, F\}$ as the set of facilities. Let $x_{j,f} \in \{0, 1\}$ be a binary decision variable indicating whether job $j \in \mathcal{J}$ is assigned to facility $f \in \mathcal{F}$, and let $s_{j,f} \in \mathbb{Z}_+$ be an integer decision variable for the start time of job j when j is assigned to facility f, i.e., when $x_{j,f} = 1$. If j is assigned to f, it incurs a cost $c_{j,f} \in \mathbb{Z}_+$, has a duration of $d_{j,f} \in \mathbb{Z}_+$ units of time, uses $r_{j,f} \in \mathbb{Z}_+$ machines at f simultaneously, and must start within some time window $\{a_{j,f}, \ldots, b_{j,f}\} \subseteq \{1, \ldots, T - d_{j,f} + 1\}$. Each facility f has a total of $C_f \in \mathbb{Z}_+$ machines. The problem minimizes the total cost of assigning jobs to facilities.

The problem $\mathcal{P}$ is stated in Figure 3. Each facility $f \in \mathcal{F}$ is associated with one CumulativeOptional constraint that schedules all jobs assigned to f, i.e., the jobs $\{j \in \mathcal{J} | x_{j,f} = 1\}$. The MIP master problem $\mathcal{M}$ is shown

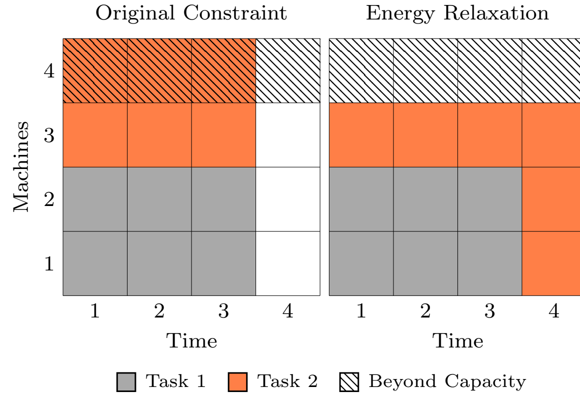

Fig. 2 An example of the energy relaxation of the CumulativeOptional constraint.

$$\min z \tag{2a}$$

subject to

$$z = \sum_{j \in \mathcal{J}} \sum_{f \in \mathcal{F}} c_{j,f} \cdot x_{j,f} \tag{2b}$$

$$\sum_{f \in \mathcal{F}} x_{j,f} = 1 \qquad \forall j \in \mathcal{J}, \tag{2c}$$

$$\text{CumulativeOptional}((x_{j,f} | j \in \mathcal{J}), (s_{j,f} | j \in \mathcal{J}), (d_{j,f} | j \in \mathcal{J}), (r_{j,f} | j \in \mathcal{J}), C_f) \qquad \forall f \in \mathcal{F}, \tag{2d}$$

$$x_{j,f} \in \{0, 1\} \qquad \forall j \in \mathcal{J}, f \in \mathcal{F}, \tag{2e}$$

$$s_{j,f} \in \{a_{j,f}, \ldots, b_{j,f}\} \qquad \forall j \in \mathcal{J}, f \in \mathcal{F}. \tag{2f}$$

Fig. 3 The Planning and Scheduling problem.

$$\min z \tag{3a}$$

subject to

$$z = \sum_{j \in \mathcal{J}} \sum_{f \in \mathcal{F}} c_{j,f} \cdot x_{j,f} \tag{3b}$$

$$\sum_{f \in \mathcal{F}} x_{j,f} = 1 \qquad \forall j \in \mathcal{J}, \tag{3c}$$

$$\sum_{j \in \mathcal{J}} r_{j,f} \cdot d_{j,f} \cdot x_{j,f} \leq C_f \cdot T \qquad \forall f \in \mathcal{F}, \tag{3d}$$

$$x_{j,f} \in \{0, 1\} \qquad \forall j \in \mathcal{J}, f \in \mathcal{F}. \tag{3e}$$

Fig. 4 The MIP master problem in an LBBD of the Planning and Scheduling problem.

in Figure 4. Constraint (3d) is the energy relaxation of Constraint (2d). The CP checking subproblem is exactly the original problem without the objective function, and hence, is not shown here.

For any candidate solution $\hat{\mathbf{x}}$ in the MIP master problem, define $1_{\hat{\mathbf{x}}} = \{(j, f) | \hat{x}_{j,f} = 1\}$ as the set of job-facility pairs taking value 1. Whenever the CP subproblem detects that $\hat{\mathbf{x}}$ is infeasible, Constraint (1) is realized as the inequality

$$\sum_{(j,f)\in 1_{\hat{\mathbf{x}}}} x_{j,f} \leq |1_{\hat{\mathbf{x}}}| - 1. \tag{4}$$

This Benders cut forces the MIP solver to choose another set of assignments by prohibiting at least one of the selected assignments.

3.2 Bound Disjunction Constraints

Benders cuts over binary variables are simple to add to a MIP model (e.g., Constraint (4)). Nutmeg supports Benders cuts over integer variables, which are non-trivial to capture in MIP. Benders cuts over integer variables can be implemented using bound disjunction constraints, which generalize a disjunction of binary variables (i.e., a clause) to a disjunction of bound tightenings of integer variables.

For any integer variable $x \in \mathbb{Z}$, define a binary variable named $[\![x = k]\!]$ that takes value 1 if and only if x takes value $k \in \mathbb{Z}$ in the same solution and takes value 0 otherwise. Similarly, define binary variables $[\![x \neq k]\!]$, $[\![x \geq k]\!]$ and $[\![x \leq k]\!]$ that respectively indicate whether $x \neq k$, $x \geq k$ and $x \leq k$. These binary indicator variables are called *literals*. Notice that $[\![x \neq k]\!] = 1 - [\![x = k]\!]$ and $[\![x \leq k]\!] = 1 - [\![x \geq k + 1]\!]$. Therefore, only some of these literals need to be considered explicitly.

For a problem with $I \in \mathbb{Z}_+$ integer variables $x_1, \ldots, x_I$, a bound disjunction constraint is a disjunction of literals of the form:

$$\bigvee_{(i,k)\in L_\geq} [\![x_i \geq k]\!] \ \vee \bigvee_{(i,k)\in L_\leq} [\![x_i \leq k]\!], \tag{5}$$

where $L_\geq, L_\leq \subseteq \{1, \ldots, I\} \times \mathbb{Z}$ are sets of pairs of variable indices and values that specify the literals of the constraint. If all the x_i variables in Constraint (5) are binary, the constraint can be trivially linearized. Otherwise, there are three common ways of implementing a bound disjunction constraint in MIP.

Propagation Constraint (5) can simply be propagated exactly like in CP. Upon selecting the next node to solve in the branch-and-bound tree, the node is preprocessed to remove inconsistent values in the variable domains. Even though the propagator is domain consistent, it does not filter the domains much since it is an extremely weak constraint. The only propagation that occurs is that whenever all but one of the literals are assigned 0, the remaining literal is fixed to 1, satisfying the constraint; and whenever all literals are assigned 0,

infeasibility of the constraint is detected. Since the constraint does not appear in the LP relaxation, this approach suffers from many of the same weaknesses as CP. For example, the constraint could be infeasible in the LP relaxation (called *rationally infeasible*), but this fact cannot be detected because propagation only reasons about one constraint at a time.

