Reformulating global constraints: the SLIDE and REGULAR constraints

Christian Bessiere[1], Emmanuel Hebrard[2], Brahim Hnich[3], Zeynep Kiziltan[4], Claude-Guy Quimper[5], and Toby Walsh[6]

[1] LIRMM, University of Montpellier, France. bessiere@lirmm.fr
[2] Emmanuel Hebrard, 4C, University College Cork, Ireland. ehebrard@4c.ucc.ie
[3] Brahim Hnich, Faculty of Computer Science, Izmir University of Economics, Turkey. brahim.hnich@ieu.edu.tr
[4] Zeynep Kiziltan, Department of Computer Science, University of Bologna, Italy. zeynep@cs.unibo.it
[5] Claude-Guy Quimper, Omega Optimisation, Canada. quimper@alumni.uwaterloo.ca
[6] Toby Walsh NICTA and University of New South Wales, Sydney, Australia. tw@cse.unsw.edu.au

Abstract. Global constraints are useful for modelling and reasoning about real-world combinatorial problems. Unfortunately, developing propagation algorithms to reason about global constraints efficiently and effectively is usually a difficult and complex process. In this paper, we show that reformulation may be helpful in building such propagators. We consider both hard and soft forms of two powerful global constraints, SLIDE and REGULAR. These global constraints are useful to represent a wide range of problems like rostering and scheduling where we have a sequence of decision variables and some constraint that holds along the sequence. We show that the different forms of SLIDE and REGULAR can all be reformulated as each other. We also show that reformulation is an effective method to incorporate such global constraints within an existing constraint toolkit. Finally, this study provides insight into the close relationship between these two important global constraints.

1 Introduction

Global constraints are one of the most important features of constraint programming. Global constraints capture common patterns occurring in models of complex, real-life combinatorial problems. For instance, a common pattern in rostering problems is that sequences of night shifts must be followed by several days off, and that no one is allowed to work more than a certain number of consecutive night shifts. Such patterns can be modelled using a REGULAR constraint [6]. More recently, we have proposed the global SLIDE to model a wide range of patterns appearing in sequencing and other problems [3]. In this paper, we explore the relationship between these two global constraints in depth. We show that the SLIDE constraint can be efficiently reformulated as the REGULAR constraint and vice versa.

In many real-world problems, it is not possible to find a feasible solution that satisfies all the constraints and preferences of the user. Consider the problem of allocating reviewers for papers submitted to a conference. Typically, a paper must be reviewed by a certain number of reviewers and each reviewer must have a certain number of papers. Reviewers also indicate preferences over the papers that they would be happy to review. Finding an allocation that satisfies the assignment constraints as well as all the reviewer preferences may not be possible. We may however give an additional paper to some reviewers and may assign a paper to a reviewer even if she is not enthusiastic about it, as long as she did not indicate a "conflict of interest". Soft versions of global constraints are useful to model and solve such over-constrained problems.

A recent direction is to convert over-constrained problems into constraint optimization problems by treating constraint violations as costs. To reason about such problems, we can design specific cost-based propagators. Several such soft global constraints have been proposed to model and solve over-constrained problems effectively and efficiently (e.g., [7, 9, 10, 12]). Whilst efficient propagators have been developed for both the SLIDE and REGULAR constraint and are available in a number of solvers, there has been less work about soft versions of these constraints. For example, no GAC propagators have yet been developed for soft versions of the SLIDE constraint. One of our contributions in this paper is to propose the first such propagators. We show that reformulation is an attractive mechanism also to implement soft versions of the SLIDE and REGULAR global constraints.

This rest of this paper is structured as follows. After giving the necessary formal background in Section 2, we explain in detail in Section 3 the SLIDE and REGULAR constraints. Then we show in Section 4 how SLIDE can be efficiently reformulated as the REGULAR constraint and vice versa. In Section 5, we focus on the soft versions of these global constraints and in Section 6 demonstrate that the different types of SOFTSLIDE constraints can be reformulated as hard forms of the SLIDE or soft form of the REGULAR constraints. We provide experimental proof in Section 7 that reformulating global constraints could be useful. We report related work in Section 8 and conclude in Section 9.

2 Formal Background

A constraint satisfaction problem consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for some subset of variables. We consider finite domain integer variables, and use capital letters for variables (e.g. X) and lower case for values (e.g. d). We write $D(X)$ for the domain of a variable X. A constraint C defined on variables $[X_i, \ldots, X_j]$ is indicated as $C(X_i, \ldots, X_j)$.

