Encodings of the SEQUENCE constraint

Sebastian Brand[1], Nina Narodytska[2], Claude-Guy Quimper[3], Peter Stuckey[1], and Toby Walsh[2]

[1] NICTA and University of Melbourne
[2] NICTA and University of NSW
[3] Omega Optimisation

Abstract. The SEQUENCE constraint is useful in modelling car sequencing, rostering, scheduling and related problems. We introduce half a dozen new encodings of the SEQUENCE constraint, some of which do not hinder propagation. We prove that down the whole branch of a search tree domain consistency can be enforced on the SEQUENCE constraint in just $O(n^2 \log n)$ time. This improves upon the previous bound of $O(n^3)$ for each call down the tree. We also consider some generalizations including multiple SEQUENCE constraints, and cyclic SEQUENCE constraints. Our experiments suggest that, on very large and tight problems, domain consistency algorithms are best. However, on smaller or looser problems, much simpler encodings are better, even though these encodings hinder propagation. When there are multiple SEQUENCE constraints, a more expensive propagator shows promise.

1 Introduction

Global constraints are an important factor contributing to the success of constraint programming. They capture common modelling patterns and provide efficient propagators for these patterns. Research has started to show that some global constraints can be efficiently and effectively encoded and propagated using a small number of building blocks. For instance, a wide range of useful global constraints like AMONG, ATMOST, LEX, and STRETCH can be efficiently and effectively encoded using Pesant's REGULAR constraint [1]. Such REGULAR constraints can themselves be efficiently and effectively encoded into ternary transition constraints [2].

Encoding global constraints in this way offers several advantages. First, it is easy to incorporate such encodings into existing solvers. Second, encodings can provide efficient *incremental* propagators. For example, with the ternary encoding of the REGULAR constraints, only those ternary constraints involving variables whose domains have changed need wake up. Third, we can profit from any optimizations to the building blocks used in the encoding. For example, these ternary transition constraints can themselves be encoded using GAC-SCHEMA constraints. Any improvements to propagators for the GAC-SCHEMA constraint will therefore improve the efficiency of the REGULAR constraint. Fourth, encodings can make it easier to construct nogoods for learning and backjumping. Fifth, the encoding gives heuristics an ability to "look inside" the global constraint when making branching decisions.

In this paper we propose and compare half a dozen different encodings of the SEQUENCE constraint. The SEQUENCE constraint was introduced by Beldiceanu and

Contejean [3]. It constrains the number of values taken from a given set in any sequence of k variables. It is useful in staff rostering to specify, for example, that every employee has at least 2 days off in any 7 day period. Another application is car sequencing problems (prob001 in CSPLib). The SEQUENCE constraint can be used to specify, for example, that at most 1 in 3 cars along the production line can have a sun-roof fitted. Several propagators for the SEQUENCE constraint have previously been proposed against which we will compare these new encodings.

2 Background

A constraint satisfaction problem (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for subsets of variables. We use capital letters for variables (e.g. X, Y and S), and lower case for values (e.g. d and d_i). A solution is an assignment of values to the variables satisfying the constraints. Constraint solvers typically explore partial assignments enforcing a local consistency property using either specialized or general purpose propagation algorithms. A *support* for a constraint C is a tuple that assigns a value to each variable from its domain which satisfies C. A *bounds support* is a tuple that assigns a value to each variable which is between the maximum and minimum in its domain which satisfies C. A constraint is *domain consistent* (*DC*) iff for each variable X_i, every value in the domain of X_i belongs to a support. A constraint is *bounds consistent* (*BC*) iff for each variable X_i, there is a bounds support for the maximum and minimum value in its domain. A constraint is *singleton domain consistent* (*SDC*) iff for each variable X_i, we can assign any value in the domain of X_i and make the resulting subproblem domain consistent. Singleton bounds consistency (*SBC*) is defined analogously. A CSP is DC/BC/SDC/SBC iff each constraint is DC/BC/SDC/SBC.

