Soft-regular with a Prefix-size Violation Measure

Minh Thanh Khong[1], Christophe Lecoutre[2], Pierre Schaus[1], and Yves Deville[1]

[1] ICTEAM, Université catholique de Louvain, Belgium
[2] CRIL-CNRS UMR 8188, Université d'Artois, Lens, France
{minh.khong, pierre.schaus, yves.deville}@uclouvain.be; lecoutre@cril.fr

Abstract. In this paper, we propose a variant of the global constraint `soft-regular` by introducing a new violation measure that relates a cost variable to the size of the longest prefix of the assigned variables, which is consistent with the constraint automaton. This measure allows us to guarantee that first decisions (assigned variables) respect the rules imposed by the automaton. We present a simple algorithm, based on a Multi-valued Decision Diagram (MDD), that enforces Generalized Arc Consistency (GAC). We provide an illustrative case study on nurse rostering, which shows the practical interest of our approach.

1 Introduction

Global constraints play an important role in Constraint Programming (CP) due to their expressiveness and capability of efficiently filtering the search space. Popular global constraints include, among others, `allDifferent` [1], `count` [2], `element` [3], `cardinality` [4, 5], `cumulative` [6] and `regular` [7]. The constraint `regular` imposes a sequence of variables to take their values in order to form a word recognized by a finite automaton. This constraint happens to be useful when modeling various combinatorial problems such as rostering and car sequencing problems.

In many time-oriented problems, such as planning, scheduling, and timetabling, one has to take decisions while paying attention to the future demands, resources, etc. Those are generally obtained from a forecasting model. The standard approach is basically to define an horizon and to try having the problem instances solved over that horizon. Unfortunately, many problem instances are over-constrained [8, 9]. Following the approach of [10], some hard-constraints can then be relaxed and replaced by their soft versions. Although attractive, this approach applied to time-oriented problems has a major drawback which is that constraints are equally penalized, whether the violation is about the beginning or the end of the horizon. We claim that the importance of completely satisfying the constraints must decrease with elapsed time. In other words, satisfying the constraints in the near future must be considered as more important than satisfying the constraints in the far future (in addition, forecasts may be more or less accurate).

In this paper, we are interested in the relaxation of the global constraint `regular`. Existing violation measures for `soft-regular` are based on the concept of distances in term of variables or edit operations [11–13]. We propose an alternative violation measure that is based on the size of the longest prefix that is consistent with the underlying automaton of the constraint. This violation measure can be useful when first variables of the sequence (scope) are critical. We illustrate our approach on a rostering application.

2 Technical Background

A Constraint Satisfaction Problem (CSP) [14–16] is composed of a set of n variables, $X = \{x_1, \dots, x_n\}$, and a set of e constraints, $C = \{c_1, \dots, c_e\}$. On the one hand, each variable x has an associated domain, denoted by $D(x)$, that contains the set of values that can be assigned to x. Assuming that the domain $D(x)$ of a variable x is totally ordered, $min(x)$ and $max(x)$ will respectively denote the smallest value and the greatest value in the domain of x. Note also that d will denote the maximum domain size for the variables in a given CSP. On the other hand, each constraint c involves an ordered set of variables, called the scope of c and denoted by $scp(c)$. Each constraint c is mathematically defined by a relation, denoted by $rel(c)$, which contains the allowed combinations of values for $scp(c)$. The arity of a constraint c is the size of $scp(c)$, and will usually be denoted by r.

Given a sequence $\langle x_1, \dots, x_i, \dots, x_r \rangle$ of r variables, an r-tuple τ on this sequence of variables is a sequence of values $\langle a_1, \dots, a_i, \dots, a_r \rangle$, where the individual value a_i is also denoted by $\tau[x_i]$ or, when there is no ambiguity, $\tau[i]$. An r-tuple τ is *valid* on an r-ary constraint c iff $\forall x \in scp(c), \tau[x] \in D(x)$. A tuple τ is *allowed* by a constraint c iff $\tau \in rel(c)$; we also say that c accepts τ. A *support* on c is a valid tuple on c that is also allowed by c. A *literal* is a pair (x, a) where x is a variable and a a value, not necessarily in $dom(x)$. A literal (x, a) is *Generalized Arc-Consistent* (GAC) iff it admits a *support* on c, i.e., a valid tuple τ on c such that τ is allowed by c and $\tau[x] = a$. A constraint c is GAC iff $\forall x \in scp(c)$, $\forall a \in D(x)$, (x, a) is GAC.