Constraint solvers typically explore partial assignments enforcing a local consistency property using either specialised or general purpose algorithms. A constraint C is *generalised arc consistent* (GAC) iff when a variable is assigned any of the values in its domain, there exist compatible values in the domains of all the other variables of C. For binary constraints, generalised arc consistency is often called simply arc consistency (AC). A constraint C is *bound consistent* (*BC*) iff when a variable is assigned the maximum (or minimum) value in its domain, there exist compatible values between the maximum and minimum domain value for all the other variables of C.

A deterministic finite automaton (DFA) is described by a 5-tuple $\mathcal{A} = \langle Q, Q_F, q_0, \delta, \Sigma \rangle$ where Σ is an alphabet, Q is a set of states, $q_0 \in Q$ is the initial state, $Q_F \subseteq Q$ is a set of final states, and $\delta \subseteq Q \times \Sigma \times Q$ is a transition table. A sequence $s_1, \ldots, s_n$ is accepted by the automaton $\mathcal{A}$ if there exists a sequence of states $t_0, \ldots, t_n$ such that $t_0 = q_0$ is the initial state, $t_n \in Q_F$ is a final state, and $\langle t_{i-1}, s_i, t_i \rangle \in \delta$ is a transition in the transition table. A language $\mathcal{L} \subseteq \Sigma^*$ is a (possibly infinite) set of sequences taken from an alphabet Σ. A *regular language* is the set of sequences accepted by a DFA.

The *Hamming distance* between two strings s_1 and s_2 of the same length is the number of positions in which they differ. The Hamming distance is denoted by $H(s_1, s_2)$. For example, the Hamming distance between the strings *abc* and *adc* is 1 as they differ only on the second position. The Hamming distance between a string s and a language $\mathcal{L}$ is $\min\{H(s,t) \mid t \in \mathcal{L}\}$. If $\mathcal{L}$ does not contain any string of the same length as s, the distance between s and $\mathcal{L}$ is undefined. The *edit distance* between two strings s_1 and s_2 is the minimum number of character insertions, deletions, and replacements to string s_1 in order to obtain s_2. The edit distance is denoted by $E(s_1, s_2)$. For example, the edit distance between the strings *abc* and *aadc* is 2 as we can replace the character b of the string *abc* by a and insert a d before c to obtain *aadc*. The edit distance between a string s and a language $\mathcal{L}$ is $\min\{E(s,t) \mid t \in \mathcal{L}\}$. If $\mathcal{L}$ is empty, the distance between s and $\mathcal{L}$ is undefined.

3 Slide and Regular constraints

The global Regular constraint was introduced by Pesant to model problems in scheduling and rostering [6]. The constraint is specified in terms of a finite automaton which accepts the string of values spelled out by the sequence of variables. More precisely, Regular($\mathcal{A}$, $[X_1, \ldots, X_n]$) holds iff X_1 to X_n form a string accepted by the DFA $\mathcal{A}$. Such a global constraint can be used to ensure certain patterns do (or do not) occur over time. For example, in shift rostering, we might have that we cannot work more than three night shifts in a row and once a sequence of night shifts ends, we must have at least two days off. This can easily be specified using a finite automaton. We have a finite automaton $\mathcal{A}$ with the states n_1, n_2, n_3, o_1 and *any*. The transition table is: $\langle any, day, any \rangle$, $\langle any, night, n_1 \rangle$, $\langle any, off, any \rangle$, $\langle n_i, night, n_{i+1} \rangle$, $\langle n_i, off, o_1 \rangle$, and $\langle o_1, off, any \rangle$. For instance, if we are in state *any*, we go to state n_1 with input *night*. Pesant gives a propagator for the Regular constraint based on dynamic programming which achieves GAC in $O(nd|Q|)$ time where $|Q|$ is the number of states of the automaton [6].

More recently, we introduced the global Slide constraint [3]. We begin with its simplest form. If C is a constraint of arity k then Slide(C, $[X_1, \ldots, X_n]$) holds iff $C(X_i, \ldots, X_{i+k-1})$ itself holds for $1 \leq i \leq n-k+1$. That is, we slide the constraint C down the sequence of variables, X_1 to X_n. For example, consider the car sequencing problem (prob001 in CSPLib) where we need to decide the order in which to build cars on an assembly line. We might want to ensure that no more than one out of every two cars has the sun roof option as it takes extra time to fit a sun roof. This can easily be specified with a SLIDE constraint. We slide a binary constraint down a sequence of decision variables representing the order in which cars will be produced. This binary constraint ensures that one of or both of the variables within its scope does not represent a car with the sun roof option. A variation of the dual encoding can be used to maintain GAC on such a SLIDE constraint in $O(nkd^k)$ time [3].

