Artificial Intelligence 212 (2014) 116–133

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Filtering AtMostNValue with difference constraints: Application to the shift minimisation personnel task scheduling problem ☆

Jean-Guillaume Fages [a,*], Tanguy Lapègue [b]

[a] École des Mines de Nantes, LINA (UMR CNRS 6241), LUNAM Université, 4 rue Alfred Kastler, La Chantrerie, BP20722, 44307 Nantes Cedex 3, France
[b] École des Mines de Nantes, IRCCyN (UMR CNRS 6597), LUNAM Université, 4 rue Alfred Kastler, La Chantrerie, BP20722, 44307 Nantes Cedex 3, France

ARTICLE INFO

Article history:
Received 3 February 2014
Received in revised form 31 March 2014
Accepted 4 April 2014
Available online 12 April 2014

Keywords:
Constraint-Programming
Global constraints
AtMostNValue
Shift Minimisation Personnel Task Scheduling Problem

ABSTRACT

The problem of minimising the number of distinct values among a set of variables subject to difference constraints occurs in many real-life contexts. This is the case of the Shift Minimisation Personnel Task Scheduling Problem, introduced by Krishnamoorthy et al., which is used as a case study all along this paper. Constraint-Programming enables to formulate this problem easily, through several AllDifferent constraints and a single AtMostNValue constraint. However, the independence of these constraints results in a poor lower bounding, hence a difficulty to prove optimality. This paper introduces a formalism to describe a family of propagators for AtMostNValue. In particular, we provide simple but significant improvement of the state-of-the-art AtMostNValue propagator of Bessière et al., to filter the conjunction of an AtMostNValue constraint and disequalities. In addition, we provide an original search strategy which relies on constraint reification. Extensive experiments show that our contribution significantly improves a straightforward model, so that it competes with the best known approaches from Operational Research.

1. Introduction

The problem of minimising the number of distinct values among a set of variables subject to difference constraints occurs in many real-life contexts, where an assignment of resources to tasks has to be optimised. For instance, in transports, crews have to be assigned to trips [42]. In schools, classes have to be assigned to rooms [12]. In airports, maintenance tasks have to be assigned to ground crew employees [16,17]. In some factories, fixed jobs have to be assigned to machines [23,24,38]. In a more theoretical context, one may need to colour a graph, such that adjacent vertices have distinct colours and not every colour can be taken by every node [28,30]. In order to illustrate our contribution, we consider the Shift Minimisation Personnel Task Scheduling Problem (SMPTSP) [36]. This problem belongs to the set of personnel scheduling problems [19, 57]. It arises when a set of tasks, fixed in time, have to be assigned to a set of shifts so that overlapping tasks are not

☆ This paper is an invited revision of a paper first published at the 19th International Conference on Principles and Practice of Constraint Programming (CP 2013) where it was recognized as the best student paper.

* Corresponding author.
E-mail addresses: jean-guillaume.fages@mines-nantes.fr (J.-G. Fages), tanguy.lapegue@mines-nantes.fr (T. Lapègue).

http://dx.doi.org/10.1016/j.artint.2014.04.001
0004-3702/

assigned to the same shift. Each shift is associated with a given subset of assignable tasks. The objective is to minimise the number of used shifts. This problem typically occurs as the second step of decomposition methods which handle the creation of shifts in a first step. With this kind of methods, side constraints related to personnel roster design are considered in the first step only, whence the simplicity of the SMPTSP formulation. Nonetheless, current exact approaches relying on Mixed Integer Programming (MIP) fail to solve large scale instances [36,43]. This is the main motivation for investigating a Constraint-Programming (CP) approach.

CP is a programming paradigm which belongs to the wide field of Artificial Intelligence (AI). It is a declarative language which enables to model both satisfaction and optimisation problems through variables, domains, and constraints [52]. Each variable is associated with a domain representing its possible values. The constraints are defined over variables to model the properties which must hold. Therefore, a solution is an assignment of values to variables which satisfies every constraint of the model. From a technical point of view, a constraint is equipped with at least one propagator which filters infeasible values from variable domains. Each time a variable domain is reduced, the propagators associated with this variable are scheduled for further filtering. This leads to series of filtering steps, called propagation [54]. A propagator is monotonic [55] when the smaller the domains before filtering, the smaller the domains after filtering, under the inclusion. Furthermore, it is idempotent [55] if applying its filtering algorithm twice in a row leads to no further inference. Overall, the solving process relies on a backtrack algorithm which alternates filtering steps to make inference and branching decisions to perform hypothesis.

The core idea of CP is to design independent constraints that can be combined through shared variables, in order to model constrained problems. However, in case of optimisation problem, such independent constraints often lead to a poor bounding of the objective function compared to a MIP approach. This restrains the deployment of CP into industrial applications [39]. Therefore, it is often more interesting to design global constraints [4]. These constraints are able to consider a larger part of the problem, hence their filtering impact is increased. For instance, the `AllDifferent` global constraint is the conjunction over a clique of binary difference constraints, and it has been proved highly relevant within CP solvers [50]. However, developing effective global constraints is often difficult, and it also tends to make CP solver maintenance more expensive, which is one of the greatest concerns of the CP community [47]. Consequently, from a practical point of view, one would rather adapt existing constraints than to implement brand new ones, in order to capitalise over previous work. In this paper we investigate the interest of considering difference constraints when filtering the well known `AtMostNValue` constraint [3,7,48]. We introduce a new propagator whose implementation is based on the state-of-the-art `AtMostNValue` propagator [7]. Furthermore, we provide a bold search strategy which relies on constraint reification to attempt strong search space reduction. A wide range of experiments shows that our contribution significantly improves the CP model, so that it competes with the most recent SMPTSP dedicated approaches.

The remainder of the paper is organised as follows: Section 2 is devoted to the description of the SMPTSP, in Section 3 we show how the straightforward CP model of the SMPTSP can be improved with a new propagator. Then, a branching scheme suited to solve the SMPTSP is presented in Section 4. Our approach is validated by an extensive experimental study in Section 5, followed by our conclusions.

2. Description of the SMPTSP

This section first introduces the notations that are used in the paper. Then, some simple complexity results are mentioned to give insight over our case study. Finally, it provides the state-of-the-art mathematical formulation of the problem.

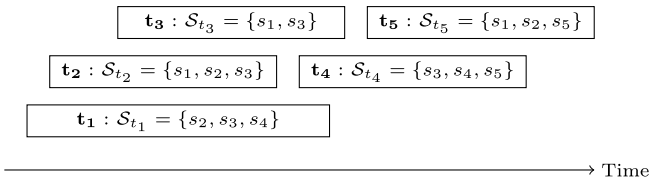
2.1. Notations

In the following, $\mathcal{T}$ and $\mathcal{S}$ respectively refer to the set of tasks and shifts. Note that a shift may be seen as a worker with specific skills and potentially a working time window [36]. Given a task $t \in \mathcal{T}$, we refer to the set of shifts that can be assigned to t as $\mathcal{S}_t \subseteq \mathcal{S}$. Therefore, the SMPTSP consists of assigning to each task $t \in \mathcal{T}$, a shift $s \in \mathcal{S}_t$, with a minimum total number of shifts, and such that overlapping tasks have different shifts. As tasks are fixed in time, finding all maximal sets of overlapping tasks, referred to as $\mathcal{C}$ is polynomial [29]. It amounts to finding the set of maximal cliques in an interval graph. The size of the largest clique, referred to as $LB_{\neq}$, provides a trivial lower bound on the required number of shifts.

Fig. 1 introduces the running example of the paper. The input instance data is displayed in Fig. 1a. It includes a set of 5 tasks $\mathcal{T} = \{t_1, .., t_5\}$ and a set of 5 shifts $\mathcal{S} = \{s_1, .., s_5\}$. Each task, represented by a rectangle, is associated with a set of compatible shifts, *e.g.* $\mathcal{S}_{t_1} = \{s_2, s_3, s_4\}$. Starting and ending time of tasks give $\mathcal{C} = \{\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3\}$ with $\mathcal{K}_1 = \{t_1, t_2, t_3\}$, $\mathcal{K}_2 = \{t_1, t_3, t_4\}$, $\mathcal{K}_3 = \{t_4, t_5\}$. Therefore, $LB_{\neq} = 3$. An optimal solution involving shifts $\{s_1, s_2, s_3\}$ is given in Fig. 1b. This example will be used all along the article to illustrate our points.

2.2. Complexity

The SMPTSP is similar to many problems such as the Fixed Job Scheduling Problem [23,24], the Interval Scheduling Problem [34,35], the Class-Room Assignment Problem [12], the Graph Colouring Problem [28] and the List Colouring Problem [18]. Therefore, many complexity results of the literature are relevant to our case study. For instance, when shifts are



(a) Input data.

Task	Shift
t_1	s_2
t_2	s_3
t_3	s_1
t_4	s_3
t_5	s_1

(b) Optimal solution with three shifts.

Fig. 1. A basic example with 5 shifts and 5 tasks.

identical ($\forall t \in \mathcal{T}, \mathcal{S}_t = \mathcal{S}$), the SMPTSP can be seen as a Graph Colouring Problem,[1] where vertices correspond to tasks, shifts correspond to colours and two vertices are connected whenever their corresponding tasks overlap in time. While this problem is NP-hard in general, it is polynomial for interval graphs, which is our case because the graph is generated from a set of time intervals (each task is associated with a fixed time interval) [25]. Nevertheless, the SMPTSP generally involves restrictions over allowed shift-task assignments, *i.e.*, shifts are not identical. The previously mentioned colouring problem has to be changed so that not every colour can be taken by every vertex. Instead, each vertex is associated with a subset of feasible colours. This problem is known as the List Colouring Problem [18], and it is NP-hard, even for interval graphs [25]. Therefore, the complexity of the SMPTSP stems from shift heterogeneity.

2.3. Mathematical formulation

The SMPTSP may be stated in Mathematical Programming by using binary variables $x_{t,s}$ and y_s, which respectively specify if the task t is assigned to the shift s and if the shift s is used. Based on these variables, the number of used shifts is given by (1) and the assignment of tasks to compatible shifts is ensured by (2). The purpose of the constraint (3) is twofold: First, it prevents shifts to be assigned to overlapping tasks. Second, it ensures that shifts assigned to at least one task are counted as used. We refer to this model as *MIP model*:

$$\textit{minimise} \quad \sum_{s \in \mathcal{S}} y_s \tag{1}$$

$$\textit{subject to:} \quad \sum_{s \in \mathcal{S}_t} x_{t,s} = 1, \quad \forall t \in \mathcal{T} \tag{2}$$

$$\sum_{t \in \mathcal{K}} x_{t,s} \leq y_s, \quad \forall s \in \mathcal{S}, \forall \mathcal{K} \in \mathcal{C} \tag{3}$$

$$x_{t,s} \in \{0,1\}, \quad \forall t \in \mathcal{T}, \forall s \in \mathcal{S}_t \tag{4}$$

$$y_s \in \{0,1\}, \quad \forall s \in \mathcal{S} \tag{5}$$

3. A CP model based on the `AtMostNValue` constraint

This section first introduces a straightforward CP formulation of the SMPTSP. Next, it recalls the former `AtMostNValue` propagator our approach is based on, and provides a new formalism to define propagators of the same family. Then, it introduces a new propagator which filters `AtMostNValue` while considering a set of difference constraints. We show how to improve and diversify its impact on variables. Finally, we discuss the case of dynamic difference constraints and provide some implementation guidelines.

3.1. A straightforward CP formulation

The SMPTSP can be formulated within CP with a set of $|\mathcal{T}|$ integer variables $\mathcal{X}$ and one objective variable z. $\mathcal{X}$ represents task-shift assignments: More precisely, if we note $d(x)$ the domain of a variable $x \in \mathcal{X}$, then $d(x_i) = \{j\}$ means that task t_i is assigned to shift s_j and $d(z)$ gives the number of shifts assigned to at least one task. We refer to this model as *CP model*:

$$\textit{minimise} \quad z \tag{6}$$

$$\textit{subject to:} \quad \texttt{AllDifferent}\big(\{x_i \mid i \in \mathcal{K}\}\big), \quad \forall \mathcal{K} \in \mathcal{C} \tag{7}$$

$$\texttt{AtMostNValue}(\mathcal{X}, z) \tag{8}$$

[1] We recall that the Graph Colouring Problem consists of assigning a minimum number of colours to vertices of a graph, such that adjacent vertices have distinct colours.

$$\textit{initial domains:} \quad d(x_i) = \{j \in \mathcal{S}_{t_i}\}, \qquad \forall x_i \in \mathcal{X} \tag{9}$$

$$d(z) = \left[LB_{\neq}, |\mathcal{S}|\right] \tag{10}$$

The expressive language offered by CP enables to model the problem through two global constraints. In (7), `AllDifferent` constraints [50] are used to forbid the assignment of overlapping tasks to the same shift. In (8), the `AtMostNValue` constraint [7] is used to restrict the number of shifts that are involved in the schedule. Then, variable initial domain definitions are given by (9) and (10). Trivial lower and upper bounds for z are respectively the maximum number of overlapping tasks ($LB_{\neq}$, the number of variables in the largest `AllDifferent` constraint) and the number of available shifts.