We will compare local consistency properties applied to sets of constraints, c_1 and c_2 which are logically equivalent. As in [4], a local consistency property Φ on c_1 is as strong as Ψ on c_2 iff, given any domains, if Φ holds on c_1 then Ψ holds on c_2; Φ on c_1 is stronger than Ψ on c_2 iff Φ on c_1 is as strong as Ψ on c_2 but not vice versa; Φ on c_1 is equivalent to Ψ on c_2 iff Φ on c_1 is as strong as Ψ on c_2 and vice versa; they are incomparable otherwise.

3 The SEQUENCE constraint

The AMONG constraint restricts the number of occurrences of some given values in a sequence of k variables. More precisely, $\text{AMONG}(l, u, [X_1, X_2, \ldots, X_k], v)$ holds iff $l \leq |\{i | X_i \in v\}| \leq u$. That is, between l and u of the variables take values in v. The AMONG constraint can be encoded by channelling into 0/1 variables using $Y_i \leftrightarrow X_i \in v$ and $l \leq \sum_{i=1}^{k} Y_i \leq u$. Since the constraint graph of this encoding is Berge-acyclic, this does not hinder propagation. Consequently, except in Section 5, 6 and 7, we will simplify notation and consider AMONG (and SEQUENCE) on binary variables Y with $v = \{1\}$. If $l = 0$, AMONG becomes an ATMOST constraint. If $u = k$, AMONG becomes an ATLEAST constraint. ATMOST (and ATLEAST) is *monotone* since given a support, we also have support for any larger (smaller) value [5].

The SEQUENCE constraint is a conjunction of overlapping AMONG constraints. More precisely, SEQUENCE$(l, u, k, [X_1, X_2, \ldots, X_n], v)$ holds iff for $1 \leq i \leq n{-}k{+}1$, AMONG$(l, u, [X_i, X_{i+1}, \ldots, X_{i+k-1}], v)$ holds. We shall refer to the decomposition of the SEQUENCE constraint into a sequence of AMONG constraints as the AD encoding. Clearly, this decomposition hinders propagation. However, if the AMONG constraint is monotone then enforcing *DC* on the decomposition is equivalent to enforcing *DC* on the SEQUENCE constraint [5]. A extension proposed in [6] is that each AMONG constraint can have different parameters (l, u and k). All the encodings proposed here can easily be extended to deal with this generalization.

Several filtering algorithms exist for SEQUENCE constraints. Regin and Puget propose a filtering algorithm for the Global Sequencing constraint (GSC) that combines a SEQUENCE and a global cardinality constraint (GCC) [7]. They encode the GSC constraint into a set of GCC constraints. This encoding hinders propagation as domain consistency on the encoding may not achieve domain consistency on the original SEQUENCE constraint. Beldiceanu and Carlsson propose a greedy filtering algorithm for the CARDPATH constraint that can be used to propagate the SEQUENCE constraint, but this again may not achieve domain consistency [8]. Regin proposes decomposing GSC into a set of variable disjoint AMONG and GCC constraints [9]. Again, this decomposition hinders propagation. Bessiere *et al.* [5] encode SEQUENCE using a SLIDE constraint, and give a domain consistency propagator that runs in $O(nd^{k-1})$ time. Finally, van Hoeve *et al.* [6] propose two filtering algorithms that establish domain consistency. The first algorithm is based on an encoding into a REGULAR constraint and runs in $O(n2^k)$ time (we will refer to this encoding as RE), whilst the second is based on computing cumulative sums and runs in $O(n^3)$ time (we call this $HPRS$ after the initials of the authors). One of our contributions here is to improve on this bound.

3.1 Domain consistency filtering algorithms based on REGULAR (*LO*)

As mentioned above, van Hoeve *et al.* give an encoding using the REGULAR constraint [6]. The states of the automata used in this encoding record which of the last k values encountered are from the set v. We can improve upon this encoding very slightly by having states record just the last $k-1$ values encountered. A transition is then permitted iff the last $k-1$ values encountered plus the current variable have the correct frequency of values from the given set.

We now give an alternative encoding using the REGULAR constraint which exploits two features of many car sequencing and staff rostering problems. First, such problems typically only place upper bounds on occurrences (e.g. at most 1 in 3 cars can have the sun-roof). We will consider therefore just sequences of ATMOST constraints. Any sequence of ATLEAST constraints can be turned into a sequence of ATMOST constraints by inverting the values counted (e.g. at least 2 in any 7 days must be rest days is equivalent to at most 5 in 7 days are work days). Second, in many problems the upper bound u is typically small (e.g. in all data files in Prob001 in CSPLib, $u \leq 2$ and $k \leq 5$).