A more complex form of the SLIDE constraint permits us to slide down two or more sequences of variables at the same time. For instance, if C is a constraint of arity $2k$ then SLIDE$(C, [X_1, \ldots, X_n], [Y_1, \ldots, Y_n])$ holds iff $C(X_i, \ldots, X_{i+k-1}, Y_i, \ldots, Y_{i+k-1},)$ itself holds for $1 \leq i \leq n-k+1$. That is, we slide the constraint C down the two sequences of variables. Consider, for example, the global contiguity constraint [5]. This ensures that within a sequence of 0/1 variables, X_1 to X_n, the 1's occur in a continuous block. We can model this by a SLIDE constraint down two sequences of 0/1 variables, X_1 to X_n and Y_1 to Y_n. The second sequence of variables, Y_1 to Y_n record if we have met the block of 1's yet or not. The constraint being slid, $C(X_i, X_{i+1}, Y_i, Y_{i+1})$ holds iff ($Y_i = 0$, $X_i = 0$, $X_{i+1} = Y_{i+1}$) or ($Y_i = Y_{i+1} = 1$ and $X_i \geq X_{i+1}$). Such more complex forms of SLIDE can be reformulated as the simple form of SLIDE down a single sequence. We merely need to interleave the different sequences and then slide a suitably modified constraint over these interleaved sequences. (See [3] for details.)

4 Reformulating SLIDE and REGULAR

We first show that SLIDE and REGULAR can be reformulated as each other. As well as providing insight into the relationship between the two global constraints, these reformulations will be useful in providing propagators for soft versions of these global constraints.

4.1 REGULAR as SLIDE

In [8], we give a simple reformulation of the REGULAR constraint in terms of a SLIDE constraint. In addition to the sequence of variables along which the REGULAR constraint is defined, we introduce a sequence of variables for the state of the automaton. We then slide the transition relation down the two sequences of variables. For instance, consider again the shift rostering problem from the last section. We can reformulate REGULAR$(\mathcal{A}, [X_1, \ldots, X_n])$ as SLIDE$(C, [X_1, \ldots, X_{n+1}], [Q_1, \ldots, Q_n])$ where X_{n+1} is a dummy variable, Q_i are variables representing the state of the automaton, and $C(X_i, X_{i+1}, Q_i, Q_{i+1})$ holds iff we move from state Q_i to Q_{i+1} on seeing X_i. Observe that X_{i+1} is not used in the definition of C. We let it in its scope just to be consistent with the simplified presentation of Slide on multiple sequences that we presented in Section 3.

Enforcing GAC on this reformulation achieves GAC on the original Regular constraint. Hence, reformulation does not hinder propagation. This reformulation is also optimal in the sense that, we can enforce GAC on either the Regular constraint or its reformulation into Slide in $O(nd|Q|)$ time. This complexity is lower than the complexity of Slide in general because of the characteristics of the constraint being slid (see [3] for details). As we show in the experimental section, this reformulation is also a practical means to propagate the Regular constraint. By reformulating Regular as a Slide constraint, we get an efficient and incremental propagator that can outperform Pesant's propagator for Regular based on dynamic programming.

4.2 Slide as Regular

The reverse is also possible. That is, we can reformulate any instance of the Slide constraint using a Regular constraint. Again this is optimal in the sense that we can enforce GAC on either the Slide constraint or the reformulation into Regular in $O(nkd^k)$ time. We prove this claim by constructing a DFA $\mathcal{A}$ recognizing the language accepted by a Slide constraint.

Let $\Sigma = \bigcup_{i=1}^{n} D(X_i)$ be the alphabet and k be the arity of the constraint C. The states of $\mathcal{A}$ are given by the set of sequences $Q = \bigcup_{i=0}^{k-1} \Sigma^i$. The empty sequence $\epsilon \in Q$ is the initial state. Any sequence of length $k-1$ is a final state. Let $T = \{[s_1, \dots, s_k] \mid C([s_1, \dots, s_k])\}$ be the set of sequences accepted by the constraint C. We construct the transition table δ of $\mathcal{A}$ as follows. Let w be a sequence of length strictly smaller than $k-1$ and $c \in \Sigma$ be a character from the alphabet. Let wc be the concatenation of the sequence w and the character c. We have a transition $\langle w, c, wc \rangle \in \delta$ if there exists a sequence in T starting with wc. Let $a, b \in \Sigma$ be two characters and $w \in \Sigma^{k-2}$ be a sequence of length $k-2$ such that awb is a sequence in T. Then we have the transition $\langle aw, b, wb \rangle \in \delta$. Notice that a state w can only be visited after parsing the sub-sequence w.