From a sementical point of view, constraints (7) are equivalent to one single `SomeDifferent` constraint. This constraint has been introduced in [51] to model a set of disequalities, in order to solve a workforce management problem [2]. A propagator establishing the Generalised Arc Consistency (GAC) on `SomeDifferent` has been introduced in [51]. Since its worst case runtime complexity is $O(n^3\beta^n)$, with $\beta \approx 3.5$, Richter et al. suggests various modifications to improve its runtime on their industrial benchmarks [2,51]. Nevertheless, their study shows that their algorithm is not suited to instances whose disequalities are numerous and form a graph which consists of few big connected components. This is unfortunately the case of SMPTSP instances, for which the graph of overlapping tasks is presumably connected. Furthermore, as `AllDifferent` constraints are detected in polynomial time, then *CP model* already has a good propagation of disequalities. Therefore, we do not use `SomeDifferent` in *CP model*.

To get a higher level of abstraction, we now consider the SMPTSP as the problem of assigning a minimum number of values to variables, which are subject to some difference constraints.

3.2. State-of-the-art filtering of the `AtMostNValue` constraint

The `AtMostNValue` constraint belongs to the *Number of Distinct Values* constraint family [3]. It has been introduced in [48] to specify music programs but the first filtering algorithm was provided in [3]. Then, the `AtMostNValue` constraint has been widely investigated in [7], where the authors proved that enforcing the GAC on `AtMostNValue` is NP-hard, and provide various filtering algorithms. According to this study, the *greedy* propagator they introduced provides a good tradeoff between filtering and runtime. Thus, we use it as the reference propagator for filtering the `AtMostNValue` constraint. Before describing this propagator we need to recall a few definitions:

Definition 1. The *intersection graph* of a set of variables $\mathcal{X}$, $G_{\mathcal{I}}(\mathcal{X}) = (V, E_{\mathcal{I}})$, is defined by a vertex set V where each variable $x_i \in \mathcal{X}$ is associated with a vertex $i \in V$, and an edge set $E_{\mathcal{I}}$ representing domain intersections: For any $(i, j) \in V^2$, there is an edge $(i, j) \in E_{\mathcal{I}}$ if and only if $d(x_i) \cap d(x_j) \neq \emptyset$.

Definition 2. An *independent set* in a graph $G = (V, E)$ is a subset, $A \subseteq V$, of disjoint vertices, *i.e.*, for any $(i, j) \in A^2$ such that $i \neq j$, $(i, j) \notin E$.

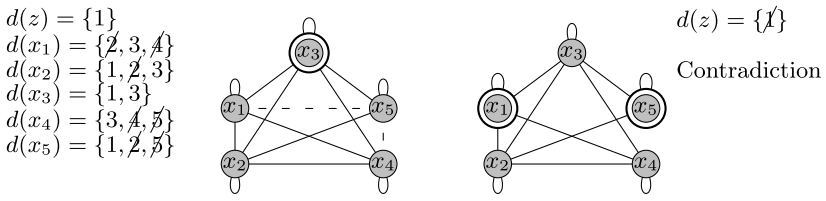
Definition 3. A *maximum independent set* in a graph G is an independent set whose cardinality is maximum. The *independence number* of a graph G, noted $\alpha(G)$, is the cardinality of a maximum independent set in G. The set of all independent sets in a graph G, is referred to as $\mathcal{IS}(G)$.

The filtering algorithm proposed in [7] stems from the search of a maximum independent set in $G_{\mathcal{I}}(\mathcal{X})$. Since this problem is NP-hard [27], it actually computes an independent set A in $G_{\mathcal{I}}(\mathcal{X})$, in a greedy way, by selecting nodes of minimum degree first [31]. This heuristic is referred to as `MD`. Then, the propagator filters according to the following rules:

- $\mathcal{R}_1$: $\underline{z} \leftarrow max(\underline{z}, |A|)$
- $\mathcal{R}_2$: $|A| = \bar{z} \Rightarrow \forall i \in V, d(x_i) \leftarrow d(x_i) \cap \bigcup_{a \in A} d(x_a)$

Where $\underline{z}$ and $\bar{z}$ respectively refer to the lower bound and the upper bound of the variable z. $\mathcal{R}_1$ states that the cardinality of A is a valid lower bound for z. $\mathcal{R}_2$ states that whenever the cardinality of the independent set A is equal to the upper bound of z, then variables in $\mathcal{X}$ have to take their values among the subset of values induced by A. Indeed, variables associated with an independent set of an intersection graph take different values, by definition. Thus, using a value outside of this subset of values would lead to use at least $|A| + 1$ values, which is a contradiction.

Regarding the time complexity of these filtering rules, $\mathcal{R}_1$ clearly runs in constant time whereas $\mathcal{R}_2$ requires a finer study. Let the number of variables and the number of values be respectively referred to as $n = |\mathcal{X}|$ and $l = |\mathcal{S}|$. We assume that the intersection, as well as the union, of two variable domains can be computed in $O(\log(l))$ worst case time with bit vector operations. The computation of $\bigcup_{a \in A} d(x_a)$ requires at most n domain unions. As it does not depend on i, it can be computed once and for all in $O(n \log(l))$ worst case time. Then, n domain intersections are performed. Thus, $\mathcal{R}_2$ is propagated in $O(n \log(l))$ worst case time.



(a) First propagation (b) Second propagation

Fig. 2. Applying $\texttt{AMNV}\langle G_{\mathcal{I}}|\texttt{MD}|\mathcal{R}_{1,2}\rangle$ to our example when $d(z) = \{1\}$.

Overall, the greedy propagator of `AtMostNValue` takes a graph G as input, calls a function F to compute independent sets in G and then filters variable domains with a set of rules $\mathcal{R}$. Therefore, we introduce the notation $\texttt{AMNV}\langle G|F|\mathcal{R}\rangle$ to define such a family of propagators. Consequently, the greedy propagator introduced in [7] is referred to as $\texttt{AMNV}\langle G_{\mathcal{I}}|\texttt{MD}|\mathcal{R}_{1,2}\rangle$. In the following, we suggest improvement for G, F and $\mathcal{R}$, leading to a new propagator which filters `AtMostNValue` and a set of difference constraints.

To illustrate the state-of-the-art propagator, we now apply it to our running example in Fig. 2, with $d(z) = \{1\}$. This state might occur during the resolution. Because of variable domain definitions, the intersection graph corresponding to our example, is a complete graph. Thus, `MD` selects only one node, x_3 for instance. Next, $\mathcal{R}_1$ states that the number of values is at least one, which provides no information because $d(z) = \{1\}$. Then, $\mathcal{R}_2$ states that values 2, 4 and 5 must be removed from the domain of the variables (Fig. 2a). Consequently, the edges (x_1, x_5) and (x_5, x_4) have to be removed in order to obtain the new intersection graph. Based on this new graph, if we assume that x_1 and x_5 are then used as a new independent set (Fig. 2b) then $\mathcal{R}_1$ states that the number of values is at least two, leading to fail.

3.3. Embedding difference constraints into `AtMostNValue`

As the SMPTSP only considers two kinds of constraints (`AtMostNValue` and `AllDifferent`), filtering their conjunction may be very profitable. For that purpose, this section introduces an implied propagator, in the form $\texttt{AMNV}\langle G|F|\mathcal{R}\rangle$, which considers difference constraints. This work stems from the very simple observation that, if a disequality states that two variables x_i and x_j of $\mathcal{X}$ must take different values, then for every solution, there is no edge between vertex i and vertex j in $G_{\mathcal{I}}(\mathcal{X})$. However, such an edge might exist during search. If so, then it would presumably lower the propagation strength, because the more edges in $G_{\mathcal{I}}(\mathcal{X})$, the smaller independent sets in $G_{\mathcal{I}}(\mathcal{X})$. Hence, we should remove those edges as soon as possible. Consequently, we introduce the *constrained intersection graph* (Definition 4), to be used instead of the intersection graph of variables.

Definition 4. Given a set of variables $\mathcal{X}$ and a set of difference constraints $\mathcal{D}$, the *constrained intersection graph*, $G_{\mathcal{CI}}(\mathcal{X}, \mathcal{D}) = (V, E_{\mathcal{CI}})$, of $\mathcal{X}$ and $\mathcal{D}$ is defined by a vertex set V where each variable $x_i \in \mathcal{X}$ is associated with a vertex $i \in V$, and an edge set $E_{\mathcal{CI}}$ representing possible classes of equivalence: For any $(i, j) \in V^2$, there is an edge $(i, j) \in E_{\mathcal{CI}}$ if and only if $d(x_i) \cap d(x_j) \neq \emptyset$ and $\texttt{neq}(x_i, x_j) \notin \mathcal{D}$.

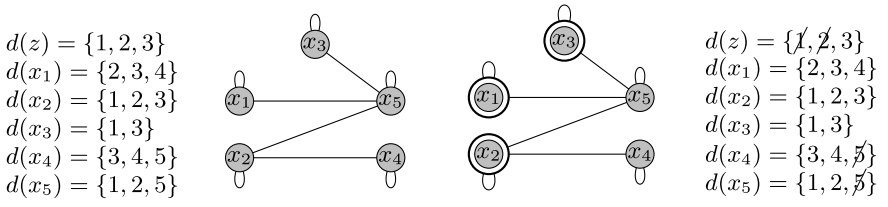
In this paper, we consider a single set of variables $\mathcal{X}$ and a single set of difference constraints $\mathcal{D}$, thus, for the sake of clarity $G_{\mathcal{CI}}(\mathcal{X}, \mathcal{D})$ and $G_{\mathcal{I}}(\mathcal{X})$ will be respectively noted $G_{\mathcal{CI}}$ and $G_{\mathcal{I}}$. $G_{\mathcal{CI}}$ can be computed by removing from $G_{\mathcal{I}}$ any edge which is associated with a disequality. It is worth noticing that $G_{\mathcal{CI}} \subseteq G_{\mathcal{I}}$.

Proposition 1. *$\mathcal{IS}(G_{\mathcal{I}}) \subseteq \mathcal{IS}(G_{\mathcal{CI}})$, hence $\alpha(G_{\mathcal{I}}) \leq \alpha(G_{\mathcal{CI}})$.*

Proof. Let $A_{\mathcal{I}}$ be an independent set in $G_{\mathcal{I}}$. Since $G_{\mathcal{CI}}$ and $G_{\mathcal{I}}$ are based on the same variable set, they share the same vertex set, so $A_{\mathcal{I}}$ is also a subset of vertices of $G_{\mathcal{CI}}$. Since vertices of $A_{\mathcal{I}}$ are pairwise disjoint in $G_{\mathcal{I}}$ (by assumption) and since all edges of $G_{\mathcal{CI}}$ also belong to $G_{\mathcal{I}}$, then vertices of $A_{\mathcal{I}}$ are also pairwise disjoint in $G_{\mathcal{CI}}$. Consequently, $A_{\mathcal{I}}$ is an independent set in $G_{\mathcal{CI}}$. Thus, all independent sets in $G_{\mathcal{I}}$ are independent sets in $G_{\mathcal{CI}}$, so $\max_{I \in \mathcal{IS}(G_{\mathcal{I}})} |I| \leq \max_{I_c \in \mathcal{IS}(G_{\mathcal{CI}})} |I_c|$, hence $\alpha(G_{\mathcal{I}}) \leq \alpha(G_{\mathcal{CI}})$. □

Note that a maximum independent set in $G_{\mathcal{I}}$, is not necessarily maximal in $G_{\mathcal{CI}}$. For instance, one may consider a non-empty set of variables with identical domains and with a difference constraint over each pair of distinct variables. Then $G_{\mathcal{I}}$ is a complete graph, whereas there are no edges, but loops, in $G_{\mathcal{CI}}$. Consequently, $\alpha(G_{\mathcal{I}}) = 1$ whereas $\alpha(G_{\mathcal{CI}}) = |V|$. It is worth noticing that in our context, the bigger the independent set, the higher the chance to filter variable domains. Thus, using $G_{\mathcal{CI}}$ is presumably better than using $G_{\mathcal{I}}$ to filter `AtMostNValue` when difference constraints figure in the model (Proposition 2).