Suppose we wish to ensure at most 1 in k variables take values from some given set. Consider an automaton whose states record the minimum of k and the distance back to the last occurrence of a value in v. If none of the values in the given set has yet occurred, the distance is taken to be k. The transition function from the state q on seeing Y_i is $t(q, Y_i) = \min(k, q+1)$ if $Y_i = 0$ and $t(q, Y_i) = 1$ if $q = k$ and $Y_i = 1$. The initial state of the automaton is k and any state is accepting. A similar automaton can be constructed for $u > 1$, but we need states to record the distances back to the last u occurrences of values in v.

Thus, to encode a SEQUENCE constraint, we convert it into a sequence of ATLEAST and ATMOST constraints. We then convert the sequence of ATLEAST constraints into a sequence of ATMOST constraints by inverting the value being counted. Finally, we construct the product of the automata for the two sequences of ATMOST constraints. The complexity of enforcing domain consistency on SEQUENCE using this encoding is $O(nk^{\min(l,k-l)+\min(u,k-u)})$. We will refer to this encoding as LO as the automaton records the last occurrence(s).

3.2 Domain consistency filtering algorithm based on cumulative sums (CS)

Our next encoding is based on computing cumulative sums. We introduce a sequence of *cumulative sum* integer variables, S_j where $S_j = \sum_{i=1}^{j} Y_i$ each with domain $[0, n]$. We encode this linearly as $S_j = 0$ and $S_i = Y_i + S_{i-1}$ for $1 \leq i \leq n$. We then post $S_j \leq S_{j+k-1} - l$ and $S_{j+k-1} \leq S_j + u$ for $1 \leq j \leq n-k+1$. We call this the CS encoding. Not surprisingly, this encoding hinders propagation. However, if we enforce a slightly stronger level of local consistency on the encoding, propagation is unhindered.

Theorem 1. *Enforcing singleton bounds consistency on CS achieves domain consistency on* SEQUENCE *and takes $O(n^3)$ time to enforce down the whole branch of a search tree.*

Proof. It is easy to show that if CS is BC then setting each S_i variable to its upper bound u_i, and setting each $Y_i = u_i - u_{i-1}$ gives a solution of CS and the SEQUENCE constraint. Hence SBC on CS clearly enforces DC of SEQUENCE.

For the complexity argument, first note that propagation for CS is $O(n^2)$ having $O(n)$ constraints which can wake at most $O(n)$ times each. A priori it would appear that enforcing SBC at each node down the search tree is $O(n^3)$ since we must check $O(n)$ assignments. But we can show for each assignment, either incremental propagation is $O(n)$ or it is $O(n^2)$ and the assignment causes failure. Since each failure fixes an Y_i variable in any forward computation this can occur at most $O(n)$ times. Hence the total complexity down the tree is $O(n^3)$

Incremental BC of CS after fixing a single Y_i variable, proceeds to modify upper and lower bounds of S_j variables. Either every bound is modified at most once, in which case the propagation is $O(n)$, or some bound is modified twice. We can use the sequence of propagations that cause the bound to be modified twice to modify it again. Applying this sequence repeatedly we eventually wipe out a domain and detect failure. $\square$

We can see the SBC applied to CS as a reworking of the original $HPRS$ algorithm in different terms, with a tighter complexity argument.

3.3 Domain consistency filtering algorithm based on separation theory (ST)

The key constraints in the CS encoding are *separation theory* constraints of the form $S \leq S' + d$, a well studied class because of their connection with shortest path algorithms. We can modify the encoding to use only reified separation theory constraints, and then use efficient methods for handling these constraints. We replace each constraint $S_i = Y_i + S_{i-1}$ by the equivalent $S_i \leq S_{i-1} + 1$, $S_{i-1} \leq S_i$, $Y_i \Leftrightarrow S_{i-1} \leq S_i - 1$. We denote this the ST encoding.