Example 1. Consider the alphabet $\Sigma = \{a, b\}$ and the constraint C that accepts any sequence of length three but the sequences aaa and bbb. We obtain the DFA depicted in Figure 1.

The DFA $\mathcal{A}$ constructed in this way represents the Slide constraint.

Theorem 1. *If n is greater than or equal to the arity of C, then the language $\{X_1 \dots X_n \mid$ Slide$(C, [X_1, \dots, X_n])\}$ formed by the sequences satisfying the* Slide *constraint is equal to the set of sequences of length n recognized by the DFA $\mathcal{A}$.*

Proof. We first prove that every sequence s of length n accepted by $\mathcal{A}$ is also accepted by the Slide constraint. Let w be the first $k-1$ characters in s. By construction of $\mathcal{A}$ after reading these $k-1$ characters, the current state is w and there exists a sequence in T starting with w. Assume that the $k-1$ characters

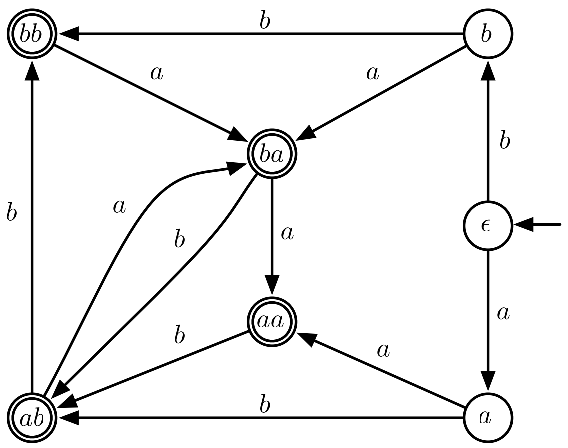

Fig. 1. DFA corresponding to Example 1.

following a position i in s are given by aw where a is a character and w is a sequence of length $k-2$. Also assume that there exists a sequence in T starting with aw and that after reading aw, $\mathcal{A}$ is in state aw. By construction of $\mathcal{A}$ when reading the character b at position i, the state of $\mathcal{A}$ changes from aw to wb. The transition guarantees that the sequence awb belongs to T and that the constraint C is satisfied at position i. By inductively repeating the argument, we conclude that the constraint C is satisfied at every position and that consequently, the Slide constraint is satisfied.

We now prove that if the sequence s satisfies the Slide constraint, then it is accepted by $\mathcal{A}$. Let awb be the first k characters of s where a and b are two characters and w is a sequence of $k-2$ characters. Since the Slide constraint is satisfied, the sequence awb satisfies the constraint C and belongs to the set T. Therefore, there is a series of states $q_0, \dots, q_{k-1}$ that parses the sequence aw where state q_i is the first i characters of aw. Suppose that the i^{th} character to be parsed is b, that the last $k-1$ parsed characters are the character a followed by the sequence w, and that $\mathcal{A}$ is in state aw. Since the Slide constraint is satisfied, the sequence awb satisfies C. Therefore, there exists a transition $\langle aw, b, wb\rangle \in \delta$ that parses the character b and leads to the state wb. Notice that wb are the last $k-1$ parsed characters. By inductively repeating the argument, we conclude that there exists a parsing for any sequence satisfying the slide constraint. Consequently, the sequence s is accepted by $\mathcal{A}$. □

5 Softening Slide and Regular

As we discussed in the introduction, real world problems are often over-constrained. One mechanism to deal with such over-constrained problems is to introduce a cost function, and find a solution of minimal cost from a feasible solution. We will formalize this process as follows. Let $C(X_1, \dots, X_n)$ be a hard global constraint like Slide or Regular. Given a distance function d between strings, the soft constraint $C_{soft}(X_1, \dots, X_n, Z)$ holds iff $Z = \min\{d(a_1, \dots, a_n, b_1, \dots, b_m) \mid a_i \in D(X_i)\ \forall i \in 1..n\ \&\ C(b_1, \dots, b_m)\}$. Distance might, for instance, be Hamming (in which case $n = m$) or edit distance. As stated in Section 2, d is not defined if the set of strings satisfying C is empty.