To illustrate the interest of $G_{\mathcal{CI}}$, we now use it on our example, with $d(z) = \{1, 2, 3\}$, in Fig. 3. Because of difference constraints, $G_{\mathcal{CI}}$ is sparser than $G_{\mathcal{I}}$ (Fig. 3a). Thus, `MD` is now able to compute a larger independent set, leading to find a



(a) Constrained intersection graph (b) Filtering from an independent set

Fig. 3. Use of $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}|\mathcal{R}_{1,2}\rangle$ on our example.

better lower bound for z. For instance, if we consider the independent set $\{x_1, x_2, x_3\}$ (Fig. 3b), then $\mathcal{R}_1$ and $\mathcal{R}_2$ respectively state that the number of values is at least three and that the value 5 can be removed from the variable domains.

Following [15], a propagator A is stronger than a propagator B if and only if, in any propagation, any value filtered by B is also filtered by A. Furthermore, A is strictly stronger than B if and only if A is stronger than B and there exists at least one problem instance for which a value is filtered by A and not by B, during a propagation step.

Proposition 2. *Given an oracle* $\texttt{O}$ *which computes all maximum independent sets of any graph, then* $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{O}|\mathcal{R}_{1,2}\rangle$ *is strictly stronger than* $\texttt{AMNV}\langle G_{\mathcal{I}}|\texttt{O}|\mathcal{R}_{1,2}\rangle$.

Proof. First of all, since $\texttt{O}$ is able to compute all maximum independent sets of any graph, then the lower bound given by $\mathcal{R}_1$ in $G_{\mathcal{CI}}$ is equal to $\alpha(G_{\mathcal{CI}})$ whereas the lower bound given by $\mathcal{R}_1$ in $G_{\mathcal{I}}$ is equal to $\alpha(G_{\mathcal{I}})$. Since $\alpha(G_{\mathcal{I}}) \leq \alpha(G_{\mathcal{CI}})$ (Proposition 1), then $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{O}|\mathcal{R}_1\rangle$ is stronger than $\texttt{AMNV}\langle G_{\mathcal{I}}|\texttt{O}|\mathcal{R}_1\rangle$. Second, since $\texttt{O}$ is able to compute all maximum independent sets of any graph, and since $\mathcal{IS}(G_{\mathcal{I}}) \subseteq \mathcal{IS}(G_{\mathcal{CI}})$ (Proposition 1), then values filtered by $\texttt{AMNV}\langle G_{\mathcal{I}}|\texttt{O}|\mathcal{R}_2\rangle$ are also filtered by $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{O}|\mathcal{R}_2\rangle$. Consequently, $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{O}|\mathcal{R}_2\rangle$ is stronger than $\texttt{AMNV}\langle G_{\mathcal{I}}|\texttt{O}|\mathcal{R}_2\rangle$. Examples given on Figs. 2 and 3 illustrate a case where $\alpha(G_{\mathcal{CI}}) > \alpha(G_{\mathcal{I}})$. Therefore $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{O}|\mathcal{R}_{1,2}\rangle$ is strictly stronger than $\texttt{AMNV}\langle G_{\mathcal{I}}|\texttt{O}|\mathcal{R}_{1,2}\rangle$. □

Proposition 3. *Given an independent set* A *in* $G_{\mathcal{CI}}$ *such that* $|A| = \bar{z}$, *any solution of the conjunction of* $\texttt{AtMostNValue}$ *and* $\mathcal{D}$ *satisfies the following formula:* $\forall i \in V \setminus A, \exists a \in A_i$ *s.t.* $x_i = x_a$, *where* A_i *denotes* $\{a \in A \mid (i, a) \in E_{\mathcal{CI}}\}$.

Proof. Given an independent set A in $G_{\mathcal{CI}}$ such that $|A| = \bar{z}$. Let's assume that there exists a solution S to the conjunction of $\texttt{AtMostNValue}$ and $\mathcal{D}$ such that there exists a vertex $i \in V \setminus A$ for which $\forall a \in A_i, x_i \neq x_a$. Thus, S is solution of the conjunction of $\texttt{AtMostNValue}$ and $\mathcal{D} \cup \{\texttt{neq}(x_i, x_a) \mid a \in A_i\}$. Consequently, $A \cup \{i\}$ is a valid independent set in $G_{\mathcal{CI}}$. Then $\mathcal{R}_1$ states that $\underline{z} \leftarrow |A \cup \{i\}|$, *i.e.*, $\underline{z} \leftarrow \bar{z} + 1$ which is not possible. Consequently, such a solution S does not exist, hence Proposition 3 holds. □

From a filtering perspective, Proposition 3 leads to consider the following rule:

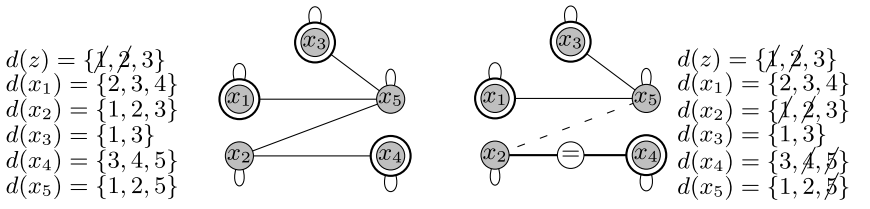
- $\mathcal{R}_3$: $|A| = \bar{z} \Rightarrow \forall i \in V \setminus A \begin{cases} d(x_i) \leftarrow d(x_i) \cap \bigcup_{a \in A_i} d(x_a) \\ A_i = \{a\} \Rightarrow d(x_a) \leftarrow d(x_a) \cap d(x_i) \end{cases}$

This rule is actually a refined variant of $\mathcal{R}_2$. While this change is quite simple, it may have a significant impact in practice, especially on large scale problems were, for any $i \in V \setminus A$, $|A_i|$ is presumably small compared to $|A|$. Note the particular case that occurs when, for some node $i \in V \setminus A$, $A_i = \{a\}$ enables to learn the valid equality $x_i = x_a$. This way, it is possible to filter the domain of the variable x_a of the independent set as well. From a theoretical point of view $\mathcal{R}_3$ is also stronger than $\mathcal{R}_2$ (Proposition 4). However, such filtering comes at a price. As it depends on i, $\bigcup_{a \in A_i} d(x_a)$ must be computed for every $i \in V \setminus A$. This raises the worst case time complexity of $\mathcal{R}_3$ to $O(n^2 \log(l))$.

To illustrate this rule, we now apply it on our example, with $d(z) = \{1, 2, 3\}$, in Fig. 4. If we consider the independent set $\{x_1, x_3, x_4\}$, then $\mathcal{R}_1$ deduces that $d(z) = \{3\}$ but $\mathcal{R}_2$ cannot filter $\mathcal{X}$ domains (Fig. 4a), because the set of values induced by the independent set is $\{1, 2, 3, 4, 5\}$. However, $\mathcal{R}_3$ (Fig. 4b) enables to filter the value 5, which is not included in $d(x_1) \cup d(x_3) = \{1, 2, 3, 4\}$, from the domain of x_5. It also removes values 1 and 2, which do not figure in $d(x_4) = \{3, 4, 5\}$, from the domain of x_2. Finally, it learns the valid equality $x_2 = x_4$, which allows to remove values 4 and 5 from the domain of x_4.

Proposition 4. *Given a deterministic heuristic* $\texttt{H}$ *which computes an independent set* A *in* $G_{\mathcal{CI}}$, *then* $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_3\rangle$ *is strictly stronger than* $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_2\rangle$.

Proof. Since $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_3\rangle$ and $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_2\rangle$ use the same deterministic heuristic $\texttt{H}$ in the same graph $G_{\mathcal{CI}}$, then they filter with the same independent set A. Since, for any node $i \in V$, $A_i = \{a \in A \mid (i, a) \in E_{\mathcal{CI}}\}$, then $A_i \subseteq A$. Consequently, any value filtered by $\mathcal{R}_2$ is also filtered by $\mathcal{R}_3$. The example given on Fig. 4 illustrates a case where $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_3\rangle$ filtrates more than $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_2\rangle$. Therefore, $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_3\rangle$ is strictly stronger than $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{H}|\mathcal{R}_2\rangle$. □



(a) Filtering with $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}|\mathcal{R}_{1,2}\rangle$ (b) Filtering with $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}|\mathcal{R}_{1,3}\rangle$

Fig. 4. $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}|\mathcal{R}_{1,2}\rangle$ *versus* $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}|\mathcal{R}_{1,3}\rangle$, on our example.

3.4. Diversifying filtering

CP frameworks traditionally perform a fix point over constraints at each branching node [54]. This implies that our model may compute thousands of independent sets during the search process. Thus, it is advised to use a greedy algorithm, such as MD, to filter the AtMostNValue constraint [7]. This heuristic is efficient but it lacks diversification for both its bound and the resulting filtering, which depend on the computed independent set. If we assume ties (of equal degree vertices) are broken in a lexicographic way, then MD is deterministic. This may lead to unfortunate results when the considered graph does not suit this heuristic.

Proposition 5. *Given two functions* $\texttt{F}_1$ *and* $\texttt{F}_2$ *which compute a set of independent sets in* $G_{\mathcal{CI}}$*, respectively noted* $\mathcal{A}_1$ *and* $\mathcal{A}_2$*. If* $\texttt{F}_1$ *and* $\texttt{F}_2$ *are such that, for any* $\mathcal{A}_1$ *induced by* $\texttt{F}_1$ *and for any* $\mathcal{A}_2$ *induced by* $\texttt{F}_2$*,* $\max_{I_2\in\mathcal{A}_2\setminus\mathcal{A}_1}|I_2| < \max_{I_1\in\mathcal{A}_1}|I_1|$*, then* $\mathit{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_1|\mathcal{R}_{1,3}\rangle$ *is strictly stronger than* $\mathit{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_2|\mathcal{R}_{1,3}\rangle$*.*

Proof. First, $\max_{I_2\in\mathcal{A}_2\setminus\mathcal{A}_1}|I_2| < \max_{I_1\in\mathcal{A}_1}|I_1|$ implies that $\max_{I_2\in\mathcal{A}_2}|I_2| \leq \max_{I_1\in\mathcal{A}_1}|I_1|$, hence $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_1|\mathcal{R}_1\rangle$ is stronger than $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_2|\mathcal{R}_1\rangle$. Second, let's now assume that a value is removed from a domain of a variable in $\mathcal{X}$ by $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_2|\mathcal{R}_3\rangle$. If this filtering occurs while considering an independent set of $\mathcal{A}_1 \cap \mathcal{A}_2$, then the same filtering is performed by $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_1|\mathcal{R}_{1,3}\rangle$. Else, this filtering occurs when considering an independent set I_2 of $\mathcal{A}_2 \setminus \mathcal{A}_1$. A necessary condition so that $\mathcal{R}_3$ triggers filtering is that, $|I_2| = \bar{z}$. As $|I_2| < \max_{I_1\in\mathcal{A}_1}|I_1|$, then the problem is actually infeasible, which is captured by $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_1|\mathcal{R}_1\rangle$. Thus, $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_1|\mathcal{R}_{1,3}\rangle$ is stronger than $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_2|\mathcal{R}_{1,3}\rangle$. Let $\texttt{F}_1 = \texttt{MD}$ and $\texttt{F}_2 = \texttt{FV}$ respectively denote the minimum degree heuristic and a (bad) heuristic which selects only the first vertex. We assume ties are broken in a lexicographic way, *i.e.*, in case of a complete graph, both heuristics output the same singleton independent set, hence both approaches are equivalent. If the graph is not complete, then there exists at least a vertex pair $(u, v) \in G_{\mathcal{CI}}$ with no edge, hence any independent set computed by MD will have at least two vertices. Thus, $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}|\mathcal{R}_{1,3}\rangle$ is strictly stronger than $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{FV}|\mathcal{R}_{1,3}\rangle$. Therefore, if $\texttt{F}_1$ and $\texttt{F}_2$ are such that, for any $\mathcal{A}_1$ induced by $\texttt{F}_1$ and for any $\mathcal{A}_2$ induced by $\texttt{F}_2$, $\max_{I_2\in\mathcal{A}_2\setminus\mathcal{A}_1}|I_2| < \max_{I_1\in\mathcal{A}_1}|I_1|$, then $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_1|\mathcal{R}_{1,3}\rangle$ is strictly stronger than $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{F}_2|\mathcal{R}_{1,3}\rangle$. □

As suggested in [3,7], and highlighted by Proposition 5, it may be interesting to obtain several independent sets to apply filtering rules over a greater set of variables. In particular, if all maximal independent sets were known, $\mathcal{R}_{1,3}$ could then be used optimally, *i.e.*, all values that could be filtered by $\mathcal{R}_{1,3}$ would actually be filtered. One way to find all maximal independent sets is to adapt the algorithm of Bron–Kerbosch, an exact algorithm which computes all maximal cliques in an undirected graph [10]. However, as the underlying problem is NP-hard, performing the Bron–Kerbosch algorithm at each propagation would presumably be very time consuming. On our large data, it turned out that the improved filtering provided by this algorithm is not worth the overhead. Therefore, we suggest to keep the philosophy of MD and to get diversification through a fast algorithm. The simplest way would be to randomly break ties of MD. However, this does not bring enough diversification to improve results. Thus, we introduce the $\texttt{R}^k$ algorithm (Algorithm 1).