Indicator Constraints Constraint (5) can be explicitly added to the model as:

$$\sum_{(i,k)\in L_{\geq}} [\![x_i \geq k]\!] + \sum_{(i,k)\in L_{\leq}} [\![x_i \leq k]\!] \geq 1. \tag{6}$$

The literals $[\![x_i \geq k]\!]$ and $[\![x_i \leq k]\!]$ must also appear in the model. They are added as binary variables, together with the indicator constraints:

$$\begin{aligned}
&[\![x_i \geq k]\!] = 1 \rightarrow x_i \geq k, \\
&[\![x_i \geq k]\!] = 0 \rightarrow x_i \leq k-1, \\
&[\![x_i \leq k]\!] = 1 \rightarrow x_i \leq k, \\
&[\![x_i \leq k]\!] = 0 \rightarrow x_i \geq k+1.
\end{aligned}$$

Indicator constraints themselves can be implemented using either a big-M constraint or an inequality and an SOS1 constraint. Indicator constraints are well-established in modern MIP solvers, so we do not discuss them in detail.

Linking Constraints Under this approach, the entire unary encoding of the domain D_{x_i} of every integer variable x_i in Constraint (6) is added as equality literals, along with constraints that link the literals to the originating integer variable:

$$\begin{aligned}
x_i &= \sum_{k\in D_{x_i}} k \cdot [\![x_i = k]\!], \\
\sum_{k\in D_{x_i}} &[\![x_i = k]\!] = 1.
\end{aligned}$$

Constraint (6) is added to the model after substituting:

$$\begin{aligned}
[\![x_i \leq k]\!] &= \sum_{j\in D_{x_i}\cap\{-\infty,\ldots,k\}} [\![x_i = j]\!], \\
[\![x_i \geq k]\!] &= \sum_{j\in D_{x_i}\cap\{k,\ldots,\infty\}} [\![x_i = j]\!].
\end{aligned}$$

This approach makes the matrix much denser and explodes the size of the matrix with many more auxiliary variables. However, for integer variables with small domains, linking constraints are faster than indicator constraints in practice [5, 29].

3.3 Conflict Analysis

In all modern SAT solvers, every change to a domain is recorded in a graph called the *implication graph*. Whenever a propagation is infeasible, the search algorithm calls a subroutine, called *conflict analysis*, that inspects the chain of propagations that led to the infeasibility and then creates a constraint, called a *nogood*, that prevents the infeasibility from occurring again in the remainder of the search tree [26]. This is because any subtree that contains the same propagations will always violate the nogood, and hence, the entire subtree can be discarded. Conflict analysis dramatically improves the solving speed and is the defining feature of contemporary SAT solvers.

Lazy clause generation CP solvers make use of SAT conflict analysis [28]. Every change to the domain of an integer variable is implicitly or explicitly associated with a literal. In CP problems with only integer variables (e.g., no set variables or graph variables), conflict analysis generates nogoods in the form of bound disjunction constraints. Nutmeg uses conflict analysis to find nogoods, which are translated into bound disjunction Benders cuts.

3.4 Assumptions Interface

Many decomposition-based solving techniques solve a sequence of closely-related subproblems as a subroutine. In SAT and lazy clause generation CP solvers, this can be facilitated by an *assumptions interface*. Rather than solving a base problem $\mathcal{C}$, $\mathcal{C}$ can be solved subject to *assumptions* $A = a_1 \wedge \ldots \wedge a_n$, where $a_1, \ldots, a_n$ are additional constraints. The solver then produces a solution satisfying both $\mathcal{C}$ and A, or returns an *assumptions nogood* N defined by a subset A' of A such that $\mathcal{C} \wedge A'$ is infeasible. The nogood N takes the form

$$\neg \bigwedge_{a \in A'} a$$

or equivalently,

$$\bigvee_{a \in A'} \neg a.$$

The advantage of the assumptions approach is that the problem $\mathcal{C}$ can be re-solved under different assumptions, while preserving the rest of the solver state. In an arbitrary solver, this could be achieved by solving each instance from scratch and then returning A if $\mathcal{C} \wedge A$ is infeasible. SAT and lazy clause generation CP solvers can do better. By preserving the database of learnt clauses, the solver can avoid re-exploring the same infeasible subtrees in multiple calls to the solver. By using conflict analysis, the solver can also trim A down to a smaller (though not necessarily minimal) set of assumptions that caused the failure. Nutmeg makes use of the assumptions interface in an underlying CP solver when repeatedly solving $\mathcal{C}$. Doing so enables Nutmeg to benefit from efficiencies in the implementation.

4 Branch-and-check in Nutmeg

This section presents the implementation of Nutmeg.

4.1 Underlying Solvers

Nutmeg is a meta-solver that implements a generalization of the branch-and-check ideas of [25, 11]. It is simply a thin layer that calls an underlying MIP solver and CP solver, and communicates information from one solver to the other. At present, the MIP solver is SCIP 6.0.2 [18]. The CP solver is a forthcoming lazy clause generation solver named Geas [17], which is the successor of the experimental but state-of-the-art solver Chuffed [9].

4.2 Modeling and Automatic Decomposition

Nutmeg provides a C++ programming interface and a MiniZinc modeling interface. Using either interface, users can declare variables and add constraints to build a model in the form of either $\mathcal{P}$, or $\mathcal{M}$ and $\mathcal{C}$.

Users can declare a model in the form of $\mathcal{P}$ and defer the decomposition of $\mathcal{P}$ into $\mathcal{M}$ and $\mathcal{C}$ to Nutmeg's automatic decomposition mechanism. To do this, users simply create variables and add constraints as usual. Nutmeg uses a pre-defined library of rewritings to add linear constraints $A_2\mathbf{x} \leq \mathbf{b}_2$ to $\mathcal{M}$ and general constraints $(\mathbf{x}, \mathbf{y}) \in S$ to $\mathcal{C}$. The library of rewritings defines a linearization or linear relaxation in $\mathcal{M}$ and global constraints augmented with redundant constraints to assist propagation in $\mathcal{C}$. The following example shows how the rewriting proceeds.

Example 3 (Planning and Scheduling) Consider the Planning and Scheduling problem in Example 2. Users can input $\mathcal{P}$ as shown in Figure 3. Given $\mathcal{P}$, Nutmeg builds $\mathcal{C}$ by dropping the objective from $\mathcal{P}$. Nutmeg builds $\mathcal{M}$ by substituting global constraints for linear constraints obtained from its library of rewritings. The library currently contains the energy relaxation of CumulativeOptional from Example 1. Therefore, Nutmeg builds $\mathcal{M}$ exactly as presented in Figure 4.

The library of rewritings is made transparent; fully exposing how an input constraint is rewritten into linear constraints in $\mathcal{M}$ and general constraints in $\mathcal{C}$. Instead of adding a high-level $\mathcal{P}$-constraint, users can directly add (a subset of) the $\mathcal{M}$- and $\mathcal{C}$-rewritings. Users can also completely ignore the library of rewritings and add any constraint they desire directly to $\mathcal{M}$ and $\mathcal{C}$. In effect, users can precisely specify $\mathcal{M}$ and $\mathcal{C}$, bypassing Nutmeg's automatic decomposition of $\mathcal{P}$.