As an example, the constraint SoftSlide$_H(C, [X_1, \dots, X_n], D)$ holds iff D is the Hamming-distance between the sequence $[X_1, \dots, X_n]$ and the language accepted by the Slide constraint. Similarly, the constraint SoftSlide$_E(C, [X_1, \dots, X_n], D)$ holds iff D is the edit-distance between the sequence $[X_1, \dots, X_n]$ and the language accepted by the Slide constraint. Similarly, SoftRegular$_H$ and SoftRegular$_E$ are soft forms of Regular obtained by applying Hamming and edit distance based violations to Regular respectively [10]. For example, SoftRegular$_E(\mathcal{A}, [X_1, \dots, X_n], D)$ is satisfied iff D is the edit-distance between the sequence $[X_1, \dots, X_n]$ and the language accepted by $\mathcal{A}$. In [10], the propagation algorithm for Regular based on dynamic programming is modified in two different ways to maintain GAC on SoftRegular$_H$ and SoftRegular$_E$, respectively.

6 SoftSlide constraint

We will show that the different types of SoftSlide constraints can be reformulated as hard forms of the Slide or soft form of the Regular constraints. Reformulation thus provides a simple mechanism to propagate the soft forms of these global constraints.

6.1 SoftSlide$_H$ as Slide

Let us consider a SoftSlide$_H(C, [X_1, \dots, X_n], D)$ where the arity of constraint C is k and where $n \geq k$. In order to reformulate this SoftSlide constraint as a Slide constraint, we introduce two sequences of extra variables.

The first sequence contains $n+k-1$ variables $[S_1, \dots, S_{n+k-1}]$ that we build in such a way that $S_1, \dots, S_n$ can be any string accepted by Slide$(C, [S_1, \dots, S_n])$. The domain of each S_i contains all the values that appear in at least one tuple belonging to C. The variables S_{n+1} to S_{n+k-1} must not be forced to satisfy the constraint being slid. Hence, a dummy value '$*$' is added to the domain of every $S_i, i > n$. Furthermore, C' is defined as a relaxation of C that contains a tuple t iff t belongs to C or t contains at least one dummy value '$*$'. For instance, if C is a ternary constraint allowing the following set of tuples $\{\langle a, b, a\rangle, \langle b, c, a\rangle, \langle a, b, c\rangle\}$, then the domain of each $S_i, i \leq n$, is $\{a, b, c\}$, the domain of each $S_i, i > n$, is $\{a, b, c, *\}$, and $C' = C \cup \{\langle d_1, d_2, d_3 \rangle \mid \exists i \in 1..3, d_i = *\}$.

The second sequence of variables contains $n + 1$ variables $[D_1, \ldots, D_{n+1}]$ which provide the cumulative count of the number of discrepancies with respect to the previous sequence. Then we introduce the following $k + 3$-ary constraint:

$$C_H(X_i, S_i, \ldots, S_{i+k-1}, D_i, D_{i+1}) \Leftrightarrow C'(S_i, \ldots, S_{i+k-1}) \ \wedge \ D_{i+1} = D_i + (S_i \neq X_i)$$

This constraint ensures that C' is satisfied by the sequence $[S_i, \ldots, S_{i+k-1}]$ (that is, $[S_i, \ldots, S_{i+k-1}]$ satisfies C if $i + k - 1 \leq n$), and $D_{i+1} = D_i$ if $S_i = X_i$, otherwise $D_{i+1} = D_i + 1$. Using the more complex form of Slide over multiple sequences, we can thus reformulate the SoftSlide$_H$ constraint by sliding C_H over the three sequences $[S_1, \ldots, S_{n+k-1}]$, $[X_1, \ldots, X_n]$, and $[D_1, \ldots, D_{n+1}]$ and by constraining D_1 to be 0 and D_{n+1} to be equal to D. That is, we have:

$$\text{SoftSlide}_H(C, [X_1, \ldots, X_n], D)$$

$$\Leftrightarrow$$

$$\text{Slide}(C_H, [S_1, \ldots, S_{n+k-1}], [X_1, \ldots, X_n], [0, D_2 \ldots, D_n, D])$$

Enforcing GAC on Slide is in $O(nkd^k)$, where n is the length of the sequence and k the arity of the constraint being slid. In the case of the reformulation of SoftSlide$_H$ as Slide, the constraint to be slid has arity $k + 3$. Thus, the time complexity of enforcing GAC on SoftSlide$_H$ using this reformulation is in $O(nkd^{k+3})$ where d is the number of values that are used in tuples allowed by the constraint C.