The $\texttt{R}^k$ heuristic performs k iterations, each one computes an independent set randomly. Each independent set is computed by successively selecting a node randomly and removing it along with its neighbours, until all nodes are removed. It thus provides a set of k independent sets, each one being maximal under inclusion. However, these independent sets are not necessarily all distincts (random node selections might lead to same results). As it relies on randomness, such propagator is neither idempotent nor monotonic. From a theoretical point of view, this approach presents several interesting properties. First, it offers control over its runtime complexity and its expected quality. Second, computing independent sets randomly tends to impact variable domains homogeneously. Note that, when $k \to \infty$, then the method tends to enumerate and filter with all maximum independent sets, which is optimal with the given filtering rules. However, from a practical point of view, one has to bound the number of iterations in order to get a reasonable runtime. We suggest to keep k between 20 and 100 as a default setting, which seems to perform better than MD, on average, without introducing a significant overhead. Regarding the probability distribution of $\texttt{R}^k$, two options may be considered: A uniform distribution or a weighted distribution where small-degree vertices are more likely to be selected. The second option may be seen as a random adaptation of MD. Based on practical experiments, and because MD offers a good approximation ratio [31], we recommend to use a uniform-distribution $\texttt{R}^k$ together with MD.

Algorithm 1 $\mathtt{R}^k$ algorithm to compute independent sets.

Require:
$G_{\mathcal{CI}}$, the constrained intersection graph of which independent sets have to be computed
k, the number of iterations (and thus the number of independent sets to compute)

$\mathcal{A} \leftarrow \emptyset$ // Creates a set of independent sets, initially empty
$count \leftarrow k$
while $(count \geq 0)$ **do**
  $count \leftarrow count - 1$
  $G \leftarrow G_{\mathcal{CI}}.copy()$ // Copies the constrained intersection graph
  $A \leftarrow \emptyset$ // Creates an independent set A, initially empty
  while $(G \neq \emptyset)$ **do**
    $x \leftarrow randomNode(G)$ // Randomly selects a node x in G
    $A \leftarrow A \cup \{x\}$// Adds x to the independent set A
    $G \leftarrow G \setminus \{y \mid (x, y) \in G\}$ // Removes x's neighbours from G (including x itself)
  end while
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{A\}$ // adds the independent set A to $\mathcal{A}$
end while
return $\mathcal{A}$ // returns a set of independent sets in $G_{\mathcal{CI}}$

3.5. The case of dynamic difference constraints

Our approach focuses on the SMPTSP which considers a set of tasks that are already fixed in time. Thus, difference constraints are given as input through a set of `AllDifferent` constraints. However, one may be interested in filtering `AtMostNValue` when difference constraints appear dynamically during the search. Let us consider a generalisation of the SMPTSP for which tasks are no longer fixed in time but have a time window instead. In such problem, disequalities implicitly appear as time variable domain reductions induce task overlaps. It is worth noticing that in this situation, difference constraints would no longer be well propagated because of the absence of `AllDifferent` constraints in the model. Fortunately, the `AtMostNValue` propagator can help to get back a global view of the problem and thus a powerful filtering thanks to Proposition 6.

Proposition 6. *If A is an independent set in $G_{\mathcal{CI}}$, let $\mathcal{X}_A$ denote $\{x_i \in \mathcal{X} \mid i \in A\}$, then the constraint* $\mathtt{AllDifferent}(\mathcal{X}_A)$ *holds.*

Proof. By definition of a constrained intersection graph, an independent set represents variables that are either already different or constrained to be. □

From Proposition 6, we derive a new filtering rule for `AtMostNValue` which calls a filtering algorithm of `AllDifferent` over variables corresponding to the independent set A it has computed:

- $\mathcal{R}_4$: $\mathtt{AllDifferent}(\{x_i \in \mathcal{X} \mid i \in A\})$

Note that this rule can either directly call a filtering algorithm of `AllDifferent` over the appropriate subset of variables or post dynamically an `AllDifferent` constraint into the solver. The first option is the simplest one, but cannot filter incrementally, so a bound-consistent (BC) filtering algorithm of `AllDifferent`, such as the one presented in [44], would presumably be more relevant than the GAC one [50]. Note that the subset of variables to be different is lost right after filtering and the next propagation may involve a worse independent set. Instead, the second option can involve incremental GAC `AllDifferent` constraints which would remain in the current search branch. However, since each independent set can post an `AllDifferent` constraint, it has the drawback of leading to a potential explosion over the number of such constraints. Thus, one needs to set up a constraint pool to manage those constraints, by retaining only the most interesting constraints and removing dominated ones. One can see a parallel with *Cut pools* of MIP solvers [1]. For solver maintenance simplicity purposes, we recommend the first option.

Note that $\mathcal{R}_4$ is only relevant when disequalities arise during search. It is useless in the case of the SMPTSP, because all maximal `AllDifferent` constraints are already preprocessed in polynomial time. Overall, $\mathcal{R}_4$ can be seen as a dynamic generalisation of a common good practice, which consists in reformulating a set of disequalities into `AllDifferent` constraints during preprocessing [30].

3.6. Implementation

So far, we have seen that it was possible to tune the original greedy `AtMostNValue` propagator. We now suggest a simple and generic pseudocode (Algorithm 2).

The constrained intersection graph is stored as a backtrackable structure which is updated at the beginning of the filtering algorithm, *i.e.*, after potentially many domain modifications. This turns out to be much faster than updating $G_{\mathcal{CI}}$ incrementally after every domain modification or rebuilding it entirely from scratch. Once the graph has been updated, the

Algorithm 2 AMNV$\langle G_{\mathcal{CI}}|F|\mathcal{R}\rangle$ – filtering algorithm.

Require:
z, an integer variable
$\mathcal{X}$, a set of variables
$\mathcal{D}$, a set of difference constraints
$G_{\mathcal{CI}} = (V, E_{\mathcal{CI}})$, a backtrackable graph
F, a function that computes a set of independent sets in a given graph
$\mathcal{R}$, a set of filtering rules

1: **if** (first call of the propagator) **then**
2: // graph generation
3: $V \leftarrow [1, |\mathcal{X}|]$
4: $E_{\mathcal{CI}} \leftarrow \emptyset$
5: **for** $((i, j) \in V^2)$ **do**
6: **if** $(d(x_i) \cap d(x_j) \neq \emptyset \wedge \mathtt{neq}(x_i, x_j) \notin \mathcal{D})$ **then**
7: $E_{\mathcal{CI}} \leftarrow E_{\mathcal{CI}} \cup \{(i, j)\}$
8: **end if**
9: **end for**
10: $G_{\mathcal{CI}} \leftarrow (V, E_{\mathcal{CI}})$
11: **else**
12: // graph lazy update
13: **for** $((i, j) \in E_{\mathcal{CI}})$ **do**
14: **if** $(d(x_i) \cap d(x_j) = \emptyset \vee \mathtt{neq}(x_i, x_j) \in \mathcal{D})$ **then**
15: $E_{\mathcal{CI}} \leftarrow E_{\mathcal{CI}} \backslash \{(i, j)\}$
16: **end if**
17: **end for**
18: **end if**
19: $\mathcal{A} \leftarrow F(G_{\mathcal{CI}})$ // computes a set $\mathcal{A}$ of independent sets
20: **for** $(A \in \mathcal{A})$ **do**
21: *filter*$(\mathcal{X}, z, G_{\mathcal{CI}}, A, \mathcal{R})$ // rule-based filtering
22: **end for**

propagator computes a set of independent sets with a function F. For each independent set, the propagator filters the lower bound of z and filters $\mathcal{X}$ with a set of rules $\mathcal{R}$.

It is worth noticing that no assumption is made on the set of difference constraints $\mathcal{D}$ which can thus grow during the resolution process. For instance, $\mathcal{D}$ can be modified by the user or other constraints. Note also that, as long as domain union and intersection operations are defined, no assumption is made about the kind of variables in $\mathcal{X}$. This means Algorithm 2 can be used to filter the `AtMostNVector` constraint [13], a variant of `AtMostNValue` which holds on continuous vector variables instead of integer variables. Indeed, continuous domains are represented by floating intervals, hence union and intersection operations are straightforward. Overall, one can see that this propagator is both flexible and easy to implement and maintain.

4. Designing a search heuristic for the SMPTSP

This section describes the search process that is used within our CP approach. It embeds some cost-based reasonings (bottom–up optimisation strategy and tailored value selection), domain reduction reasonings (heuristic symmetry breaking and variable selection) and finally a bit of learning (throught `last-conflict` [41]).

4.1. Optimisation strategy

The classical optimisation strategy of a CP solver consists of enumerating improving solutions. If the search completes, the last solution found is proved to be optimal. This process is known as the *top–down* approach. However, the filtering of AMNV$\langle G|\mathtt{F}|\mathcal{R}\rangle$ is related to the lower bounding of the problem. Therefore, the lower the objective upper bound, the stronger the filtering. Hence, we naturally employed a *bottom–up* minimisation strategy. It tries to compute a solution involving k values, where k is initialised to the root lower bound of z and incremented by one each time the solver proves unsatisfiability. Thus, the first solution found is optimal. This is done by simply branching on the objective variable and assigning it its current lower bound.

The search heuristic that is used over $\mathcal{X}$ is the following: The variable associated with the smallest domain is selected first [32], and then fixed to the first value in its domain that is already assigned to another variable. If no such value exists, then the variable is fixed to its lower bound. Thus, the value selection naturally tends to use few different values. This branching scheme is reinforced by the `last-conflict` heuristic [41] which aims at identifying critical variables.

4.2. When search tries its luck

In [21], we employed a two-step search strategy that first finds a subset of z allowed values and then assign a value to every decision variable $x \in \mathcal{X}$. The first step requires to associate every shift $s_j \in \mathcal{S}$ with a new binary variable $y_j \in \mathcal{Y}$ to tell whether or not the value j must be assigned to a variable in $\mathcal{X}$. However, this approach makes the model heavier because of additional variables and channelling constraints. Moreover, branching on $\mathcal{Y}$ first might not be appropriate for

$d(b_{reif}) = \{0, 1\}$
$d(z) = \{3\}$
$d(x_1) = \{2, 3, 4\}$
$d(x_2) = \{1, 2, 3\}$
$d(x_3) = \{1, 3\}$
$d(x_4) = \{3, 4, 5\}$
$d(x_5) = \{1, 2, 5\}$

(a) Domains before branching on b_{reif}

$d(b_{reif}) = \{\not{0}, 1\}$
$d(z) = \{3\}$
$d(x_1) = \{2, 3, \not{4}\}$
$d(x_2) = \{1, 2, 3\}$
$d(x_3) = \{1, 3\}$
$d(x_4) = \{3, \not{4}, \not{5}\}$
$d(x_5) = \{1, 2, \not{5}\}$

(b) Filtering $\texttt{maximum}(\mathcal{X}) = z$

Fig. 5. Impact of b_{reif}, which reifies the constraint $\texttt{maximum}(\mathcal{X}) = z$.

every instance. If so, compelling the search to follow this scheme may decrease performances. Therefore, while keeping the general idea of a two-step search strategy, we now suggest a lighter and more flexible branching scheme.