The linearization or linear relaxation of some global constraints, such as CumulativeOptional, are easy to implement. Other global constraints are much more difficult to implement, even if they have simple linearizations. Example 4 describes one such constraint.

1. **Node Selection:** Select an open node. Terminate if no open nodes remain.
2. **Feasibility Check:** For every variable x_i with local bounds $\hat{a}_i \leq x_i \leq \hat{b}_i$, add assumptions $x_i \geq \hat{a}_i$ and $x_i \leq \hat{b}_i$ to A. Propagate $\mathcal{C} \wedge A$. If any variable bounds were tightened in $\mathcal{C} \wedge A$, tighten them in $\mathcal{M}$. In the case of an empty domain, the node is infeasible, so perform conflict analysis, add the resulting nogood to both $\mathcal{C}$ and $\mathcal{M}$, and go back to Step 1.
3. **Suboptimality Check:** Solve the LP relaxation. If the objective value is worse than the incumbent solution, go back to Step 1. Otherwise, retrieve its solution $\hat{\mathbf{x}}$.
4. **Candidate Solution Check:** For every variable x_i with value $\hat{x}_i$ in the LP solution, add assumptions $x_i \geq \lfloor \hat{x}_i \rfloor$ and $x_i \leq \lceil \hat{x}_i \rceil$ to A. Solve $\mathcal{C} \wedge A$. If it fails, perform conflict analysis, add the nogood to both $\mathcal{C}$ and $\mathcal{M}$, and go back to Step 3. If it succeeds, store the CP solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ as the incumbent.
5. **Branch:** Select a variable x_i with fractional value $\hat{x}_i$ in the LP solution, if any. Then, create a child node with $x_i \leq \lfloor \hat{x}_i \rfloor$ and another child with $x_i \geq \lceil \hat{x}_i \rceil$. Go to Step 1.

Fig. 5 The branch-and-check algorithm.

Example 4 (Assignment Problem) The AllDifferent global constraint captures the Assignment Problem substructure. Given a vector $\mathbf{x} = (x_1, \ldots, x_I)$ of $I \in \mathbb{Z}_+$ integer variables, AllDifferent($\mathbf{x}$) permits each value to be assigned to at most one of $x_1, \ldots, x_I$. Let $\mathcal{I} = \{1, \ldots, I\}$. The linearization of AllDifferent($\mathbf{x}$) is

$$\sum_{i \in \mathcal{I}} [\![x_i = k]\!] \leq 1 \qquad \forall k \in \bigcup_{i \in \mathcal{I}} D_{x_i},$$

where D_{x_i} is the domain of x_i. Since this linearization is very tight, it is likely worthwhile to include the entire linearization in $\mathcal{M}$. Nutmeg creates every literal $[\![x_i = k]\!]$ as a binary variable in $\mathcal{M}$. These literals are then connected to the originating variable x_i in $\mathcal{C}$. If the integer variable x_i exists in $\mathcal{M}$ (e.g., because another linearization or linear relaxation requires the integer value), Nutmeg also adds a linking constraint. The implementation of all these concepts is highly non-trivial.

4.3 The Branch-and-Check Algorithm

Nutmeg solves $\mathcal{M}$ and $\mathcal{C}$ using the branch-and-cut form of LBBD known as branch-and-check [35,4]. The key difference to branch-and-cut is that cut separation is performed automatically by conflict analysis within a CP solver, rather than implemented specifically for one family of cuts, and that the cuts can span all variables, rather than one class of variables (e.g., subtour elimination cuts in the Traveling Salesman Problem only concern arc variables).

The branch-and-check algorithm is sketched in Figure 5. The main difference to the basic LBBD algorithm in Figure 1 is that it calls the CP subproblem on every LP solution, instead of only on $\mathcal{M}$-optimal solutions, i.e., $\mathcal{P}$-superoptimal solutions.

A subtlety here is that primal and dual bounds are applied to the objective variable (i.e., x_1 from Section 3.1) in the Feasibility Check and Candidate

Solution Check. Therefore, $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ in the Candidate Solution Check is always an improving solution.

Recall from Section 3.1 that $A_1\mathbf{x} \leq \mathbf{b}_1$ is included in the CP subproblem even though it already exists in the MIP master problem. Consider the constraint $2x_1 + x_2 \leq 1$ with a fractional solution $\hat{x}_1 = \hat{x}_2 = 1/3$. The Candidate Solution Check transfers this fractional solution to the CP subproblem as the bounds $0 \leq x_1 \leq 1$ and $0 \leq x_2 \leq 1$. Explicitly including the constraint in the CP subproblem will propagate $x_1 \leq 0$, and hence, fix $x_1 = 0$.

As defined, the branch-and-check algorithm eagerly runs the Candidate Solution Check after every LP solve. This provides for the most interaction between the two problems and prunes nodes at the earliest opportunity. However, in highly fractional LP solutions, this scheme requires more computation time since the CP subproblems could be very difficult to solve.

Calling the Candidate Solution Check only at MIP integral solutions leads to less interaction and less time spent in the CP subproblem but at the expense of a larger search tree. However, this CP subproblem is much easier to solve since the $\mathbf{x}$ variables have integer values, and hence, are fixed by assumptions. Alternatively, the Candidate Solution Check can be run on fractional solutions with a limited computation budget. If running the check on a fractional solution takes too long, aborting the check and continuing with processing the node has no adverse effects. Of course, checking an integer solution cannot be terminated early since it could be a leaf node. The decision about when to call the Candidate Solution Check and with what budget are parameters in the implementation.

4.4 Nogoods

By making assumptions consisting of only bound changes, all resultant nogoods are guaranteed to be bound disjunction constraints that only contain literals concerning variables that appear in the master problem. The procedure for upgrading a nogood and then adding it to the master problem is shown in Figure 6.

The nogoods are not necessarily minimal; i.e., they may contain excess literals not necessary for explaining a failure. Nutmeg begins with a preprocessing step that temporarily removes one literal from the nogood and makes new assumptions on those literals. If the CP subproblem reports the new assumptions are infeasible, then the literal did not contribute to the failure and is permanently removed. Otherwise, the literal is reinstated and the process continues to the next literal. This occurs until every literal is examined. A time limit and conflict limit, given as parameters, are placed on this optional strengthening phase.

If the improved nogood is empty, then the problem is globally infeasible, so the solver exits. If the nogood contains exactly one literal, the bound change is enacted globally. If the nogood only contains literals over binary variables, it can be easily added as a cut. Otherwise, the nogood contains integer variables

1. **Preprocess:** Temporarily remove each literal from the nogood in turn. Make new assumptions using the literals in the nogood. Recheck using the CP subproblem. If feasible, reinstate the literal into the nogood.
2. **Empty Nogood:** If the nogood is empty, the problem is infeasible. Terminate.
3. **Singleton Nogood:** If the nogood contains exactly one literal, the bound change is set globally.
4. **Binary Nogood:** If the nogood only contains literals of binary variables, it is added to the MIP master problem as the cut

$$\sum_{i \in L_1} x_i + \sum_{i \in L_0} (1 - x_i) \geq 1,$$

   where $L_1 = \{i | (i, 1) \in L_{\geq}\}$ and $L_0 = \{i | (i, 0) \in L_{\leq}\}$.
5. **Integer Nogood:** If the nogood contains integer variables, a bound disjunction constraint is added.