We can convert a conjunction of separation theory constraints C into a weighted directed graph $G_C = (N_C, E_C)$ defined as $N_C = vars(C)$ and $E_C = \{S \xrightarrow{c} S' \mid S \leq S' + c \in C\}$, where $S \xrightarrow{c} S'$ is a directed edge from S to S' with weight c. The connection with shortest path algorithms is folklore

Proposition 1. *C is satisfiable iff G_C contains no negative length cycles. Assuming C is satisfiable then C implies $S \leq S' + c$ iff the shortest path from S to S' in G_C is length $c' \leq c$.* □

We construct a *DC* propagator by using the current Y assignment to construct a conjunction of separation theory constraints C. After checking the satisfiability of C, we then check whether C implies $S_{i-1} \leq S_i - 1$ in which case we set $Y_i = 1$, or if it implies $S_i \leq S_{i-1}$ (the negation of $S_{i-1} \leq S_i - 1$) in which case we set $Y_i = 0$.

Theorem 2. *The separation theory propagator on the ST encoding enforces domain consistency of* SEQUENCE *in $O(n^2 \log n)$ time down the whole branch of a search tree.*

Proof. Suppose that $Y_i = 1$ has no support given the current domains. Since each solution of the SEQUENCE constraint can be extended to a solution of ST, there can be no solution to ST with $Y_i = 1$. Hence the separation theory constraints will imply $S_i \leq S_{i-1}$ and the algorithm will set $Y_i = 0$. The reasoning is analogous for $Y_i = 0$.

Cotton and Maler [10] define incremental algorithms for (a) detecting negative cycles in a weighted directed graph after addition of a new edge in $O(|E| + |N| \log |N|)$, and (b) checking whether the shortest path has changed after addition of a new edge for a set P of pairs of nodes in $O(|E| + |N| \log |N| + |P|)$. For the ST encoding $P = \{(S_i, S_{i-1}), (S_{i-1}, S_i) \mid 1 \leq i \leq n\}$ and $|N_C|$, $|E_C|$, and $|P|$ are all $O(n)$, hence the complexity of incremental propagation after adding a single edge (e.g. when Y_i is fixed) is $O(n \log n)$. Since we only add $O(n)$ edges overall the total complexity over a branch of the search tree is $O(n^2 \log n)$. □

Our current implementation of ST uses incremental all-pairs shortest path algorithms, rather than the single-source shortest path algorithms of [10]. Let s_{ij} be the shortest path from S_i to S_j for $1 \leq i, j \leq n$. Adding a single new arc $S_k \xrightarrow{c} S_l$ then there exists a negative cycle iff $s_{lk} + c < 0$. If no negative cycle exists then we can update all shortest paths variables s_{ij} by $s_{ij} = \min\{s_{ij}, s_{ik} + c + s_{kl}\}$. The cost for adding a single arc is then $O(n^2)$ and hence $O(n^3)$ down the whole branch of the search tree.

3.4 Domain consistency filtering algorithm based on partial sums (PS)

The penultimate encoding is arguably the simplest encoding which gives domain consistency. The PS encoding simply decomposes the constraint into a set of equations based on partial sums: $P_{i,j} = \sum_{l=i}^{j} Y_i$ each with domain $[0, u]$. The PS encoding of the SEQUENCE constraint is $P_{i,i+k-1} \leq u$ and $P_{i,i+k-1} \geq l$ for $1 \leq i \leq n-k+1$ as well as $P_{i,i} = Y_i$ for $1 \leq i \leq n$ and most importantly, all possible ways of adding two of these variables to create another: $P_{i,j} = P_{i,m}+P_{m+1,j}$ for $1 \leq i \leq m < j \leq n, j \leq i+k-1$. Note there are $O(nk^2)$ constraints of the last form.

Lemma 1. *Bounds consistency on the PS encoding enforces domain consistency of the* SEQUENCE *constraint in $O(nk^2u)$ down the whole branch of a search tree.*

Proof. Define domain D to bounds capture C if for each $Y_i + \cdots + Y_j \leq c \in C$, $\max D(P_{i,j}) \leq c$ and for each $Y_i + \cdots + Y_j \geq c \in C$, $\min D(P_{i,j}) \geq c$. Clearly the domain resulting from BC applied to PS bounds captures the AD encoding.