It is worth noticing that if shifts were all equivalent, *i.e.*, if values of assignment variables were all interchangeable, then the problem would be polynomial to solve. From a modelling perspective, we could force to use any arbitrary set of z values. More precisely, we could use the first z values. Assuming that values are consecutive and start at one, we could add the constraint $\texttt{maximum}(\mathcal{X}) = z$. This symmetry breaking constraint [26] may dramatically reduce the search space. Unfortunately, values are generally not interchangeable, so we cannot post such a hard constraint directly. Nevertheless, it may occur in some instances that values are *nearly* interchangeable, in the sense that finding a valid subset of values to be used is a minor issue compared to finding a valid variable-value assignment. In other words, if z^* denotes the optimum value, then the problem admits a solution for *numerous* value subsets of size z^*. For this reason, we introduce a binary variable b_{reif} to reify [5] the above symmetry breaking constraint. The search heuristic first fixes b_{reif} to 1 in order to propagate this constraint and reduce the search space [20]. In case no solution exists, a backtrack fixes it to 0 and turns the constraint into its opposite *i.e.*, $\texttt{maximum}(\mathcal{X}) \neq z$, so that the approach remains exact. This is a full reification. However, the filtering of the right branch will be weak, so a half-reification [22] of the constraint $\texttt{maximum}(\mathcal{X}) = z$ would be sufficient to reproduce our approach. Note that the initial set of shifts may be randomly shuffled to handle the case where it is ordered in a specific way. One can see this search trick as a way to perform symmetry breaking from search. However, as the model is unlikely to contain such symmetries, it is rather a heuristic gamble. More precisely, there are $\binom{|S|}{z^*}$ different value sets of the size of the optimal value. Let K be the number of different value sets that belong to at least one optimal solution. We have $0 \leq K \leq \binom{|S|}{z^*}$. The probability of the value set $\{1, 2, ..., z^*\}$ to belong to an optimal solution is therefore $P(b_{reif} = 1) = \frac{K}{\binom{|S|}{z^*}}$.

To illustrate the use of b_{reif} we now apply it on our running example, with $d(z) = \{3\}$ (the optimum) in Fig. 5. For the sake of clarity, the shift set of the example has not been shuffled, hence a value $i \in [1, 5]$ corresponds to the shift s_i. However, in practice, we randomly shuffle the initial shift set to reduce bias, so values 1, 2 and 3 might refer to shifts s_2, s_4 and s_5 for instance. When b_{reif} is set to 1, the constraint $\texttt{maximum}(\mathcal{X}) = z$ leads to remove values 4 and 5 from $\mathcal{X}$ domains. This is a lucky guess, because optimal solutions use the value sets $\{1, 2, 3\}$ and $\{1, 3, 4\}$, *i.e.*, $K = 2$. Therefore, the constraint $\texttt{maximum}(\mathcal{X}) = z$ has a probability to hold on optimal solutions of $\frac{2}{\binom{5}{3}} = 0.2$. This illustrates why it has to be reified.

5. Experimental study

In order to evaluate the interest of our contribution, we perform extensive tests. First, Section 5.1 introduces and motivates a new benchmark data set for the SMPTSP. Next, in Section 5.2, we focus on the objective lower bound quality at root node. Section 5.3 highlights the potential benefit of strengthening and diversifying the filtering. Section 5.4 investigates the interest of adding the reification of a symmetry breaking constraint. Section 5.5 focuses on the scalability issue, *i.e.*, it evaluates the ability of our approach to compute tight bounds, even on large instances. Finally, in Section 5.6 we compare our approach to the best known results on the state-of-the-art SMPTSP instances.

Our algorithms have been implemented in JAVA, we have used Choco 3-1-1 [14] to implement *CP model* and Cplex 12.4 with default settings to implement *MIP model*. Note that we have used the automatic tunning tool of Cplex to get the best setting, which turned out to be the default one. All our implementations are available online at [40]. In the following we respectively note z^* and $\underline{z}_r$ the optimal objective and the objective lower bound at root node. Finally, a CP model using a propagator $\texttt{AMNV}\langle G|\texttt{F}|\mathcal{R}\rangle$, is noted $\downarrow\texttt{AMNV}\langle G|\texttt{F}|R\rangle$ when used within a top–down minimisation strategy and $\uparrow\texttt{AMNV}\langle G|\texttt{F}|\mathcal{R}\rangle$ when used within a bottom–up minimisation strategy.

5.1. A new set of challenging SMPTSP instances

Very recently, Smet et al. have shown in [56] that the 137 instances provided by Krishnamoorthy and Ernst in [37] admit a feasible solution with an objective value equal to $LB_{\neq}$. Since worker skills are taken into account in [37], this result is somehow surprising: One may expect that considering worker skills would have an impact on the optimum, but actually, it

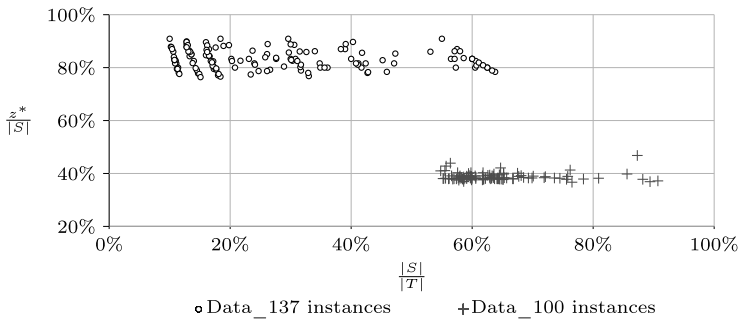


Fig. 6. Comparison of instance input and output characteristics.

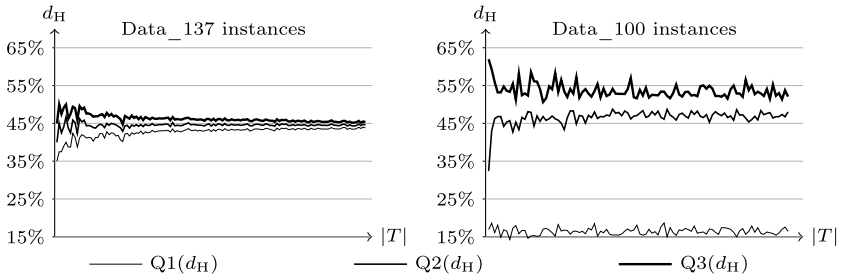


Fig. 7. Homogeneity of shifts (measured with quartiles) depending on instances, sorted by increasing number of tasks. d_H: normalised Hamming distance.

only makes it harder to find a feasible solution. Consequently, these instances do not provide much challenge regarding to the search of interesting lower bounds. Therefore, we propose a new set of challenging instances whose maximal number of overlapping tasks does not provide a good lower bound.

In order to generate this new benchmark we use a dedicated procedure based on some of the empirical results presented in [37] and [56]. First of all, it is specified in [37] that the average tightness, defined as the sum of processing times over the sum of shift lengths should be close to 90% in order to obtain challenging instances. Another hardness analysis [56] shows that the smaller the average task processing time, the more difficult instances are to solve. Based on these two conclusions, we designed a dedicated procedure able to generate a new set of challenging instances. The procedure which is given in details in [40] generates randomly a set of tasks ranging from 15 minutes to 2 hours. We consider random skills and six different time windows to generate shifts. The first three aim at splitting a working day into 8 hours time intervals, which is very common in personnel scheduling [57]. The other three are obtained from the previous ones by introducing an offset to their starting time, so that each task is entirely contained in at least one time interval. Based on this simple procedure, we provide 100 new instances with a number of tasks ranging from 70 to 1600 and a number of available shifts ranging from 60 to 950. These instances along with our generator are available online [40]. In the following, we refer to the instances of Krishnamoorthy et al. as Data_137, and our generated instances as Data_100. In order to apprehend the differences between Data_137 and Data_100, we now provide some comparisons between those two data sets.

As a first step, we show the ratios $\frac{|S|}{|T|}$ and $\frac{z^*}{|T|}$ on every instance (represented as a mark) in Fig. 6. Both $|S|$ and $|T|$ are input data, whereas z^* is known (or at least well approximated) after resolution. This enables to highlight both input and output characteristics of these instances. As can be seen, the two instance sets form two disjoint clusters. It is interesting to notice that instances of the same data set can then be split into sub-clusters, revealing some patterns in their generator. The main difference between the two data sets is related to $\frac{z^*}{|T|}$, which has an average value of 39% and 83%, respectively for Data_100 and Data_137. Regarding input data only, $\frac{|S|}{|T|}$ is 2.2 times bigger in Data_100 than in Data_137. Thus, Data_100 instances have on average more available shifts per task than Data_137, but they use a significantly smaller percentage of these shifts in optimal solutions. Therefore, finding a shift set which corresponds to an optimal solution may be more difficult.

As a second step, we investigate the homogeneity of shifts. In order to evaluate this characteristic, we use a normalised Hamming distance: Given an instance i, the distance $d_H(s_1, s_2)$ between two shifts s_1 and s_2 is equal to the number of tasks that belong to one and only one of the shifts, divided by the total number of tasks within i. To measure the homogeneity of shifts we compute this distance for every pair of shifts in all instances. The analysis of these data relies then on the use of quartiles. As illustrated in Fig. 7, it turned out that the median distance between shifts is on average relatively similar from Data_137 (44.5%) to Data_100 (46.6%). However, the first and third quartiles of this distance are on average quite different from one data set to another: Many shifts in Data_100 are either very similar or quite different, whereas shifts in Data_137 are always a bit different. This difference of homogeneity mainly comes from the use of 8 hours shifts within Data_100:

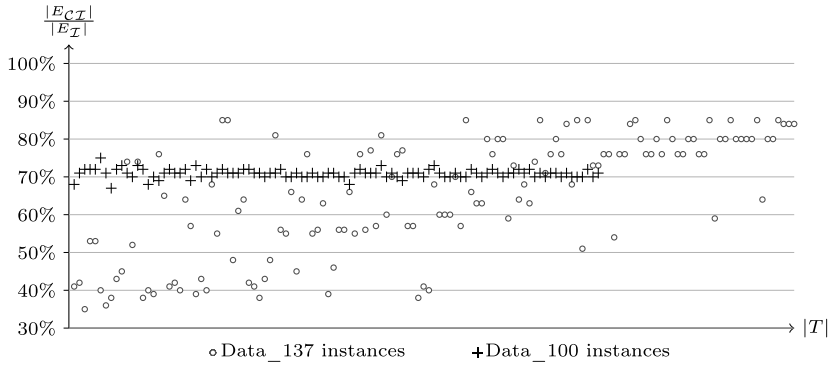


Fig. 8. Ratio $\frac{|E_{\mathcal{CI}}|}{|E_{\mathcal{I}}|}$ of all instances, sorted by increasing number of tasks.

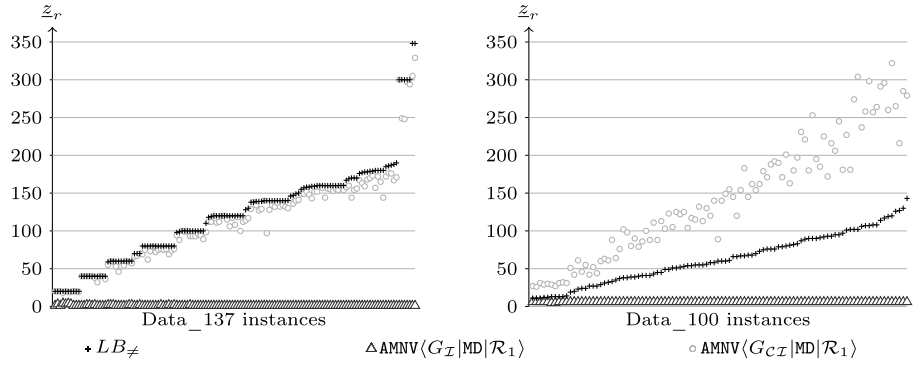


Fig. 9. Impact of $G_{\mathcal{CI}}$ on $\underline{z}_r$. Instances are sorted by increasing value of $LB_{\neq}$.

Shifts that belong to the same time interval are relatively close whereas shifts that belong to different time intervals are completely different.

5.2. Impact of $G_{\mathcal{CI}}$ on root node

Basically, using $G_{\mathcal{CI}}$ instead of $G_{\mathcal{I}}$ aims at reducing the number of edges to obtain larger independent sets, hence a better filtering. As a first step to measure the interest of $G_{\mathcal{CI}}$, we evaluate the ratio $\frac{|E_{\mathcal{CI}}|}{|E_{\mathcal{I}}|}$. On Data_137, this ratio ranges between 35% and 85% with a mean value of 65%. On Data_ 100, the trend is different since the minimum and the maximum ratio are respectively 67% and 75%, with a mean value of 71%. Note also that this ratio tends to increase on Data_137, whereas it is constant on Data_100. These differences surely come from the strengthened structure of Data_100 compared to Data_137. Overall, $G_{\mathcal{CI}}$ has significantly less edges than $G_{\mathcal{I}}$. (See Fig. 8.)