Fig. 6 The procedure for adding a nogood to the MIP master problem.

and cannot be simplified. Nutmeg uses SCIP as the MIP solver, which natively supports bound disjunction by propagation. It runs a SAT-style propagator but also branches on one of the disjuncts in certain cases. If a different MIP solver is used, any one of the three approaches mentioned in Section 3.2 for implementing bound disjunction constraints can be used.

Example 5 (Planning and Scheduling) Consider an instance of the Planning and Scheduling problem from Example 2 with time horizon $T = 4$, a single facility 1 with capacity 3 and two jobs 1 and 2. Each job must start between time 1 and 2, has a duration of 3 and uses 2 resources. Activating both jobs satisfies the relaxation (Constraint (3d)) but does not satisfy the original constraint (Constraint (2d)), as illustrated in Figure 2. Given this candidate solution, the CP subproblem will detect an infeasibility and create the nogood

$$[\![x_{1,1} = 0]\!] \vee [\![x_{2,1} = 0]\!],$$

which is added to the MIP master problem as the row

$$(1 - x_{1,1}) + (1 - x_{2,1}) \geq 1,$$

or equivalently, as the clique constraint

$$x_{1,1} + x_{2,1} \leq 1.$$

This constraint is incompatible with Constraint (3c), enabling the MIP solver to declare that the problem is infeasible.

4.5 Block-diagonal Structure

In mathematical programming decompositions (e.g., Dantzig-Wolfe or Benders), block-diagonal structure can be decomposed into one independent subproblem per block. Nutmeg takes a different approach: all “blocks” are contained within

the singular subproblem $\mathcal{C}$. The reason for this is that conflict analysis will never generate a nogood spanning multiple blocks because they are independent by definition.

5 Experimental Results

The experiments compare branch-and-check against pure CP and pure MIP. Two variants of branch-and-check are tested: (1) B&C-LP runs the CP checking subproblem at every fractional and integer solution with a limit of 0.3 seconds and 300 conflicts for fractional solutions, and (2) B&C-MIP runs the CP checking subproblem only at integer solutions. The standalone approaches use the same solvers, specifically, Geas and SCIP. All solvers are single-threaded and are run for ten minutes on an Intel Xeon E5-2660 V3 CPU at 2.6 GHz. The four methods are evaluated on four experiments that explore Nutmeg's performance on different problem classes. The findings are presented below.

5.1 Known Successful Problems

The first experiment runs the four methods on three problems recognized to have successful implementations of LBBD. The purpose of this experiment is to verify that the decomposition, and especially the automatic decomposition, is indeed faster than the standalone approaches. This experiment evaluates the solvers on the following three problems:

- **Planning and Scheduling (P&S):** This problem is introduced in Example 2. There are a total of 335 instances.
- **Capacity- and Distance-constrained Plant Location Problem (CDCPLP):** This problem, formalized in the appendix, is a variation on the classical Facility Location problem [3]. The problem allocates customers to facilities at some cost. All facilities are initially closed and can be opened at some cost if assigned customers. The problem also contains a fleet of distance-limited vehicles stationed at the facilities to serve the customers. The number of vehicles in use also contributes to the total cost. At each facility, customers are assigned to a particular vehicle using a BinPacking constraint, which is a special case of the CumulativeOptional constraint. There are 300 instances.
- **Vehicle Routing Problem with Location Congestion (VRPLC):** This problem, formalized in the appendix, routes vehicles to various sites to deliver goods subject to travel time, time windows and vehicle capacity constraints [24]. Each site features a Cumulative constraint that schedules the deliveries around the availability of equipment for unloading the vehicles. The MIP master problem contains the base Vehicle Routing Problem and the empty relaxation of the Cumulative constraints. The objective minimizes the total travel distance. There are 450 instances.

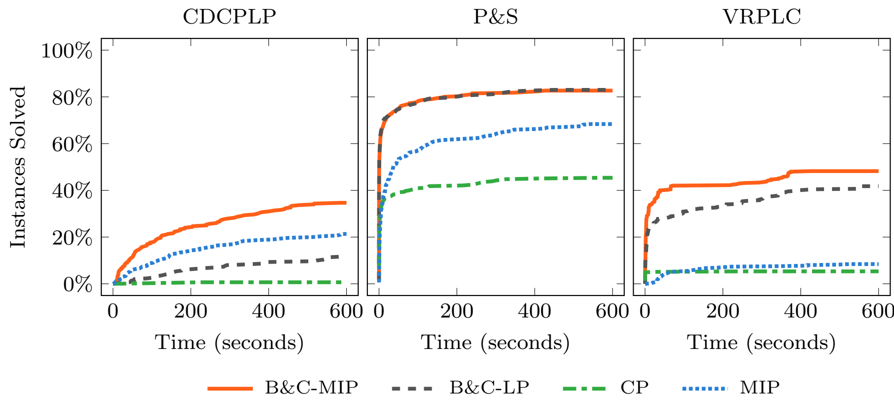

Fig. 7 Cumulative number of instances solved over time for each problem in the first experiment. Higher is better.

All these problems have similar structure: they contain the Assignment Problem plus some side constraints and either CUMULATIVE or CUMULATIVEOPTIONAL global constraints. This problem structure is ideally suited to LBBD because MIP easily solves the base Assignment Problem since it possesses the *integrality property* (i.e., it can be solved in one call to an LP solver [10]), and CP excels at cumulative scheduling but has difficulty at reasoning over linear constraints. Given this problem structure, LBBD follows naturally.

Figure 7 plots the number of instances solved to proven optimality or infeasibility. On CDCPLP, B&C-MIP solves the most instances, followed by MIP. B&C-LP solves one-third the number of instances solved by B&C-MIP. On the instances with feasible but not provably optimal solutions, B&C-MIP, B&C-LP and MIP achieve an average optimality gap of 8.9%, 15.5% and 19.1% respectively. (Since CP does not solve a relaxation, dual bounds and gaps are not available.)

On P&S, the two branch-and-check methods are essentially identical, solving many more instances than the standalone methods. B&C-MIP, B&C-LP and MIP obtain an average optimality gap of 19.9%, 22.3% and 1.9% on the instances with feasible but not optimal solutions. There are several instances in which branch-and-check gets stuck within the CP subproblem, resulting in a 100% optimality gap, which skews these statistics in favor of MIP.

On VRPLC, the two branch-and-check methods solve substantially more instances than standalone CP and MIP. The optimality gap of feasible but not provably optimal instances are 21.5%, 45.7% and 53.9% for B&C-MIP, B&C-LP and MIP.

These results suggest that B&C-MIP dominates B&C-LP. Presumably, this is due to the large number of calls asking the subproblem to check fractional solutions that are CP-feasible; resulting in no cut being generated but time wasted on checking them. Overall, these results indicate that Nutmeg's automatic decomposition succeeds on problems with structure suited to LBBD.