3 Constraint `soft-regular`^{prx}

Definition 1 (DFA). *A* deterministic finite automaton *(DFA) is defined by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is a finite set of symbols called the alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states.*

Given an input string (a finite sequence of symbols taken from the alphabet Σ), the automaton starts in the initial state q_0, and for each symbol in sequence of the string, applies the transition function to update the current state. If the last state reached is a final state then the input string is accepted by the automaton. The set of strings that the automaton accepts constitutes a language, denoted by $L(M)$, which is technically a regular language.

In [7], a global constraint, called `regular`, is introduced: the sequence of values taken by the successive variables in the scope of this constraint must belong to a given regular language. For such constraints, a DFA can be used to determine whether or not a given tuple is accepted. This can be an attractive approach when constraint relations can be naturally represented by regular expressions in a known regular language. For example, in rostering problems, regular expressions can represent valid patterns of activities.

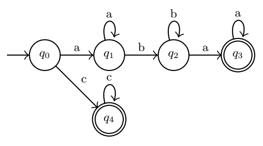

Fig. 1. A DFA with initial state q_0 and final states q_3 and q_4.

Definition 2 (`regular`). *Let M be a DFA, and $X = \langle x_1, \ldots, x_r \rangle$ be a sequence of r variables. The constraint* `regular`(X, M) *accepts all tuples in* $\{(v_1, \ldots, v_r) \mid v_1 \ldots v_r \in L(M) \wedge v_i \in D(x_i), \forall i \in 1..r\}$.

Example 1. Let us consider a sequence of 4 variables $X = \langle x_1, x_2, x_3, x_4 \rangle$ with $D(x_i) = \{a, b, c\}, \forall i \in 1..4$ and the automaton M depicted in Figure 1. The tuples (c, c, c, c) and (a, a, b, a) are accepted by the constraint `regular`(X, M) whereas (c, a, a, b) and (c, c, c, a) are not.

Working with constraints defined by a DFA, Pesant's filtering algorithm [7] enforces generalized arc consistency by means of a two-stage forward-backward exploration. This two-stage process constructs a layered directed multi-graph and collects the set of states that support each literal (x, a). The worst-case time and space complexities of this incremental algorithm are both $O(rd|Q|)$. In [17], the authors proposed a filtering algorithm for extended finite automata (with counters) by reformulating them as a conjunction of signature and transition constraints; their method can also be applied to constraints `regular` when there are no counters.

It is worth noting that there is a direct correspondence between MDDs (Multi-valued Decision Diagrams) and DFAs. An acyclic and minimized deterministic finite automaton is equivalent to a reduced (ordered) MDD [18]. This is the reason why we can use an underlying MDD [19, 20] as a basis for filtering constraints `regular`.

When problems are over-constrained, it is sometimes relevant to relax some hard constraints. More specifically, it is possible to soften global constraints by adopting some known violation measures [10, 11]. A violation measure μ is simply a cost function that guarantees that cost 0 is associated with, and only with, any tuple that fully satisfies the constraint. The violation measure "var" is general-purpose: it measures the smallest number of variables whose values must be changed in order to satisfy the constraint (it basically expresses a Hamming distance).

For the relaxed version of `regular`, we can use "var" as well as the violation measure "edit", defined in [11] and revised in [12], which stands for the smallest number of insertions, deletions and substitutions required to satisfy the constraint. In a local search context, it was proposed [13] a violation measure by dividing the sequence of variables into segments that are accepted by the underlying DFA. This measure overestimates the Hamming distance. In this paper, we propose an original violation measure "prx" that is related to the size (length) of the longest prefix of a tuple compatible with a DFA.

Definition 3 (Violation Measure "prx"). *The violation measure μ^{prx} of an r-tuple τ with respect to a DFA M is $\mu^{prx}(M, \tau) = r - k$, where k is the size of the longest prefix of τ that can be extended to an r-tuple in $L(M)$.*

Definition 4 (`soft-regular`^{prx}). *Let M be a DFA, $X = \langle x_1, \ldots, x_r\rangle$ be a sequence of r variables and z be a (cost) variable. The constraint* `soft-regular`$^{prx}(X, M, z)$ *accepts all tuples in* $\{(v_1, \ldots, v_r, d) \mid \mu^{prx}(M, (v_1, \ldots, v_r)) \leq d \wedge v_i \in D(x_i), \forall i \in 1..r \wedge d \in D(z)\}$.