We then compare the value of the root lower bound, $\underline{z}_r$, obtained with AMNV$\langle G_{\mathcal{CI}}|MD|\mathcal{R}_1\rangle$ and AMNV$\langle G_{\mathcal{I}}|$MD$|\mathcal{R}_1\rangle$, on both data sets. $LB_{\neq}$ is used as a baseline comparison. Results are reported on Fig. 9. The horizontal axis represents instances, sorted by increasing value of $LB_{\neq}$. On both data sets, using AMNV$\langle G_{\mathcal{CI}}|MD|\mathcal{R}_1\rangle$ instead of AMNV$\langle G_{\mathcal{I}}|$MD$|\mathcal{R}_1\rangle$ dramatically improves the value of $\underline{z}_r$. On Data_137, the use of $G_{\mathcal{CI}}$ allows to get a relative gap between $LB_{\neq}$ and $\underline{z}_r$ of about 6%. As $LB_{\neq}$ is always equal to z^* on Data_137, this means that $\underline{z}_r$ is already very close to z^*. On Data_100, the use of $G_{\mathcal{CI}}$ allows to get values of $\underline{z}_r$ that are more than twice $LB_{\neq}$. On both data sets, our approach scales much better than the classical one which is not able to increase $\underline{z}_r$ whatever the size of the instance. More precisely the lower bound given by AMNV$\langle G_{\mathcal{I}}|$MD$|\mathcal{R}_1\rangle$ turns around 1 (respectively 6), which corresponds to the different time windows on Data_137 (respectively Data_100). Actually, this is not really surprising since AMNV$\langle G_{\mathcal{I}}|$MD$|\mathcal{R}_1\rangle$ is blind to `AllDifferent` constraints, which represent a big part of the problem. Using $G_{\mathcal{CI}}$ instead of $G_{\mathcal{I}}$ makes these `AllDifferent` constraints visible to `AtMostNValue`, hence the increased of $\underline{z}_r$.

5.3. Managing the tradeoff between filtering and runtime

As explained in Section 3.4, using the algorithm R^k is a simple and effective way to obtain diversification in filtering. The parameter k enables to manage the tradeoff between expected filtering power and runtime. Since R^k is suggested as a complement to MD, it must be seen as an improvement opportunity. Back to our case, many values of k, together with various filtering rules have been tested, within a time limit of 5 minutes. Within these experiments, the search heuristic does not branch on b_{reif}, which will be evaluated in the next section.

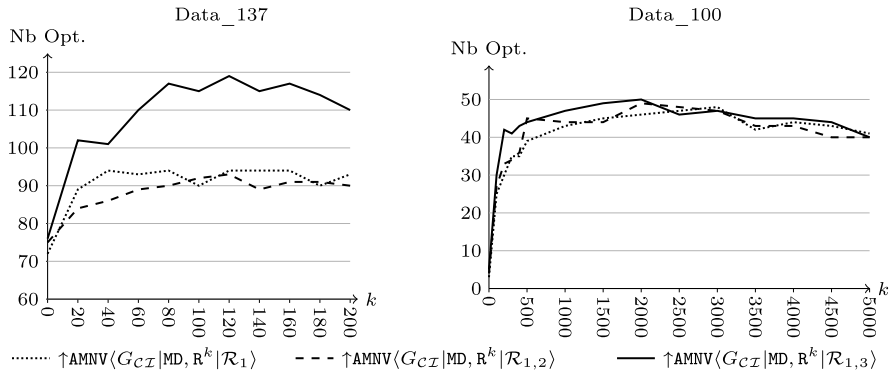


Fig. 10. Number of optima proved depending on k for several filtering configurations, with a time limit of 5 minutes.

Impact of `AMNV` *back-propagation*

In order to evaluate the influence of the back-propagation of `AMNV` on performances, we compare the number of optima proved by *CP model* with filtering rules $\mathcal{R}_1$, $\mathcal{R}_{1,2}$ and $\mathcal{R}_{1,3}$ (Fig. 10). $\mathcal{R}_{1,2}$ introduces a slight overhead compared to $\mathcal{R}_1$ only, without providing much more domain reductions. Thus, $\mathcal{R}_1$ is generally more efficient than $\mathcal{R}_{1,2}$ on Data_137. Instead, $\mathcal{R}_3$ has a strong impact on Data_137: It enables to solve 26 more instances within the time limit. Note that a minimum number of iterations is required before $\mathcal{R}_3$ brings a significant improvement. However, results are more qualified on Data_100. Overall, $\mathcal{R}_3$ is worth the overhead. Thus, the interest of $\mathtt{R}^k$ not only lies in finding larger independent sets ($\mathcal{R}_1$), but also in diversifying the filtering ($\mathcal{R}_3$).

Interestingly, the best setting for k changes quite a lot from one data set to another, hence using an automatic algorithm configuration program [33] seems relevant to get the best results. About Data_137, a few dozens of iterations enable to enhance performances, with a best value for k around 120. On the contrary, hundreds and even thousands of iterations are required to solve Data_100 instances. In particular, performing 5000 iterations on Data_100 is still much better than performing none. This means that the benefit of R^k is absolutely worth the induced overhead. The optimal setting seems to be around 2000. Note that within 5 minutes, the best configuration of *CP model* is able to solve around 80% of Data_137, compared to only half of Data_100.

Impact of `AllDifferent` *propagation*

In this section, we investigate the interest of $\mathcal{R}_4$. As explained in Section 3.5, this rule is not relevant to the SMPTSP, because all maximal `AllDifferent` constraints are already known. Nevertheless, it may be worthwhile when disequalities appear dynamically. To get insight into the behaviour of $\mathcal{R}_4$, while preserving the SMPTSP as an illustration, we remove all the `AllDifferent` constraints of *CP model* and replace them by cliques of binary disequalities (suffixed with a *). This enables to measure the filtering ability stemming from $\mathcal{R}_4$, which only makes sense if maximal `AllDifferent` constraints are not in the original model.

Thus, the semantic of the problem remains the same while the global view of the `AllDifferent` constraints is lost. Note that, we no longer use $LB_{\neq}$ as an initial lower bound, because it corresponds to the size of the largest `AllDifferent` constraint. Given this new model, we compare the number of optima respectively proved by $\uparrow\mathtt{AMNV}\langle G_{\mathcal{CI}}|\mathtt{MD}, \mathtt{R}^k|\mathcal{R}_{1,3}\rangle^*$ and $\uparrow\mathtt{AMNV}\langle G_{\mathcal{CI}}|\mathtt{MD}, \mathtt{R}^k|\mathcal{R}_{1,3,4}\rangle^*$, for different values of k, after a resolution of 5 minutes (Fig. 11). As justified in Section 3.5, $\mathcal{R}_4$ calls the `AllDifferent` BC filtering algorithm on every independent set.

As a baseline comparison, we also display the results of $\uparrow\mathtt{AMNV}\langle G_{\mathcal{CI}}|\mathtt{MD}, \mathtt{R}^k|\mathcal{R}_{1,3}\rangle$, which includes `AllDifferent` constraints.[2] Therefore, Fig. 11 enables to evaluate the role of $\mathcal{R}_4$ when `AllDifferent` constraints are not given, but also to compare the good propagation of `AllDifferent` constraints with the naive propagation of binary disequalities.

Two main observations can be made on Data_137 results. First, the good propagation of `AllDifferent` constraints only makes a difference once the `AMNV` filtering is strong enough, *i.e.*, when $k > 0$. For instance, with $k = 120$, it enables to find 118 optima, whereas $\uparrow\mathtt{AMNV}\langle G_{\mathcal{CI}}|\mathtt{MD}, \mathtt{R}^k|\mathcal{R}_{1,3}\rangle^*$ and $\uparrow\mathtt{AMNV}\langle G_{\mathcal{CI}}|\mathtt{MD}, \mathtt{R}^k|\mathcal{R}_{1,3,4}\rangle^*$ respectively solve 101 and 99 instances to optimality. Second, $\mathcal{R}_4$ has a low impact on results. A slight improvement can be noticed for small k values, but this trend reverses when k increases. This means $\mathcal{R}_4$ is unfortunately not able to get back the good tradeoff between filtering and runtime of the original model, which has `AllDifferent` constraints in it.

Surprisingly, with $k < 1000$, $\mathcal{R}_4$ leads to even better results than the original model on Data_100. It provides a better tradeoff between filtering and runtime. However, increasing k even more introduces a strong overhead. Interestingly, it can be seen that $\uparrow\mathtt{AMNV}\langle G_{\mathcal{CI}}|\mathtt{MD}, \mathtt{R}^k|\mathcal{R}_{1,3}\rangle$ and $\uparrow\mathtt{AMNV}\langle G_{\mathcal{CI}}|\mathtt{MD}, \mathtt{R}^k|\mathcal{R}_{1,3}\rangle^*$ lead to very similar results. This means that achieving a high level of consistency over `AllDifferent` constraints or simply propagating binary disequalities does not

[2] The default implementation of `AllDifferent` in Choco 3-1-1 performs an ad hoc compromise between BC and GAC, in order to provide a good tradeoff between filtering and runtime.

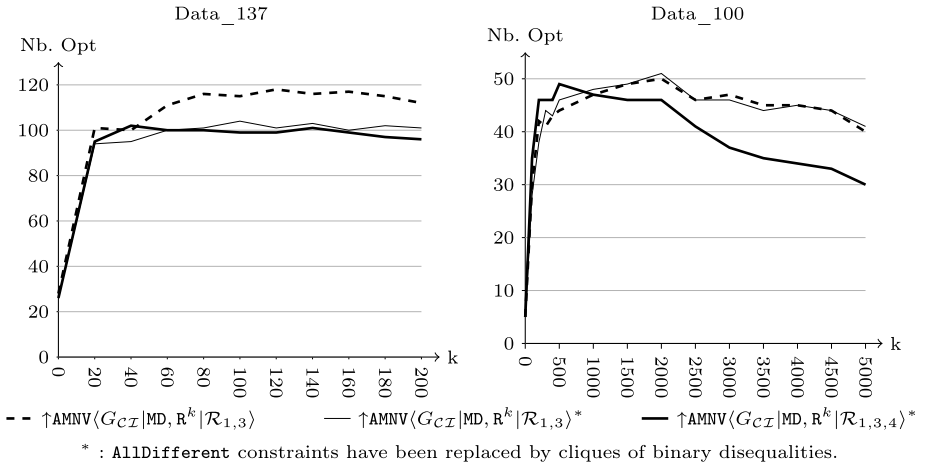


Fig. 11. Impact of $\mathcal{R}_4$ on the number of optima proved, depending on k. Note that $LB_{\neq}$ is no longer used as an initial lower bound for z.

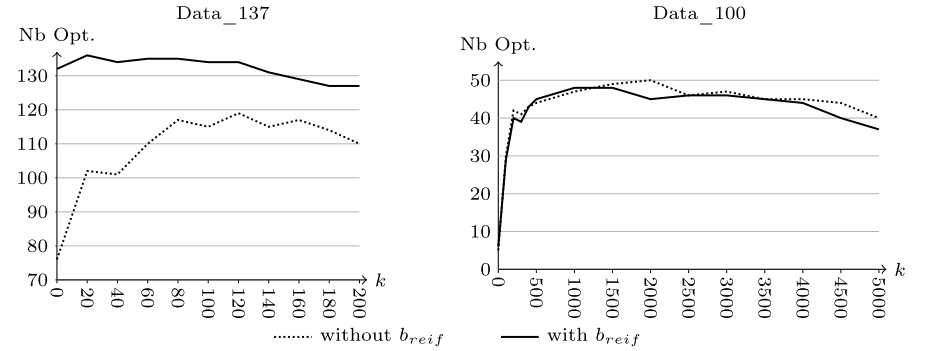


Fig. 12. Number of optima proved by ↑AMNV⟨$G_{\mathcal{CI}}$|MD, R^k|$\mathcal{R}_{1,3}$⟩ within a 5 minutes time limit, for various k settings, with and without b_{reif}.

really matter on this data set. The difficulty of such instances mainly stems in the lower bounding of the problem. Perhaps the AMNV propagation must be improved before AllDifferent constraint propagations play a key role.