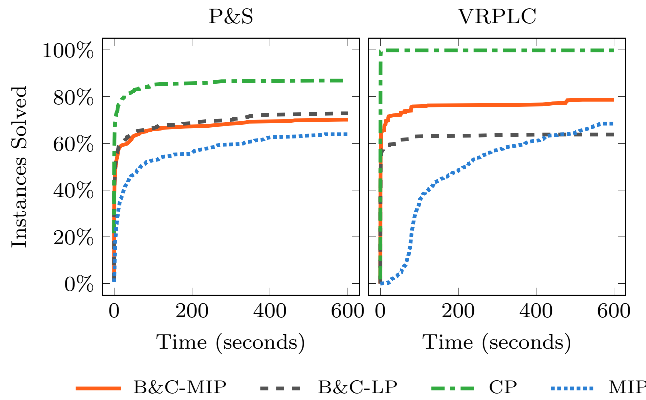

Fig. 8 Cumulative number of instances solved over time for each problem in the second experiment. Higher is better.

5.2 Makespan Objective Function

The previous experiment consists of problems with linear constraints, scheduling constraints and a linear objective function, which is natural for MIP but difficult for CP. The second experiment swaps the cost objective for the makespan objective, which appears frequently in scheduling problems. This objective function minimizes the time at which the last task is completed. This kind of minimax objective is known to perform poorly in MIP because it has a weak linear relaxation, but perform well in CP because optimizing a minimax objective is essentially equivalent to solving a sequence of progressively tighter satisfiability problems. The goal of the second experiment is to determine whether the MIP master problem can optimize a CP-preferred objective solely via the subproblem.

This experiment compares the four methods on P&S and VRPLC from the previous experiment but with the makespan objective. CDCPLP is excluded because it uses the BinPacking constraint, which has no meaning in the context of scheduling, despite being implemented by the same propagator as CumulativeOptional.

Figure 8 clearly shows that the minimax objective sufficiently destroys the nice structure, allowing pure CP to outperform the other approaches. For P&S, the average optimality gap of feasible but not provably optimal instances are 40.0%, 39.3% and 18.0% for B&C-MIP, B&C-LP and MIP. Again, MIP finds bounds tighter than branch-and-check on average because branch-and-check gets trapped within the CP subproblem on some instances, resulting in a gap of 100%. For VRPLC, the instances are trivial for CP. B&C-MIP, B&C-LP and MIP find an average optimality gap of 50.4%, 63.6% and 67.3% on feasible but not optimal instances. These results demonstrate that the branch-and-check master problem has little information for optimizing the objective after moving the scheduling constraints into the subproblem.

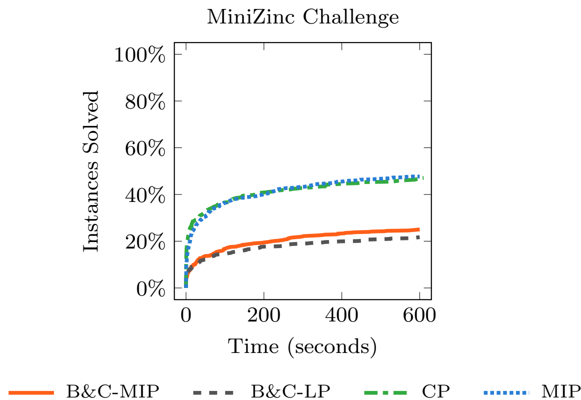

Fig. 9 Cumulative number of instances solved over time in the third experiment. Higher is better.

5.3 MiniZinc Challenge

The next experiment comprises problems from the 2013 to 2019 MiniZinc Challenges. This experiment, the first of its kind, evaluates the main contribution of Nutmeg; that is, the automatic LBBD of arbitrary problems, including those for which decomposable structure is either not known or not explicitly programmed. Six problems (35 instances) are excluded because they contain global constraints whose propagator is not yet implemented in Geas. In total, this experiment consists of 667 instances across 94 problems. These problems are highly varied: while several problems contain simple structure that could be suitable for LBBD after some manual manipulation, many problems are clearly unstructured.

Figure 9 shows that branch-and-check performs poorly without appropriate structure. Branch-and-check solves significantly fewer instances than CP and MIP because the master problem has little or no knowledge about the constraints moved into the subproblem, and hence, continually searches for solutions that do not exist. Even though MIP solves many more instances, the average optimality gaps of B&C-MIP, B&C-LP and MIP on feasible instances are 68.3%, 70.9% and 218.8%. These numbers are skewed against MIP because there are outliers with a very high optimality gap (one as high as 10200%).

5.4 Spot5

Analyzing the results to each problem in the MiniZinc Challenges reveals that Nutmeg does perform well on one of these problems, namely, the Spot5 problem. This problem, formalized in the appendix, only contains Table global constraints, which currently has no linear relaxation. That is, the MIP master problem has no constraints at all.

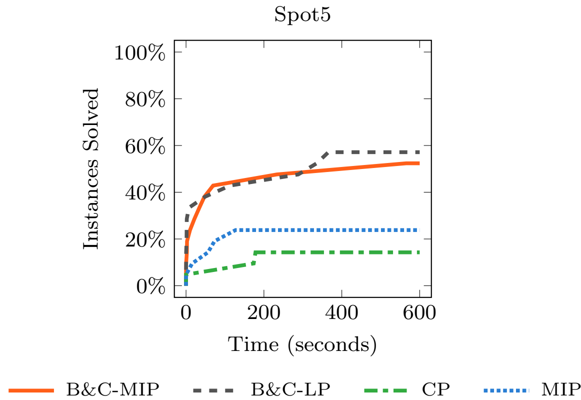

Fig. 10 Cumulative number of instances solved over time in the fourth experiment. Higher is better.

Contributors to the MiniZinc Challenge submit a model together with a large number of instances to the organizers, who shortlist a few instances based on their difficulty. Ten instances of Spot5 are included in the MiniZinc Challenge but a total of 21 instances are submitted. The final experiment runs all 21 instances in order to confirm results suggesting that Nutmeg performs well on the initial ten instances.

Figure 10 indicates that B&C-LP outperforms the other methods on this problem, solving one more instance than B&C-MIP. The average optimality gap of B&C-MIP, B&C-LP and MIP on the instances with feasible but not optimal solutions are 58.1%, 43.4% and 60.4%.

This experiment demonstrates that Nutmeg does indeed have a purpose. In general, decomposable structure is necessary for Nutmeg to perform well. However, even when that structure is not declared, there are problems (albeit few) where the automatic decomposition is useful. These findings suggest that a hand-tailored implementation of branch-and-check could be worthwhile for the Spot5 problem.

6 Conclusions and Future Work

LBBD is an important framework for separating a problem into a master problem and one or more subproblems that can be tackled using any technology for which an inference dual is available. The master problem is often MIP and the subproblems CP. Unlike standard Benders decomposition in mathematical programming, LBBD is much more general, and hence, its Benders cuts have no exact form; their form must be invented for every problem, essentially making re-use impossible.

Recently, two teams independently developed CP conflict analysis into a generic procedure for separating Benders cuts in LBBD. The present paper,

authored by both teams together, further advances these ideas in a definitive journal version of the earlier conference works. This paper describes the implementation of a new MIP and CP hybrid meta-solver named Nutmeg. The solver hybridizes MIP and CP using branch-and-check, a method based on LBBD that tightly connects a MIP master problem and a CP checking subproblem within a single branch-and-bound search tree. Given an arbitrary high-level CP problem, Nutmeg automatically derives a MIP relaxation that omits global constraints with weak linear relaxations, and uses the LP relaxation to compute dual bounds, enabling earlier pruning of suboptimal subtrees in comparison to pure CP solvers. Reasoning about individual substructures embedded within global constraints is handled using CP, which generates Benders cuts via conflict analysis. Information about domains and bounds are also passed in both directions.