Note that for this encoding to work correctly, the variables $[S_1, \ldots, S_{n+k-1}]$ should not be constrained by other constraints, so that GAC on Slide$(C, [S_1, \ldots, S_{n+k-1}])$ guarantees a solution. Since such variables are introduced during the reformulation, they will be invisible to the users of the SoftSlide$_H$ constraint, hence such an assumption is reasonable.

6.2 SoftSlide$_E$ as SoftRegular$_E$

By using the same reformulation we proposed of Slide as Regular, we can reformulate SoftSlide$_E$ as SoftRegular$_E$. The set of strings accepted by the hard version of the SoftSlide$_E$ constraint must not be empty because the propagator of SoftRegular$_E$ described in [10] is defined for automata accepting a non empty language. This propagator achieves GAC on the variables X_i for $1 \leq i \leq n$ and BC on D in $O(n|\delta| + n|Q|\log(n|Q|))$ steps. The DFA $\mathcal{A}$ has $O(|\Sigma|^i)$ states labelled with a sequence of length i for a total of $|Q| = \sum_{i=0}^{k-1} O(|\Sigma|^i) = O(|\Sigma|^k)$ states. The outgoing degree of every state is bounded by $|\Sigma|$. We therefore have $|\delta| = O(|\Sigma|^{k+1})$ transitions. Filtering the SoftSlide$_E$ constraint therefore requires $O(n|\Sigma|^{k+1} + n|\Sigma|^k \log(n|\Sigma|^k)) = O(n|\Sigma|^{k+1} + nk|\Sigma|^k \log(n|\Sigma|))$ time.

7 Experimental analysis

We now show that reformulation is an effective mechanism to provide propagators for SLIDE and REGULAR constraints. First, we compare reasoning about a REGULAR constraint encoded as SLIDE, with reasoning about it directly using Pesant's GAC propagator [6]. Then, we analyse the reverse reformulation and compare reasoning a SLIDE constraint reformulated as a REGULAR, with reasoning about it directly using the GAC propagator given in [3]. Pesant presents two propagators. We implemented the one that keeps track of all supports for each value in the domain of every variable. Whilst the first set of experiments are done using ILOG Solver 6.1 on a 900 MHz Pentium running Linux Debian, the second set is done using ILOG Solver 6.2 on a 2.8GHz Intel Xeon computer running Linux FC2.

7.1 REGULAR as SLIDE

As in [6], we generated random automata with $|Q|$ states and an alphabet of size $|\Sigma|$. We selected 30% of all possible tuples $(c, q_i) \in \Sigma \times Q$ and randomly chose a state $q_j \in Q$ to form the transition $T(c, q_i) = q_j$. We obtained the set of final states F by randomly selecting 50% of the states in Q. Following Pesant, we used a random variable ordering and random value selection. All experiments are averaged over 30 runs. Table 1 shows the results. We observe that the reformulation of REGULAR constraints in terms of SLIDE is as efficient as and most of the times slightly more efficient than propagating directly the REGULAR constraints. The propagator for SLIDE uses a sequence of built-in TABLE constraints. We conjecture that these are highly optimized and contribute to the performance offered by SLIDE.

We also ran experiments on a model for the Mystery Shopper problem due to Helmut Simonis that appears in CSPLib (prob004). This model contains a large number of AMONG constraints. We represented these AMONG constraints using REGULAR constraints, and again either reasoned with these REGULAR constraints directly using Pesant's propagator or reformulated them using SLIDE constraints.

Results are given in Table 2. All instances solved in the experiments use a time limit of 5 minutes. Both methods achieve GAC on the AMONG constraint, so the search trees are identical and it is only the efficiency of the propagator which differ. Again, reformulation of REGULAR using SLIDE is slightly more efficient.