5.4. Impact of branching on b_{reif}

We now evaluate the interest of branching on b_{reif} (Fig. 12), as suggested in Section 4. This binary variable reifies the constraint maximum$(\mathcal{X}) = z$, which breaks symmetries arising from the case where all shifts are equivalent. We mention that no instance satisfies such an assumption. Hence, branching on b_{reif} is a totally heuristic choice, aiming at reducing the search space. Results show that b_{reif} is very well suited to Data_137. It enables to solve 136 instances in less than 5 minutes (with $k = 20$). This may come from the $\frac{z^*}{|S|}$ ratio and the homogeneous repartition of task-shift compatibilities: Any set of z^* shifts is very likely to cover all tasks. Nevertheless, it is worth mentioning that, on this data set, no two shifts are identical. Unfortunately, b_{reif} does not help solving Data_100 instances, which have more structure in the design of shifts, to the contrary. We observe a slight performance loss. Note that the possibility to use constraint reification within search is a powerful feature of CP.

5.5. Scalability study

We now evaluate the ability of our approach to provide good bounds even on large scale instances. For that, we evaluate the relative gap $(\bar{z}-\underline{z})/\bar{z}$ depending on the running model. Both *MIP model* and *CP model* are run with a time limit of 6 minutes. Default values for $\underline{z}$ and $\bar{z}$ are respectively 0 and $|S|$, *i.e.*, $LB_{\neq}$ is not used. Therefore, a run which times out without computing any bound leads to a relative gap of 100%. In order to get tight bounds, *CP model* is first run 3 minutes in a bottom–up approach to compute a good lower bound and then, if no optimum has been found, 3 other minutes in a top–down mode to get a good upper bound. Given the size of the instances, finding a feasible solution may be difficult. As the AtMostNValue propagation relies on lower bounding, we can expect a strong propagation during the bottom–up step. This is no longer expectable in a top–down approach, for which the faster propagation the better. Therefore, the filtering of ↓AMNV⟨$G_{\mathcal{CI}}$|MD, R^k|$\mathcal{R}_{1,3}$⟩ is lightened by setting k to 0. Furthermore, as enforcing maximum$(\mathcal{X}) = z$ removes feasible solutions, b_{reif} is set to 0 as well.

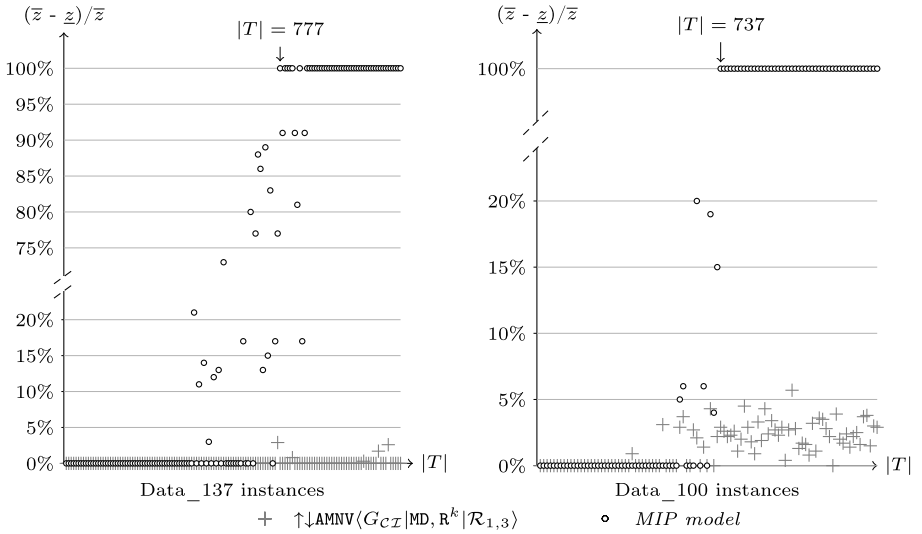


Fig. 13. Relative gap between $\underline{z}$ and $\bar{z}$ depending on the running model, on Data_137 and Data_100, after a 6 minutes resolution. Instances are sorted by increasing number of tasks.

Fig. 13 gives general trends regarding the ability to scale of *MIP model* and *CP model*. It shows that *MIP model* fails to solve about half of instances on both data sets: It is not able to provide either lower bounds or upper bounds (100% gap). On Data_137, there are 12 instances for which it is able to compute a lower bound, but no upper bound (70%–95% gap). This indicates that embedding the linear relaxation of the problem within a global constraint is not promising at all on these large scale instances. Regarding this scalability issue, the top–down and bottom–up CP approaches perform very well. The relative gap is on average 0.1% and 1.4%, with a maximum of 2.9% and 5.7%, for respectively Data_137 and Data_100. Thus, the relative gap does not increase much with the instance size. Consequently, using $\uparrow\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}, \texttt{R}^k|\mathcal{R}_{1,3}\rangle$ and $\downarrow\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}|\mathcal{R}_{1,3}\rangle$ leads to finding tight bounds for z, even on large instances. This makes *CP model* a more reliable approach than *MIP model* to bound the objective in a short time.

5.6. A competitive approach

In order to estimate the quality of our approach, we now compare it to the state-of-the-art SMPTSP approaches, on instances of the literature (Data_137). The first two SMPTSP approaches were given by Krishnamoorthy et al., who introduced both *MIP model* and a dedicated metaheuristic [36]. Next, Smet et al. provided a metaheuristic based on a local branching improvement heuristic [56]. Finally, and very recently, Lin and Ying proposed a three-phase exact approach [43]: The first phase builds an initial solution with a constructive heuristic, the second one improves it with an iterated greedy algorithm, and the third one solves *MIP model*, initialized with the best solution found.

We compare the performances of *CP model* with existing approaches. This comparison is based on their ability to reach an optimal solution in a short time. For *CP model*, we used $\uparrow\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}, \texttt{R}^{40}|\mathcal{R}_{1,3}\rangle$, the best configuration on Data_137. Regarding the results of [43], it occurs that their last phase times out on instances for which the input solution (computed in phases one and two) is optimal. Presumably, they were not aware that $LB_{\neq}$ is optimal on Data_137, which was first pointed out by Smet et al. Therefore, to have a fair evaluation of their method, we consider that it stops whenever one of the three phases has found an optimal solution. This slightly shorten the running time of [43].

As illustrated in Fig. 14, both *CP model* and [56] dominate other approaches. They are comparable in the sense that they both find all optimal solutions within the given time limit. Nevertheless, [56] focuses on finding good solutions and its only lower bounding procedure is $LB_{\neq}$. Therefore, this approach is not able to prove optimality in general, whereas *CP model* is able to provide optimality certificates. We point out that the results of *CP model* are better than those presented in [21]. This comes from slight code refactoring along with the use of b_{reif} (see Section 4).

As the SMPTSP often occurs as a subproblem of a more general method, it seems critical to provide a good lower bound in a short time. To evaluate the capacity of $\uparrow\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}, \texttt{R}^k|\mathcal{R}_{1,3}\rangle$ to provide quickly a good value of $\underline{z}$, we compare $\underline{z}_r$ and its required runtime with those of the lower bounding procedure given in [36]. This procedure is based on a Lagrangian relaxation, therefore, we refer to its lower bound as $\underline{z}_{\mathcal{L}}$. For a full resolution, $\uparrow\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}, \texttt{R}^{40}|\mathcal{R}_{1,3}\rangle$ gives the best results on Data_137. However, if the purpose is to design an effective lower bounding procedure, it may be interesting to use a much higher value of k, which may reduce the gap to optimality with a small overhead. Consequently, we compare $\underline{z}_{\mathcal{L}}$ with $\underline{z}_r$ for two different settings of k: 40 and 1000. To ease the reading, $\underline{z}_r$ will be noted $\underline{z}_r^{40}$ when obtained with $k = 40$ and $\underline{z}_r^{1000}$ when obtained with $k = 1000$. To have a fair comparison, we do not use $LB_{\neq}$ in our model, since it gives the optimal value on the state-of-the-art instances.

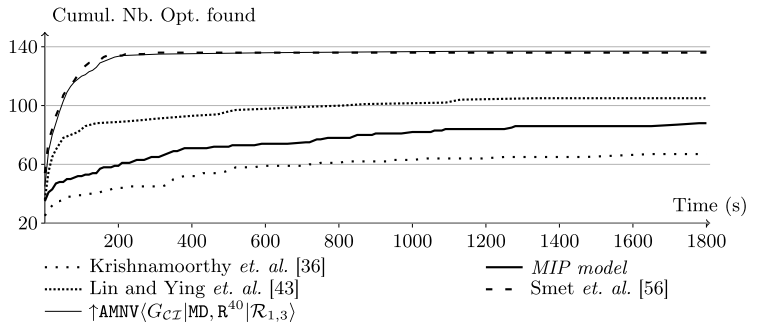


Fig. 14. Cumulated number of optima found, depending on runtime, on Data_137.

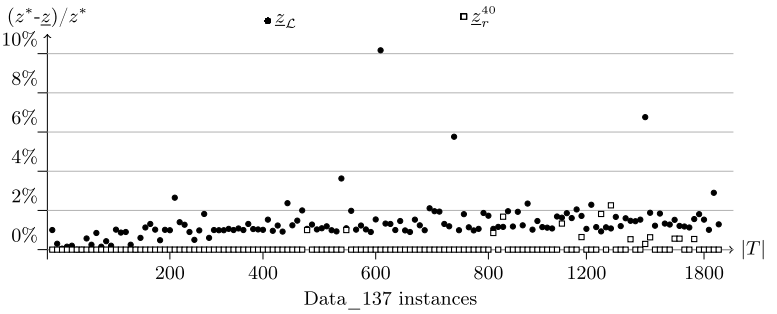


Fig. 15. Relative gap between $\underline{z}$ and z^* depending on the running model, on Data_137 instances, sorted by increasing task number. $\underline{z}_r^{1000}$ is not displayed because it is equal to z^* on every instance.

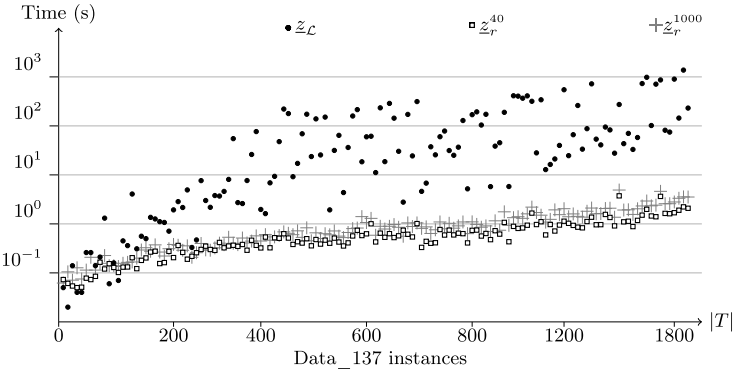


Fig. 16. Runtime (in seconds) to compute $\underline{z}$, depending on the running model.

As illustrated in Fig. 15, $\underline{z}_r^{40}$ and $\underline{z}_r^{1000}$ are much closer to z^* than $\underline{z}_{\mathcal{L}}$. More precisely, $\underline{z}_r^{1000}$ is always equal to z^* and $\underline{z}_r^{40}$ is on average more than ten times closer to z^* than $\underline{z}_{\mathcal{L}}$. Moreover, the empirical worst-case gap of $\underline{z}_r^{40}$ (2.26%) is also better than the one of $\underline{z}_{\mathcal{L}}$ (10.17%). Finally, as illustrated by Fig. 16, the runtime of $\underline{z}_r^{40}$ and $\underline{z}_r^{1000}$ are much smaller than the one of $\underline{z}_{\mathcal{L}}$: The average runtime of $\underline{z}_r^{40}$ turns around half a second compared to more than one hundred seconds for $\underline{z}_{\mathcal{L}}$. Note that the average runtime of $\underline{z}_r^{1000}$ is less than twice as big as the one of $\underline{z}_r^{40}$, which highlights the interest of increasing k when searching a good lower bound. On the whole, the root node provided by $\uparrow\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}, \texttt{R}^k|\mathcal{R}_{1,3}\rangle$ is very close to the optimum, and it can be computed quickly, even on large scale instances.

6. Conclusions

In this paper, we have introduced a CP approach to solve the SMPTSP, along with a new set of challenging instances. This model outperforms previous exact approaches on state-of-the-art benchmarks. Furthermore, it provides very good lower and upper bounds, even on large instances, in a short time. Thus, the CP approach competes with both exact and metaheuristic state-of-the-art approaches.