Nutmeg is evaluated on a variety of problem classes. The results indicate that branch-and-check performs well on problems with decomposable structure that can be nicely separated into MIP-preferred portions and CP-preferred portions (e.g., an Assignment Problem with cumulative scheduling). However, in general, branch-and-check performs poorly on problems lacking this structure. The master problem repeatedly proposes solutions that are simply infeasible because it has no knowledge about the omitted constraints. Nevertheless, Nutmeg solves more instances of the Spot5 problem from the MiniZinc Challenge than pure MIP and pure CP, even though no decomposable structure is available. This result indicates that studying the problem to find a decomposable structure and then developing a hand-tailored problem-specific solver could be worthwhile.

Calling the checking subproblem on every fractional solution in the branch-and-bound tree is often slower than calling it only on integer solutions. Presumably, this is because of the large number of fractional solutions feasible in the subproblem and also because every cutting plane internally added by SCIP during the branch-and-cut search also induces another round of calling the checking subproblem. This leads to an excessive number of calls, which may not generate useful nogoods. Future work should devise better strategies of when to solve the subproblem and for how long.

Nutmeg does not currently divide block-diagonal structure into multiple independent subproblems, but rather, uses a single monolithic subproblem. Even though there is no benefit in theory due to conflict analysis, solving multiple smaller subproblems is likely to be faster than solving one large subproblem in practice. A low-priority improvement to the implementation is to add the ability to split the subproblem into several easier subproblems.

Nutmeg's automatic decomposition relies on its pre-defined library of rewritings, which is fairly basic at this stage and remains a topic of continuing work. Many of the global constraints currently use an empty linear relaxation, and hence, the MIP master problem receives no information at all, leaving the entire constraint to be considered in the CP subproblem. For the approach to perform well, the number of constraints in the linear relaxation of a difficult global constraint must be few and these constraints must be reasonably tight. One ongoing challenge is to find a good balance between a relaxation that

closely resembles the original global constraint and one that does not introduce too many new columns and/or rows into the MIP master problem.

Strengthening cuts is a common technique of branch-and-cut models of many MIP problems. Usually, the strengthening relies on problem-specific polyhedral analysis absent in general. Problem-specific cuts can be strengthened by recognizing the form of the nogoods and then lifting them as in the existing approaches [25]. A general mechanism to strengthen cuts for arbitrary problems remains an important unanswered question.

Cuts separated by SCIP during the solution process (e.g., knapsack cover cuts) are not transferred to the CP checking subproblem. Since linear constraints do not propagate strongly in CP, these cuts are not expected to have large impact. Future work can consider transferring these cuts and investigating new types of generic (global) constraints that will propagate strongly in CP.

A branch-and-check option can be considered within a parallel portfolio of solvers that includes standalone CP and MIP. Run-time information from the independent runs of the CP and MIP solvers may be able to guide the search towards a better trade-off, or indeed decide that all effort should be concentrated on a pure CP or pure MIP approach.

It will also be interesting to implement automatic LBBD within other automatic decomposition solvers, such as GCG [16,15]. GCG can seamlessly reformulate a problem using Dantzig-Wolfe decomposition and solve it using branch-and-cut-and-price. The two main benefits of Dantzig-Wolfe reformulation are that it achieves a tighter linear relaxation and that it completely eliminates symmetries arising from permuting the index of variables. For these two reasons, state-of-the-art methods for many Vehicle Routing Problems are based on branch-and-cut-and-price. Conversely, state-of-the-art methods for problems with cumulative scheduling are based on CP because their linear relaxations are weak. Connecting Nutmeg with GCG will allow a combination of automatic Dantzig-Wolfe reformulation and automatic LBBD of high-level models. This should benefit problems like the VRPLC by removing symmetry in the index of the vehicles using Dantzig-Wolfe reformulation and by reasoning across the timing of all routes using propagation.

In conclusion, Nutmeg adds the first hybrid solving option to the many choices of singular-approach solvers available within the MiniZinc modeling system. For inexperienced modelers, Nutmeg enables effortless hybrid solving with one click. For expert modelers, Nutmeg serves as a useful tool for quickly evaluating different structures for LBBD within a user-friendly modeling language.

References

1. Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optimization **4**(1), 4 – 20 (2007)
2. Achterberg, T.: SCIP: solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)
3. Albareda-Sambola, M., Fernández, E., Laporte, G.: The capacity and distance constrained plant location problem. Computers & Operations Research **36**, 597–611 (2009)

4. Beck, J.C.: Checking-up on branch-and-check. In: D. Cohen (ed.) Principles and Practice of Constraint Programming – CP 2010, *Lecture Notes in Computer Science*, vol. 6308, pp. 84–98. Springer Berlin Heidelberg (2010)
5. Belov, G., Stuckey, P.J., Tack, G., Wallace, M.: Improved linearization of constraint programming models. In: M. Rueher (ed.) Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016. Proceedings, pp. 49–65. Springer International Publishing (2016)
6. Benchimol, P., van Hoeve, W.J., Régin, J.C., Rousseau, L.M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints **17**(3), 205–233 (2012)
7. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische mathematik **4**(1), 238–252 (1962)
8. Bensana, E., Lemaitre, M., Verfaillie, G.: Earth observation satellite management. Constraints **4**(3), 293–299 (1999)
9. Chu, G.G.: Improving combinatorial optimization. Ph.D. thesis, University of Melbourne (2011). URL `http://hdl.handle.net/11343/36679`
10. Conforti, M., Cornuéjols, G., Zambelli, G.: Integer programming, vol. 271. Springer (2014)
11. Davies, T.O., Gange, G., Stuckey, P.J.: Automatic logic-based Benders decomposition with MiniZinc. In: AAAI, pp. 787–793 (2017)
12. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: I.P. Gent (ed.) Principles and Practice of Constraint Programming – CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings, pp. 352–366. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
13. Focacci, F., Lodi, A., Milano, M.: Optimization-oriented global constraints. Constraints **7**(3-4), 351–365 (2002)
14. Fontaine, D., Michel, L., Van Hentenryck, P.: Constraint-based Lagrangian relaxation. In: B. O'Sullivan (ed.) Principles and Practice of Constraint Programming: 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8656, pp. 324–339. Springer International Publishing (2014)
15. Gamrath, G.: Generic branch-cut-and-price. Ph.D. thesis, Technischen Universität Berlin (2010)
16. Gamrath, G., Lübbecke, M.E.: Experiments with a generic dantzig-wolfe decomposition for integer programs. In: P. Festa (ed.) Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings, pp. 239–252. Springer Berlin Heidelberg (2010)
17. Gange, G., Berg, J., Demirović, E., Stuckey, P.J.: Core-guided and core-boosted search for CP. In: Proceedings of the 7th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. (to appear) (2020)
18. Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gamrath, G., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Lübbecke, M.E., Maher, S.J., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Schubert, C., Serrano, F., Shinano, Y., Viernickel, J.M., Walter, M., Wegscheider, F., Witt, J.T., Witzig, J.: The SCIP Optimization Suite 6.0. ZIB-Report 18-26, Zuse Institute Berlin (2018). URL `http://nbn-resolving.de/urn:nbn:de:0297-zib-69361`
19. Hooker, J.N.: A hybrid method for planning and scheduling. In: M. Wallace (ed.) Principles and Practice of Constraint Programming – CP 2004, *Lecture Notes in Computer Science*, vol. 3258, pp. 305–316. Springer Berlin Heidelberg (2004)
20. Hooker, J.N.: Planning and scheduling by logic-based Benders decomposition. Operations Research **55**(3), 588–602 (2007)
21. Hooker, J.N., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96**(1), 33–60 (2003)
22. Junker, U., Karisch, S.E., Kohl, N., Vaaben, B., Fahle, T., Sellmann, M.: A framework for constraint programming based column generation. In: J. Jaffar (ed.) Principles and Practice of Constraint Programming – CP'99: 5th International Conference, CP'99, Alexandria, VA, USA, October 11-14, 1999. Proceedings, pp. 261–274. Springer (1999)
23. Lam, E.: Hybrid optimization of vehicle routing problems. Ph.D. thesis, University of Melbourne (2017). URL `http://hdl.handle.net/11343/220534`
24. Lam, E., Van Hentenryck, P.: A branch-and-price-and-check model for the vehicle routing problem with location congestion. Constraints **21**(3), 394–412 (2016)