7.2 SLIDE as REGULAR

We consider a variant of the Nurse Scheduling Problem [4] that consists of generating a schedule for each nurse of shifts duties and days off within a short-term planning period. There are three types of shifts (day, evening, and night). We ensure that (1) each nurse should take a day off or be assigned to an available shift; (2) each shift has a minimum required number of nurses; (3) each nurse work load should be between specific lower and upper bounds; (4) each nurse

n	$\|\Sigma\|$	$\|Q\|$	Regular	Regular as Slide
25	5	10	0.0032	**0.0031**
		20	0.0029	**0.0025**
		40	0.0052	**0.0046**
		80	0.0079	**0.0041**
25	10	10	0.0053	**0.0038**
		20	0.0099	**0.0063**
		40	0.0165	**0.0087**
		80	0.0284	**0.0136**
25	20	10	0.0113	**0.0057**
		20	0.0195	**0.0083**
		40	0.0399	**0.0140**
		80	0.0812	**0.0226**
n	$\|\Sigma\|$	$\|Q\|$	Regular	Regular as Slide
50	5	10	**0.0047**	0.0051
		20	0.0047	**0.0037**
		40	0.0101	**0.0086**
		80	0.0168	**0.0087**
50	10	10	0.0105	**0.0071**
		20	0.0207	**0.0129**
		40	0.0359	**0.0185**
		80	0.0631	**0.0301**
50	20	10	0.0232	**0.0119**
		20	0.0396	**0.0177**
		40	0.0814	**0.0289**
		80	0.1655	**0.0457**

Table 1. Time in seconds to find a sequence satisfying a randomly generated automaton either using Pesant's propagator for the Regular constraint or reformulating it as a Slide constraint

can work at most 5 consecutive days; (5) each nurse must have at least 12 hours of break between two shifts; (6) the shift assigned to a nurse cannot change more than once every three days. We develop two models to solve this problem. In both, we introduce one variable for each nurse and each day, indicating to what type of shift, if any, this nurse is affected on this day. The constraints (1)-(3) are enforced using a set of global cardinality constraints. The constraints (4), (5) and (6) form sequences of respectively 6-ary, binary and ternary constraints. Notice that (4) is monotone, hence we simply post these constraints in both models. The conjunction of constraints (5) and (6) is slid using the tuple encoding of Slide in the first model, and an encoding of Slide using Regular in the second model.

We test the models by using the instances available at *http://www.projectmanagement.ugent.be/nsp.php* in which nurses have no maximum workload, but a set of preferences is to be optimised. We ignore these preferences and post a constraint for bounding the maximum workload to at most 5 day shifts, 4 evening shifts and 2 night shifts per nurses and per week. Similarly, each nurse must have

	Regular			Regular as Slide		
Size	#fails	cpu time	#solved	#fails	cpu time	#solved
10	6	0.01022	9/10	6	**0.00755**	9/10
15	8342	1.19897	32/52	8342	**1.15954**	32/52
20	12960	5.63347	21/35	12960	**3.40063**	21/35
25	6186	1.41279	4/20	6186	**0.87862**	4/20
30	1438	0.72189	3/10	1438	**0.47626**	3/10
35	6297	3.73623	20/56	6297	**2.36849**	20/56

Table 2. Mystery Shopper problem, Regular v. Regular as Slide. #fails and cpu time are only averaged on instances solved by both methods.

	Slide	Slide as Regular
instances solved	56/99	56/99
time	3.77	4.39
backtracks	4761	4761

Table 3. Nurse scheduling problem (30 nurses, 28 days), Slide v. Slide as Regular. #fails and cpu time are only averaged on instances solved by both methods.

at least 2 rest days per week. We solve a sample of 99 instances involving a crew of 30 nurses to schedule over 28 days. We use the same static variable ordering for both models. The days are scheduled in chronological order, and within each day, we allocate a shift to every nurse in lex order. Initial experiments show that this simple heuristic is more efficient than dynamic minimum domain heuristic.

In Table 3, we report the mean fails and cpu time required to solve the instances. We observe that the first model is about 15% faster than the second model.

8 Related Work

Reformulating new global constraints in terms of those that already available within the constraint toolkit has started to gain attention within the constraint programming community. For instance, in [11], the AmongSeq constraint used in car sequencing on a production line is studied and alternative propagation methods are discussed. One approach reformulates AmongSeq as a Regular constraint. This reformulation is shown to be the most efficient in practice compared to the other proposed propagators.