Note that only the bounds of z are considered. We do not reason about equality, because z is supposed to be minimized. The constraint `soft-regular`$^{prx}(X, M, z)$ supports an r-tuple $\tau = (v_1, \ldots, v_r)$ for X iff τ is valid and $\mu^{prx}(M, \tau) \leq max(z)$, i.e., the first $r - max(z)$ values of τ can be extended to a tuple recognized by M. We also have that a value $d \in D(z)$ is a support of the constraint iff there exists a valid r-tuple τ for X such that $d \geq \mu^{prx}(M, \tau)$.

Example 2. Let us consider the automaton M depicted in Figure 1, a sequence of 4 variables $X = \langle x_1, x_2, x_3, x_4\rangle$ with $D(x_i) = \{a, b, c\}, \forall i \in 1..4$ and a cost variable z with $D(z) = \{0, 1, 2\}$. The constraint `soft-regular`$^{prx}(X, M, z)$ supports (c, c, c, a) for X but not (c, a, a, b) because $\mu^{prx}(M, (c, c, c, a)) = 1 \leq max(z)$ while $\mu^{prx}(M, (c, a, a, b)) = 3 > max(z)$. Suppose now that x_3 and x_4 are assigned to c and a, respectively. The r-tuple with the longest prefix consistent with M is $\tau = (c, c, c, a)$ with violation cost $\mu^{prx}(M, \tau) = 1$. Hence, $z = 0$ can not satisfy the constraint and should be removed from $D(z)$.

4 A GAC Algorithm

We now introduce a filtering algorithm to enforce generalized arc consistency on a constraint `soft-regular`^{prx}, i.e., a soft regular constraint, using "prx" as violation measure. As presented in [7, 11], the main data structure, the DFA, can be traversed in order to identify the values that are supported by the constraint. An MDD-based algorithm [19] can be applied for this step. Indeed, it is rather immediate to unfold the automaton on r levels, where each level corresponds to a variable in the main sequence X of variables. Arcs labelled with values for the first variable leave the root node (at level 0), whereas arcs labelled with values for the last variable reach the terminal node (at level r). An illustration is given by Figure 2.

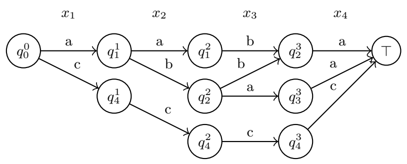

Fig. 2. MDD Built for the Constraint `soft-regular`^{prx} in Example 2.

The principle of filtering is the following. The MDD is traversed first in order to identify the deepest level for which a consistent prefix exists. With this information, it is then possible to update the min bound, $min(z)$, of the cost variable z. This is exactly the same kind of filtering performed with respect to the variable y in an inequality binary constraint $x \leq y$ when we apply: $min(y) \leftarrow max(min(y), min(x))$.

The value of the max bound, $max(z)$, is also useful for possibly pruning some values for variables in X. More precisely, for the first $r - max(z)$ variables in X, we need to only keep the values that occur in any consistent prefix whose size is at least equal to $r - max(z)$. This is the reason why we use an array *collected* that gives for each variable x_i, such that $i \leq r - max(z)$, the set of values $collected[x_i]$ that respect that condition. Roughly speaking, this is the spirit of the filtering performed with respect to the variable x in an inequality binary constraint $x \leq y$ when we apply: $max(x) \leftarrow min(max(x), max(y))$.

Algorithm 1 can be called to enforce GAC on a specified constraint `soft-regular`$^{prx}(X, M, z)$. We introduce a map, called *explored*, that stores for any processed node the size of the longest consistent prefix that can be reached from it. As usual for a map, we use the operations (i) *clear*(), for reinitializing the map, (ii) *contains*(), for determining if a specified node has already been processed, (iii) *get*(), for getting the size of the longest consistent prefix that can be reached from a specified node, (iiii) *put*(), for putting an entry in the map.

Algorithm 1: `soft-regular`$^{prx}(X = \langle x_1, \ldots, x_r \rangle, root, z)$

```
1 $explored.clear()$
2 $collected[x] \leftarrow \emptyset, \forall x \in X$
3 $maxSuccessLevel \leftarrow exploreTree(root)$
4 $min(z) \leftarrow max(min(z), r - maxSuccessLevel)$
5 if $D(z) = \emptyset$ then
6     return Failure
7 else
8     foreach $i \in 1..r - max(z)$ do
9         $D(x_i) \leftarrow collected[x_i]$
```

Algorithm 1 works as follows. Data structures are initialized at lines 1 and 2. Then, the exploration of the MDD starts from the root, and the size of the longest found consistent prefix is stored in the variable $maxSuccessLevel$. At this point, it is possible to update both the domain of the cost variable z and the domains of the first $r - max(z)$ variables of X, as explained earlier in this section. Note that if the domain of z becomes empty, a failure is returned. Also, note that no domain wipe-out (failure) can happen when updating the domains of variables in X.