25. Lam, E., Van Hentenryck, P.: Branch-and-check with explanations for the Vehicle Routing Problem with Time Windows. In: J.C. Beck (ed.) Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings, pp. 579–595. Springer, Cham (2017)
26. Marques Silva, J.a.P., Sakallah, K.A.: GRASP–a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design, ICCAD '96, pp. 220–227. IEEE Computer Society (1996)
27. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: C. Bessière (ed.) Principles and Practice of Constraint Programming – CP 2007, pp. 529–543. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
28. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
29. Refalo, P.: Linear formulation of constraint programming models and hybrid solvers. In: R. Dechter (ed.) Principles and Practice of Constraint Programming – CP 2000, pp. 369–383. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
30. Régin, J.C.: Cost-based arc consistency for global cardinality constraints. Constraints **7**(3), 387–405 (2002)
31. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints **16**(3), 250–282 (2010)
32. Shen, K., Schimpf, J.: Eplex: Harnessing mathematical programming solvers for constraint logic programming. In: Principles and Practice of Constraint Programming: 11th International Conference, CP2005, Proceedings, *LNCS*, vol. 3709, pp. 622–636. Springer (2005)
33. Steiger, R., van Hoeve, W.J., Szymanek, R.: An efficient generic network flow constraint. In: Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 893–900. ACM (2011)
34. Taşkin, Z.C.: Benders Decomposition. In: Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc. (2010)
35. Thorsteinsson, E.: Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: T. Walsh (ed.) Principles and Practice of Constraint Programming – CP 2001, *Lecture Notes in Computer Science*, vol. 2239, pp. 16–30. Springer Berlin Heidelberg (2001)

Acknowledgments

We would like to thank the two anonymous reviewers whose comments have substantially improved this paper.

Declarations

– **Funding:** This study was not funded by a grant.
– **Conflicts of Interest:** The authors declare that they have no conflicts of interest.
– **Code Availability:** The source code of Nutmeg is made available at the first author's website[1].
– **Availability of Data and Material:** All problems and instances are either contained within the source code repository of Nutmeg or available at the MiniZinc Challenge website[2].

[1] https://ed-lam.com
[2] https://www.minizinc.org

Appendix

This appendix presents several models used in the experiments.

Capacity- and Distance-constrained Plant Location Problem

The Capacity- and Distance-constrained Plant Location Problem (CDCPLP) is proposed in [3]. This problem considers a set $\mathcal{F}$ of facilities. Every facility $f \in \mathcal{F}$ is initially closed but can be opened at a cost $w_f^{\text{open}} \in \mathbb{Z}_+$. Let $o_f \in \{0,1\}$ be a binary variable indicating whether facility $f \in \mathcal{F}$ is opened.

Let $\mathcal{C}$ be the set of customers, each of which must be assigned to an opened facility. Assigning customer $c \in \mathcal{C}$ to facility $f \in \mathcal{F}$ incurs a cost $w_{c,f}^{\text{assign}} \in \mathbb{Z}_+$. Let $x_{c,f} \in \{0,1\}$ be a binary variable indicating whether customer $c \in \mathcal{C}$ is assigned to facility $f \in \mathcal{F}$. Every customer $c \in \mathcal{C}$ requires $d_c \in \mathbb{Z}_+$ of demand. Each facility $f \in \mathcal{F}$ can support up to $D_f \in \mathbb{Z}_+$ of demand.

Customers assigned to a facility receive deliveries from trucks stationed at the facility. Allow up to $T \in \mathbb{Z}_+$ trucks to be stationed at a facility, and let $\mathcal{T} = \{1, \ldots, T\}$ be the set of trucks. Let $t_c \in \mathcal{T}$ be an integer variable for the number of the truck assigned to customer $c \in \mathcal{C}$. Define $q_{c,f} \in \mathbb{Z}_+$ as the distance from facility $f \in \mathcal{F}$ to customer $c \in \mathcal{C}$ and back. Each truck can carry the goods of only one customer at a time, and can travel up to $Q \in \mathbb{Z}_+$ in total distance. Let $n_f \in \mathcal{T}$ be an integer variable for the total number of trucks required at facility $f \in \mathcal{F}$. Every truck kept at facility $f \in \mathcal{F}$ incurs a cost $w_f^{\text{truck}} \in \mathbb{Z}_+$.

The model is shown in Figure 11. Objective Function (7a) minimizes the total cost of (1) opening facilities, (2) assigning customers to facilities and (3) keeping trucks at facilities. Constraint (7b) opens a facility if it is assigned customers. This constraint also limits the number of customers assigned to a facility according to its maximum demand. Constraint (7c) assigns customers to trucks stationed at a facility while considering the total travel distance of each truck. Constraint (7d) calculates the number of trucks used at a facility.

Constraint (7e) and (7f) are redundant constraints, which improve the propagation. Constraint (7e) bounds the number of trucks required at a facility. Constraint (7f) limits the number of customers assigned to a facility. Constraints (7g) to (7j) give the variable domains.

Vehicle Routing Problem with Location Congestion

The Vehicle Routing Problem with Location Congestion (VRPLC) is introduced in [24]. The problem tasks a set of vehicles to deliver goods from a central depot to various locations subject to vehicle constraints and location constraints.