Furthermore, we have provided the notation $\texttt{AMNV}\langle G|F|\mathcal{R}\rangle$, to describe a family of `AtMostNValue` propagators. In particular, we introduced a new propagator, $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD}, \texttt{R}^k|\mathcal{R}_{1,3}\rangle$, to filter the conjunction of an `AtMostNValue` constraint

and disequalities. This propagator keeps close to the existing AtMostNValue propagator because it is fast to propagate, but also to capitalise over previous work in order to reduce implementation and maintenance issues. $\texttt{AMNV}\langle G_{\mathcal{CI}}|\texttt{MD},\texttt{R}^k|\mathcal{R}_{1,3}\rangle$ relies on a more appropriate graph structure ($G_{\mathcal{CI}}$), as well as refined filtering rules ($\mathcal{R}_3$), and a simple way to diversify filtering ($\texttt{R}^k$), in order to improve the overall approach. Moreover, this propagator gives control over the tradeoff between filtering and runtime. As it is simple to implement, effective and relevant for many applications, we believe that this propagator would benefit the CP community.

Nevertheless, the maximum independent set is not a tight relaxation for AtMostNValue, and there are several leads to achieve a stronger filtering. One of these is the Lovász number [45], a real number introduced in Graph Theory as an upper bound over the Shannon capacity of a graph. Lovász's sandwich theorem indicates that the Lovász number of $G_{\mathcal{CI}}$ provides a valid lower bound over the number of distinct values to be taken by $\mathcal{X}$, which is greater or equal to the maximum independent set relaxation. This number can be approximated with Semi Definite Programming (SDP) solvers (*e.g.* [8,11]), which are unfortunately not yet fast enough to allow an integration into a global constraint. Therefore, future improvement in SDP may have a positive impact on CP applications related to the SMPTSP. Another research perspective would be to investigate the application of this propagator in the context of dynamic difference constraints, which occurs in many disjunctive scheduling problems.

Finally, we introduced an original way to enhance search by introducing and reifying a global constraint. The success of this dedicated and naive attempt draw new perspectives regarding search heuristics. We believe this concept may be generalised toward new black-box search strategies [9,46,49], based on global constraint reification. This may reduce the need of designing dedicated search procedures [53], which still restricts the diffusion of CP. Past work on both symmetry breaking [26] and global constraint learning [6] may be a good lead.

Acknowledgements

The authors thank the anonymous referees as well as Samuel Burer, Xavier Lorca, Thierry Petit, Damien Prot and Charles Prud'Homme for their useful comments. This work has been funded by the regional council of Pays de la Loire and the CNRS.

References

[1] G. Andreello, A. Caprara, M. Fischetti, Embedding {0, 1/2}-cuts in a branch-and-cut framework: a computational study, INFORMS J. Comput. 19 (2) (2007) 229–238.
[2] S. Asaf, H. Eran, Y. Richter, D.P. Connors, D.L. Gresh, J. Ortega, M.J. Mcinnis, Applying constraint programming to identification and assignment of service professionals, in: CP, in: Lecture Notes in Computer Science, vol. 6308, Springer, 2010, pp. 24–37.
[3] N. Beldiceanu, Pruning for the minimum constraint family and for the number of distinct values constraint family, in: CP, in: Lecture Notes in Computer Science, vol. 2239, Springer, 2001, pp. 211–224.
[4] N. Beldiceanu, M. Carlsson, S. Demassey, T. Petit, Global constraint catalogue: past, present and future, Constraints 12 (1) (2007) 21–62.
[5] N. Beldiceanu, M. Carlsson, P. Flener, J. Pearson, On the reification of global constraints, Constraints 18 (1) (2013) 1–6.
[6] N. Beldiceanu, H. Simonis, A constraint seeker: finding and ranking global constraints from examples, in: CP, in: Lecture Notes in Computer Science, vol. 6876, Springer, 2011, pp. 12–26.
[7] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, T. Walsh, Filtering algorithms for the NValue constraint, Constraints 11 (4) (2006) 271–293.
[8] B. Borchers, Csdp, a c library for semidefinite programming, Optim. Methods Softw. 11 (1–4) (1999) 613–623.
[9] F. Boussemart, F. Hemery, C. Lecoutre, L. Sais, Boosting systematic search by weighting constraints, in: European Conference on Artificial Intelligence, 2004, pp. 146–150.
[10] Bron, Coen, Kerbosch, Joep, Algorithm 457: finding all cliques of an undirected graph, Commun. ACM 16 (9) (1973) 575–577.
[11] S. Burer, R.D.C. Monteiro, A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization, Math. Program. 95 (2) (2003) 329–357.
[12] M.W. Carter, C.A. Tovey, When is the classroom assignment problem hard? Oper. Res. 40 (1) (1992) 28–39.
[13] G. Chabert, L. Jaulin, X. Lorca, A constraint on the number of distinct vectors with application to localization, in: CP, in: Lecture Notes in Computer Science, vol. 5732, Springer, 2009, pp. 196–210.
[14] CHOCO Team, Choco: an open source Java constraint programming library, Tech. Rep., Ecole des Mines de Nantes, 2010, http://www.emn.fr/z-info/choco-solver/.
[15] R. Debruyne, C. Bessière, Some practicable filtering techniques for the constraint satisfaction problem, in: Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI'97, Morgan Kaufmann Publishers Inc., 1997, pp. 412–417.
[16] M.C. Dijkstra, L.G. Kroon, M. Salomon, J.A.E.E. Van Nunen, L.N. van Wassenhove, Planning the size and organization of KLM's aircraft maintenance personnel, Interfaces 24 (6) (1994) 47–58.
[17] D. Dowling, M. Krishnamoorthy, H. Mackenzie, D. Sier, Staff rostering at a large international airport, Ann. Oper. Res. 72 (1997) 125–147.
[18] P. Erdős, A. Rubin, H. Taylor, Choosability in graphs, in: Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing, in: Congres., vol. 26, 1979, pp. 125–157.
[19] A.T. Ernst, H. Jiang, M. Krishnamoorthy, D. Sier, Staff scheduling and rostering: a review of applications, methods and models, Eur. J. Oper. Res. 153 (1) (2004) 3–27.
[20] F. Fages, On the use of constraint reification within search, 2013, personal communication.
[21] J.G. Fages, T. Lapègue, Filtering atmostnvalue with difference constraints: application to the shift minimisation personnel task scheduling problem, in: CP, in: Lecture Notes in Computer Science, vol. 8124, Springer, 2013, pp. 63–79.
[22] T. Feydy, Z. Somogyi, P.J. Stuckey, Half reification and flattening, in: CP, in: Lecture Notes in Computer Science, vol. 6876, Springer, 2011, pp. 286–301.
[23] M. Fischetti, S. Martello, P. Toth, The fixed job schedule problem with spread-time constraints, Oper. Res. 35 (6) (1987) 849–858.
[24] M. Fischetti, S. Martello, P. Toth, The fixed job schedule problem with working-time constraints, Oper. Res. 37 (3) (1989) 395–403.
[25] B. Flavia, D. Guillermo, M. Javier, Exploring the complexity boundary between coloring and list-coloring, Electron. Notes Discrete Math. 25 (2006) 41–47.

[26] P. Flener, J. Pearson, M. Sellmann, P.V. Hentenryck, M. Ågren, Dynamic structural symmetry breaking for constraint satisfaction problems, Constraints 14 (4) (2009) 506–538.
[27] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.
[28] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, 2nd edn, Ann. Discrete Math., vol. 57, Elsevier, 2004.
[29] M. Gondran, M. Minoux, S. Vajda, Graphs and Algorithms, John Wiley & Sons, Inc., New York, NY, USA, 1984.
[30] S. Gualandi, F. Malucelli, Exact solution of graph coloring problems via constraint programming and column generation, INFORMS J. Comput. 24 (2012) 81–100.
[31] M.M. Halldórsson, J. Radhakrishnan, Greed is good: approximating independent sets in sparse and bounded-degree graphs, Algorithmica 18 (1) (1997) 145–163.
[32] R.M. Haralick, G.L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, in: Proceedings of the 6th International Joint Conference on Artificial Intelligence, IJCAI'79, vol. 1, Morgan Kaufmann Publishers Inc., 1979, pp. 356–364.
[33] S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney, Isac – instance-specific algorithm configuration, in: ECAI, in: Frontiers in Artificial Intelligence and Applications, vol. 215, IOS Press, 2010, pp. 751–756.
[34] A.W.J. Kolen, J.K. Lenstra, C.H. Papadimitriou, F.C.R. Spieksma, Interval scheduling: a survey, Nav. Res. Logist. 54 (2007) 530–543.
[35] M.Y. Kovalyov, C.T. Ng, T.C.E. Cheng, Fixed interval scheduling: models, applications, computational complexity and algorithms, Eur. J. Oper. Res. 178 (2007) 331–342.
[36] M. Krishnamoorthy, A. Ernst, D. Baatar, Algorithms for large scale shift minimisation personnel task scheduling problems, Eur. J. Oper. Res. 219 (2012) 34–48.
[37] M. Krishnamoorthy, A. Ernst, The personnel task scheduling problem, in: X. Yang, K. Teo, L. Caccetta (Eds.), Optimization Methods and Applications, in: Applied Optimization, vol. 52, Springer, US, 2001, pp. 343–368.
[38] L.G. Kroon, M. Salomon, L.N.V. Wassenhove, Exact and approximation algorithms for the tactical fixed interval scheduling problem, Oper. Res. 45 (4) (1997).
[39] C. Kuip, Public remark at workshop "CP solvers: modeling, applications, integration, and standartization", in: 19th International Conference on Principles and Practice of Constraint Programming, 2013.
[40] T. Lapègue, J.G. Fages, D. Prot, O. Bellenguez-Morineau, Personnel task scheduling problem library, https://sites.google.com/site/ptsplib/smptsp/home, 2013.
[41] C. Lecoutre, L. Sais, S. Tabary, V. Vidal, Reasoning from last conflict(s) in constraint programming, Artif. Intell. 173 (18) (2009) 1592–1614.
[42] R. Leone, P. Festa, E. Marchitto, A bus driver scheduling problem: a new mathematical model and a GRASP approximate solution, J. Heuristics 17 (4) (2010) 441–466.
[43] S.W. Lin, K.C. Ying, Minimizing shifts for personnel task scheduling problems: a three-phase algorithm, Eur. J. Oper. Res. (2014).
[44] A. López-Ortiz, C.G. Quimper, J. Tromp, P. van Beek, A fast and simple algorithm for bounds consistency of the alldifferent constraint, in: IJCAI, Morgan Kaufmann, 2003, pp. 245–250.
[45] L. Lovász, On the Shannon capacity of a graph, IEEE Trans. Inf. Theory 25 (1) (1979) 1–7.
[46] L. Michel, P. Van Hentenryck, Activity-based search for black-box constraint programming solvers, in: CPAIOR, 2012, pp. 228–243.
[47] J.N. Monette, P. Flener, J. Pearson, Towards solver-independent propagators, in: CP, in: Lecture Notes in Computer Science, vol. 7514, Springer, 2012, pp. 544–560.
[48] F. Pachet, P. Roy, Automatic generation of music programs, in: CP, 1999, pp. 331–345.
[49] P. Refalo, Impact-based search strategies for constraint programming, in: Principles and Practice of Constraint Programming, 2004, pp. 557–571.
[50] J.C. Régin, A filtering algorithm for constraints of difference in CSPs, in: National Conference on Artificial Intelligence, AAAI, 1994, pp. 362–367.
[51] Y. Richter, A. Freund, Y. Naveh, Generalizing alldifferent: the somedifferent constraint, in: CP, in: Lecture Notes in Computer Science, vol. 4204, Springer, 2006, pp. 468–483.
[52] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006.
[53] T. Schrijvers, G. Tack, P. Wuille, H. Samulowitz, P.J. Stuckey, Search combinators, in: Principles and Practice of Constraint Programming, 2011, pp. 774–788.
[54] C. Schulte, P.J. Stuckey, Efficient constraint propagation engines, ACM Trans. Program. Lang. Syst. 31 (1) (2008).
[55] C. Schulte, G. Tack, Weakly monotonic propagators, in: Principles and Practice of Constraint Programming, 2009, pp. 723–730.
[56] P. Smet, T. Wauters, M. Mihaylow, G. Vanden Berghe, The shift minimisation personnel task scheduling problem: a new hybrid approach and computational insights, Technical Report, University of North Carolina, 2013.
[57] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, L. De Boeck, Personnel scheduling: a literature review, Eur. J. Oper. Res. 226 (3) (2013) 367–385.