Algorithm 2 makes an exploration of a (sub-)MDD and returns the size of the longest prefix compatible with the DFA that can be reached from the specified node. If this node corresponds to the terminal node (this is identified by a level equal to r) or a node that has already been processed, the algorithm returns the corresponding level. Otherwise, the algorithm explores each child (i.e., node reached from an outgoing edge) such that the value labeling the linking arc is still valid. Some values can then be collected, but note that we collect supported values for only the first $r - max(z)$ variables (lines 10-11). The level of a node is equal to the maximum level reached by its children, which can be expressed by the formula: $maxLevel(node) = max(node.level, max_{n:node.outs} maxLevel(n))$. Finally, Algorithm 2 adds an entry to the map *explored* and returns the size of the longest consistent prefix.

Algorithm 2: exploreTree($node$) : $Integer$

```
1  if $node.level = r$ then
2      return $r$
3  if $explored.contains(node)$ then
4      return $explored.get(node)$
5  $x \leftarrow node.var$
6  $maxLevel \leftarrow node.level$
7  foreach $arc \in node.outs$ do
8      if $arc.value \in D(x)$ then
9          $currMaxLevel \leftarrow exploreTree(arc.dest)$
10         if $node.level \leq r - max(z) \leq currMaxLevel$ then
11             $collected[x] \leftarrow collected[x] \cup \{arc.value\}$
12         $maxLevel \leftarrow max(maxLevel, currMaxLevel)$
13 $explored.put(node, maxLevel)$
14 return $maxLevel$
```

Example 3. Let us consider the constraint `soft-regular`$^{prx}(X, M, z)$ from Example 2. After the execution of Algorithm 1, we have: $collected[x_1] = \{a, c\}$, $collected[x_2] = \{a, b, c\}$, $collected[x_3] = collected[x_4] = \{\}$. Since $max(z) = 2$, only domains of x_1 and x_2 may be updated. Here, b is removed from $D(x_1)$.

Suppose now that x_3 and x_4 are respectively assigned to c and a, and 2 is removed from $D(z)$, i.e., $max(z) = 1$. Since $max(z)$ is now 1, we need to collect values for the first $4 - 1 = 3$ variables x_1, x_2, and x_3. Any value labeling an arc is collected iff this arc can reach at least the level 3. We have: $collected[x_1] = collected[x_2] = collected[x_3] = \{c\}$. Consequently, we now have $D(x_1) = D(x_2) = D(x_3) = \{c\}$. Note that the deepest reachable consistent level is 3, which implies that $min(z) = 1$.

Proposition 1. *Algorithm 1 enforces GAC on any specified constraint* `soft-regular`$^{prx}(X, M, z)$.

Proof. After the execution of Algorithm 2, $\forall x_i \in X$, with $i \in 1..r - max(z)$, $\forall v_i \in D(x_i)$, the arc associated with (x_i, v_i) must reach a node with a level at least equal to $r - max(z)$, i.e., there must exist a path from the root to this node whose size is at least $r - max(z)$. Values collected on this path can represent a support for (x_i, v_i). The maximum level reached from the root corresponds to the size of the longest prefix of a tuple that is consistent with the DFA M. Hence, the lower bound of $D(z)$ must be updated using this value. □

Since Algorithm 2 traverses at most one each arc in the corresponding graph, the time complexity of Algorithm 1 is $O(r|\delta|)$ where $|\delta|$ is the number of transitions in the DFA.

5 Possible Decomposition

The constraint $\texttt{soft-regular}^{prx}$ can be decomposed using cost MDD [21] constraints with the unfolded automaton described in Section 4 by adding the following arcs: for each node at level i, add an escape arc to the terminal node with cost $r - i$; each arc in the unfolded automaton has cost 0. The time complexity to filter this constraint is also linear in the number of arcs but it is not straightforward to implement if the cost MDD is not available.