Let $\mathcal{R}$ be the set of requests to be delivered. Let $\perp$ denote the depot, and let $\mathcal{N} = \mathcal{R} \cup \{\perp\}$. Let $\mathcal{L}$ be the set of locations. Every request $i \in \mathcal{R}$ must be delivered to location $l_i \in \mathcal{L}$. Let $\mathcal{R}_l = \{i \in \mathcal{R} | l_i = l\}$ be the requests to be

$$\min \sum_{f\in\mathcal{F}} w_f^{\text{open}} \cdot o_f + \sum_{c\in\mathcal{C}} \sum_{f\in\mathcal{F}} w_{c,f}^{\text{assign}} \cdot x_{c,f} + \sum_{f\in\mathcal{F}} w_f^{\text{truck}} \cdot n_f \tag{7a}$$

subject to

$$\sum_{c\in\mathcal{C}} d_c \cdot x_{c,f} \leq D_f \cdot o_f \qquad \forall f \in \mathcal{F}, \tag{7b}$$

$$\textsc{BinPacking}((x_{c,f}|c\in\mathcal{C}),(t_c|c\in\mathcal{C}),(q_{c,f}|c\in\mathcal{C}),Q) \qquad \forall f \in \mathcal{F}, \tag{7c}$$

$$x_{c,f} \rightarrow n_f \geq t_c \qquad \forall f \in \mathcal{F}, c \in \mathcal{C}, \tag{7d}$$

$$n_f \leq \sum_{c\in\mathcal{C}} x_{c,f} \qquad \forall f \in \mathcal{F}, \tag{7e}$$

$$\sum_{c\in\mathcal{C}} q_{c,f} \cdot x_{c,f} \leq Q \cdot n_f \qquad \forall f \in \mathcal{F}, \tag{7f}$$

$$o_f \in \{0,1\} \qquad \forall f \in \mathcal{F}, \tag{7g}$$

$$x_{c,f} \in \{0,1\} \qquad \forall c \in \mathcal{C}, f \in \mathcal{F}, \tag{7h}$$

$$t_c \in \mathcal{T} \qquad \forall c \in \mathcal{C}, \tag{7i}$$

$$n_f \in \mathcal{T} \qquad \forall f \in \mathcal{F}. \tag{7j}$$

Fig. 11 The high-level model of the CDCPLP.

delivered to location $l \in \mathcal{L}$. The key difference to standard Vehicle Routing Problems is the inclusion of location scheduling constraints. Every location l has a limited number $C_l \in \mathbb{Z}_+$ of equipment for unloading a vehicle. Therefore, deliveries must be scheduled around the availability of the equipment.

Let $T \in \mathbb{Z}_+$ be the time horizon. All deliveries must be completed and all vehicles must return to the depot before T. The problem also considers the usual vehicle capacity and time window constraints. Every request $i \in \mathcal{N}$ has weight $q_i \in \mathbb{Z}_+$ ($q_\perp = 0$) and every vehicle can carry up to $Q \in \mathbb{Z}_+$ in weight. Delivery $i \in \mathcal{N}$ must begin after $a_i \in \mathbb{Z}_+$ ($a_\perp = 0$) and before $b_i \in \mathbb{Z}_+$ ($b_\perp = T$). Performing delivery $i \in \mathcal{R}$ uses one piece of equipment at l_i for $s_i \in \mathbb{Z}_+$ time.

Let $\mathcal{A} = \mathcal{N} \times \mathcal{N} \setminus \{(i,i)|i \in \mathcal{N}\}$ denote the arcs. Define $x_{i,j}$ as a binary variable indicating whether a vehicle travels along arc $(i,j) \in \mathcal{A}$. Traveling along (i,j) consumes $c_{i,j} \in \mathbb{Z}_+$ time. Define a variable w_i for the total weight delivered after $i \in \mathcal{N}$ along a route, and define a variable t_i for the time of starting delivery $i \in \mathcal{N}$. The model is presented in Figure 12. Objective Function (8a) minimizes the total travel time. Constraints (8b) and (8c) requires every request to be delivered. Constraint (8d) limits the total weight on-board a vehicle. Constraint (8e) enforces the travel time between two deliveries. Constraint (8f) schedules the requests at locations. Constraints (8g) to (8i) give the variable domains.

$$\min \sum_{(i,j)\in\mathcal{A}} c_{i,j} \cdot x_{i,j} \tag{8a}$$

subject to

$$\sum_{h:(h,i)\in\mathcal{A}} x_{h,i} = 1 \qquad \forall i \in \mathcal{R}, \tag{8b}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{i,j} = 1 \qquad \forall i \in \mathcal{R}. \tag{8c}$$

$$x_{i,j} \to w_j \geq w_i + q_j \qquad \forall (i,j) \in \mathcal{A}, \tag{8d}$$

$$x_{i,j} \to t_j \geq t_i + c_{i,j} \qquad \forall (i,j) \in \mathcal{A}, \tag{8e}$$

$$\textsc{Cumulative}((t_i | i \in \mathcal{R}_l), (s_i | i \in \mathcal{R}_l), (1 | i \in \mathcal{R}_l), C_l) \qquad \forall l \in \mathcal{L}, \tag{8f}$$

$$x_{i,j} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, \tag{8g}$$

$$w_i \in \{q_i, \ldots, Q\} \qquad \forall i \in \mathcal{N}, \tag{8h}$$

$$t_i \in \{a_i, \ldots, b_i\} \qquad \forall i \in \mathcal{N}. \tag{8i}$$

Fig. 12 The high-level model of the VRPLC.

Spot5

The Spot5 problem is proposed in [8]. The problem concerns one of the SPOT commercial imaging satellites; specifically, the fifth satellite. Given a set of images purchased by clients, the Spot5 problem decides on a subset of images to capture in the next day, subject to operational constraints, such as the availability of imaging instruments and sufficient transition time between two successive images.

The satellite has three imaging instruments labeled 1, 2 and 3 from left to right. Let I be the set of purchased images, and let x_i be an integer variable storing the instruments that will capture image $i \in I$. An image i can be postponed ($x_i = 0$) or captured using a predetermined subset of compatible instruments: left only ($x_i = 1$), middle only ($x_i = 2$), right only ($x_i = 3$) or both left and right ($x_i = 13$) for stereoscopic images. Let $D = \{\{0,2\},\{0,13\},\{0,1,2,3\}\}$, and let $D_i \in D$ be the set of possible instruments for capturing image i, i.e., the domain of x_i. Every image i not captured is penalized by a cost $c_i \in \mathbb{Z}_+$.

The problem uses the Table global constraint. Given a vector $\mathbf{p} \in \mathbb{Z}^n$ with length n and a set $P \subset \mathbb{Z}^n$ of vectors with length n, the constraint $\textsc{Table}(\mathbf{p}, P)$ states that $\mathbf{p} \in P$. Put simply, the constraint requires $\mathbf{p}$ to be equal to a row in a table with rows P.

The problem contains Table constraints that define compatibility between two images, called binary constraints, and constraints stating compatibility between three images, called ternary constraints. Let $\mathcal{A}$ be the set of binary constraints, where each constraint $a \in \mathcal{A}$ is associated with two images $u_a, v_a \in I$ and a compatibility table $T_a \subset D \times D$. Let $\mathcal{B}$ be the set of ternary constraints,

$$\min \sum_{i \in I} c_i \cdot [\![x_i = 0]\!] \tag{9a}$$

subject to

$$\textsc{Table}((x_{u_a}, x_{v_a}), T_a) \qquad \forall a \in \mathcal{A}, \tag{9b}$$
$$\textsc{Table}((x_{u_b}, x_{v_b}, x_{w_b}), T_b) \qquad \forall b \in \mathcal{B}, \tag{9c}$$
$$x_i \in D_i \qquad \forall i \in \mathcal{I}. \tag{9d}$$

Fig. 13 The high-level model of the Spot5 problem.

where each constraint $b \in \mathcal{B}$ is associated with three images $u_b, v_b, w_b \in I$ and a compatibility table $T_b \subset D \times D \times D$.

The model is now presented in Figure 13. Objective Function (9a) penalizes postponed images. Constraints (9b) and (9c) define compatible images. Constraint (9d) give the variable domains.