Given the large number of global constraints that have been identified, another direction of study is "general-purpose" global constraints. Such constraints can be used in conjunction with the primitive constraints to reformulate a wide range of global constraints without the need to extend the constraint toolkits. This is especially useful if a constraint toolkit does not provide a propagator for the global constraint or if the constraint is difficult to propagate. Slide is such a general constraint because it helps encode and propagate many sequencing

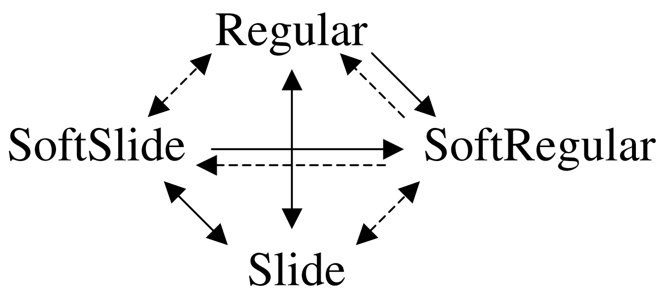

Fig. 2. The relationship between Slide, SoftSlide, Regular, and SoftRegular constraints.

constraints [3]. Other examples of general constraints are Range and Roots [2]. They are shown to be very useful for reformulating diverse global constraints appearing in counting and occurrence problems.

One of the simplest ways to soften the Slide constraint is to relax the number of times the slid constraint holds on the sequence. This gives the CardPath constraint. CardPath$(C, [X_1, \ldots, X_n], N)$ holds iff C holds N times on the sequence $[X_1, \ldots, X_n]$ [1]. Interestingly, the CardPath constraint can itself be reformulated as a Slide constraint [3]. We can therefore use the propagator for Slide to propagate the CardPath constraint. In fact, this reformulation is the first and only method proposed so far in the literature for enforcing GAC on CardPath.

9 Conclusions

To model real-world constraint problems and to solve them efficiently, many global constraints have been proposed in recent years. In this paper, we have focused on two important global constraints, Slide and Regular which are useful for encoding and propagating a wide range of rostering and sequencing problems. Since problems are often over-constrained, we have also studied soft forms of these global constraints. We showed that the different forms of Slide and Regular can all be reformulated as each other. We also showed that reformulation is an effective method to incorporate such global constraints within an existing constraint toolkit. This study has provided insight into the close relationship between these two important global constraints.

The relationships depicted in Figure 2 demonstrate the close links between the hard and soft versions of the Slide and Regular constraints. An arrow from a constraint C_i to a constraint C_j indicates in the figure that C_i can be reformulated as C_j. A thick arrow is either due to findings in this paper or due to the fact that a soft form of a constraint can be used to propagate its hard form by not allowing any violation. The dashed arrows can be obtained by transitivity from the thick arrows. For instance, given that REGULAR can be reformulated as SLIDE which can itself be reformulated as SOFTSLIDE, we can derive that REGULAR can be reformulated as SOFTSLIDE.

References

1. N. Beldiceanu and M. Carlsson. Revisiting the cardinality operator and introducing cardinality-path constraint family. In *Proc. of ICLP'01*, LNCS 2237,pp. 59–73. Springer, 2001.
2. C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. The range and roots constraints: Specifying counting and occurrence problems. In *Proc. of IJCAI'05*, pp. 60–65. Professional Book Center, 2005.
3. C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. The Slide Meta-Constraint. Comic technical report, 2006 (available at http://homes.ieu.edu.tr/ bhnich/comic/).
4. Burke, E. K., Causmaecker, P. D., Berghe, G. V., and Landeghem, H. V. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.
5. M. Maher. Analysis of a global contiguity constraint. In *Proc. of the CP'02 Workshop on Rule Based Constraint Reasoning and Programming*, 2002.
6. G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proc. of CP'04*, LNCS 3258, pp. 482–295. Springer, 2004.
7. T. Petit, J-C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *Proc. of CP'01*, LNCS 2236, pp. 451–463. Springer, 2001.
8. C.-G. Quimper and T. Walsh. Global grammar constraints. In *Proc. of CP'06*, LNCS 4204, pp. 751–755. Springer, 2006.
9. W-J. van Hoeve. A hyper-arc consistency algorithm for the soft alldifferent constraint. In *Proc. of CP'04*, LNCS LNCS 3258, pp. 679–689. Springer, 2004.
10. W-J. van Hoeve, G. Pesant, and L-M. Rousseau. On global warming : Flow-based soft global constaints. *Journal of Heuristics*, 12(4-5):347–373, 2006.
11. W-J. van Hoeve, G. Pesant, L-M. Rousseau, and A. Sabharwal. Revisiting the sequence constraint. In *Proc. of CP'06*, LNCS 42024, pp. 620–634. Springer, 2006.
12. A. Zanarini, M. Milano, and G. Pesant. Improved algorithm for the soft global cardinality constraint. In *Proc. of CP-AI-OR'06*, LNCS 3990, pp. 288–299. Springer, 2006.