The constraint $\texttt{soft-regular}^{prx}(X, M, z)$ can also be decomposed by using reified table constraints [22] and (ordinary) table constraints as follows. First, we introduce $r + 1$ new variables y_i $(i \in 0..r)$ such that $D(y_0) = \{q_0\}$, $D(y_i) = Q, \forall i \in 1..r - 1$ and $D(y_r) = F$. We also introduce r Boolean variables b_i, $i \in 1..r$, for reification purpose. Next, we introduce r reified table constraints $c_i^{reif} : c_i \Leftrightarrow b_i$ where c_i is a classical ternary positive table constraint such that $scp(c_i) = \{y_{i-1}, x_i, y_i\}$ and $rel(c_i) = \delta$ for $i = 1..r$. These constraints reflect the truth of a valid transition. We then introduce r functionality constraints c_i^f $(i = 1..r)$ where each constraint c_i^f is a ternary negative table constraint such that $scp(c_i^f) = \{y_{i-1}, x_i, y_i\}$ and $rel(c_i^f) = \{(q_k, v, \neq q_l) | \ (q_k, v, q_l) \in \delta\}$. Finally, we enforce the prefix restriction by adding the constraints: $(z \leq r - k) \Rightarrow (\sum_{i=1..k} b_i = k)$. Note that when $z \leq r - k$, a prefix of size at least equal to k must be consistent with the underlying DFA, which implies that $b_i = 1$, $i = 1..k$ and this is equivalent to $\sum_{i=1..k} b_i = k$.

6 Experimental Results

We illustrate the practical interest of our approach on a variant of the Nurse Rostering Problem (NRP). This problem has been extensively studied in both domains of Operational Research and Artificial Intelligence for more than 40 years due to its importance in real-world hospital contexts [23, 24].

The NRP consists of creating a roster by assigning nurses different shift types satisfying some constraints [23]. They are divided in two groups: hard constraints and soft constraints. Hard constraints must be satisfied in order to have a feasible solution whereas soft constraints can be partially violated by introducing some violation costs. The objective is to minimize the sum of these costs.

In real NRP situations, it may happen that one or even several nurses indicate that in a near future (not precisely indicated) they will have to be absent. One possible solution is to relax the hard `regular` constraints corresponding to the regulation rules for these nurses while trying to find a roster satisfying as much as possible the first steps (shifts) of the underlying automata. The constraint $\texttt{soft-regular}^{prx}$ can be applied for that. In [8], Schaus proposed to relax the demands instead using a `soft-cardinality` constraint. However, this relaxation does not allow us to optimize the longest feasible prefix easily over a horizon.

We have conducted an experimentation under Linux (CPUs clocked at 2.5 GHz, with 5GB of RAM). We have been interested in the NRP instances recently proposed in [25]. We have also randomly generated some variants for the last four instances by slightly modifying soft demands of nurses (we use symbols a and b as suffix in their names). For our experiments, a nurse was randomly chosen to be the person who can not totally follow the roster. So, we first minimize the global violation cost (as initially computed for these instances), and then we attempt to maximize the longest prefix satisfying the shift horizon of the "relaxed" nurses. Note that we also force this staff's roster to satisfy regulation rules over at least the first half of the scheduling horizon.

Table 1. Results obtained on NRP instances. Timeout set to 3, 600s.

Instance	#days	#nurses	#shifts	LNS objective	soft-regular LNS (1 nurse) objective	soft-regular LNS (1 nurse) violHoz
Instance1	14	8	1	607.0	512.0	5.9
Instance2	14	14	2	890.4	837.7	4.0
Instance3	14	20	3	1055.6	1006.2	3.6
Instance4	28	10	2	1732.4	1645.3	12.6
Instance4a	28	10	2	1694.2	1548.1	10.9
Instance4b	28	10	2	1722.4	1672.4	12.1
Instance5	28	16	2	1477.1	1261.9	11.5
Instance5a	28	16	2	1471.2	1390.0	11.7
Instance5b	28	16	2	1382.2	1303.0	11.1
Instance6	28	18	3	2629.1	2498.9	11.2
Instance6a	28	18	3	2539.6	2433.9	7.5
Instance6b	28	18	3	2706.7	2642.1	8.8
Instance7	28	20	3	1756.4	1555.1	9.1
Instance7a	28	20	3	1903.3	1810.6	8.3
Instance7b	28	20	3	1674.4	1485.8	9.4

We run our algorithm 10 times on each instance, with a timeout set to 3, 600 seconds, and we report average results. We chose Large Neighborhood Search (LNS) [26] as a method to solve NRP instances. The variable ordering heuristic selects the variable that admits the highest violation cost over a day while the value ordering heuristic selects the value that reduces the overall cost most. Concerning relaxation, firstly, one nurse is selected randomly. If no solution is found after 10 executions, the number of nurses relaxed will increase by one. It will be set back to one when a solution is found. For each restart, the number of failures is limited to 100K.

Table 1 shows some representative results for only one relaxed nurse. The first 4 columns gives some information about the instances. The 3 last columns present solving results. The first column (out of these 3 last columns) shows the violation cost obtained with LNS for the initial problem (i.e., without any relaxation). The last 2 columns present results for the relaxed problem: the obtained objective cost, and the horizon violated by the roster of the relaxed nurse. Generally speaking, one can observe an improvement on the overall objective while keeping a reasonable roster for the relaxed nurse.

Table 2. Execution time(s) for dedicated and decomposition approaches.

	instance1	instance2	instance3	instance4	instance5	instance6	instance7
dedicated	71.7	40.4	58.4	38.0	64.5	98.5	98.6
decomp	190.7	81.2	122.6	57.9	74.4	110.5	118.9

We also compared our soft constraint `soft-regular`^{prx} with the decomposition approach. For simplicity, a static branching was used, and the program was stopped when the number of failures reached 500K. The execution times in seconds are reported in Table 2. Clearly, the dedicated approach is more robust than the decomposition one, as it can be twice as fast.

Acknowledgments

The first author is supported by the FRIA-FNRS. The second author is supported by the project CPER Data from the "Hauts-de-France".

References

1. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI'94*, pages 362–367, 1994.
2. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994.
3. P. Van Hentenryck and J.-P. Carillon. Generality versus specificity: An experience with AI and OR techniques. In *Proceedings of AAAI'88*, pages 660–664, 1988.
4. J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of AAAI'96*, pages 209–215, 1996.
5. J.N. Hooker. *Integrated Methods for Optimization.* Springer, 2012.
6. A. Aggoun and N. Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.
7. G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proceedings of CP'04*, pages 482–495, 2004.
8. P. Schaus. Variable objective large neighborhood search: A practical approach to solve over-constrained problems. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 971–978. IEEE, 2013.
9. W. van Hoeve. Over-constrained problems. In *Hybrid Optimization*, pages 191–225. Springer, 2011.
10. T. Petit, J.-C. Régin, and C. Bessiere. Specific filtering algorithms for over-constrained problems. In *Proceedings of CP'01*, pages 451–463, 2001.
11. W. van Hoeve, G. Pesant, and L.-M. Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4-5):347–373, 2006.
12. J. He, P. Flener, and J. Pearson. Underestimating the cost of a soft constraint is dangerous: revisiting the edit-distance based soft regular constraint. *Journal of Heuristics*, 19(5):729–756, 2013.
13. J. He, P. Flener, and J. Pearson. An automaton constraint for local search. *Fundamenta Informaticae*, 107(2-3):223–248, 2011.
14. U. Montanari. Network of constraints : Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.
15. R. Dechter. *Constraint processing.* Morgan Kaufmann, 2003.
16. C. Lecoutre. *Constraint networks: techniques and algorithms.* ISTE/Wiley, 2009.
17. N. Beldiceanu, M. Carlsson, and T. Petit. Deriving filtering algorithms from constraint checkers. In *International Conference on Principles and Practice of Constraint Programming*, pages 107–122. Springer, 2004.
18. T. Hadzic, E.R. Hansen, and B. O'Sullivan. On automata, MDDs and BDDs in constraint satisfaction. In *Proceedings of ECAI'08 Workshop on Inference methods based on Graphical Structures of Knowledge*, 2008.
19. K. Cheng and R. Yap. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304, 2010.

20. G. Perez and J.-C. Régin. Improving GAC-4 for Table and MDD constraints. In *Proceedings of CP'14*, pages 606–621, 2014.
21. G. Perez and J.-C. Régin. Soft and cost mdd propagators. In *Proceedings of AAAI'17*, pages 3922–3928, 2017.
22. M.T. Khong, Y. Deville, P. Schaus, and C. Lecoutre. Efficient reification of table constraints. In *Tools with Artificial Intelligence (ICTAI), 2017 IEEE 29th International Conference on*. IEEE, 2017.
23. E. Burke, P. De Causmaecker, G.V. Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.
24. A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.
25. T. Curtois and R. Qu. Computational results on new staff scheduling benchmark instances. *Technical report, ASAP Research Group, School of Computer Science, University of Nottingham*, 06-Oct-2014.
26. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998.