We show that if D is BC with PS and bounds captures C then it also bounds captures C' which results from eliminating the least (or greatest) indexed variable Y_i.

We consider the least variable Y_i, the greatest is similar. Consider Fourier elimination of Y_i. For each pair of constraints in C of the form $Y_i + \cdots + Y_{j_1} \leq c_1$ and $Y_i + \cdots + Y_{j_2} \geq c_2$, Fourier elimination creates the constraint (a) if $j_1 > j_2$ then $Y_{j_2+1} + \cdots + Y_{j_1} \leq c_1 - c_2$, (b) if $j_1 < j_2$ then $Y_{j_1+1} + \cdots + Y_{j_2} \geq c_2 - c_1$, or (c) if $j_1 = j_2$ then $0 \leq c_1 - c_2$. Now since D bounds captures C we have $\max D(P_{i,j_1}) \leq c_1$ and $\min D(P_{i,j_2}) \geq c_2$. For case (a) by BC on the constraint $P_{i,j_1} = P_{i,j_2} + P_{j_2+1,j_1}$ we have $\max D(P_{j_2+1,j_1}) \leq c_1 - c_2$, for (b) BC on $P_{i,j_2} = P_{i,j_1} + P_{j_1+1,j_2}$ gives $\min D(P_{j_1+1,j_2}) \geq c_2 - c_1$, and for (c) the new constraint is $true$ since otherwise $D(P_{i,j_1}) = \emptyset$. Hence the new constraint is bounds captured by D.

To prove DC of SEQUENCE, let C be the AD encoding plus inequalities fixing Y variables in the current domain D (which we assume is BC with PS). Clearly D bounds captures C. Consider any variable Y_i, and eliminate from C in order $Y_1, \ldots, Y_{i-1}, Y_n$, $Y_{n-1}, \ldots, Y_{i+1}$ to obtain C'. Now C' only involves the variable Y_i. By the correctness of Fourier elimination[4] there are solutions of C extending any solution of C'. Since D bounds captures C' by repeated use of the above argument we have that there are solutions to C for each $d \in D(Y_i)$.

For the complexity argument, we note that the domains of the variables in each constraint $P_{i,j} = P_{i,m}+P_{m+1,j}$ can change at most $3u$ times in a forward computation. Each propagation is $O(1)$ hence the overall complexity down a branch is $O(nk^2u)$. □

3.5 A log based encoding of SEQUENCE (*LG*)

Our final encoding (called LG) is based on a simple dynamic program that builds up partial sums on counts. We introduce $L[i,j]$ with domain $[0,u]$ for the partial sums $\sum_{k=j}^{j+2^i-1} Y_k$ where $0 \leq i \leq \lfloor \log k \rfloor$ and $1 \leq j \leq n - 2^i + 1$. Note that $L[i,j] = P_{j,j+2^i-1}$. This requires the constraints: $L[0,j] = Y_j, 1 \leq j \leq n$ and $L[i,j] = L[i-1,j] + L[i-1,j+2^{i-1}], 1 \leq j \leq n, i > 0$. Suppose $k = \sum_{i=1}^{m} 2^{a_i}$ where $a_1 < \ldots < a_m$ (in other words, a_i is the ith bit set in the binary representation of k). We also need the vector, Z_1 to Z_{n-k+1} each with domain $[l,u]$ and the constraint:

$$Z_j = \sum_{i=1}^{m} L[a_i, j + \sum_{k=1}^{i-1} 2^{a_k}]$$

[4] While Fourier is for real variable elimination for the constraints C it coincides with integer elimination.

We have $O(n \log k)$ variables L that are subject to $O(n \log k)$ ternary constraints and $O(n)$ variables Z that are subject to $O(n)$ linear constraints of length $O(\log k)$, therefore we can enforce bounds consistency on this encoding in $O(n \log ku)$ time down the whole branch of a search tree. But this may not achieve domain consistency on the SEQUENCE constraint.

There are a number of redundant constraints which we can add to improve propagation. For example, we can post:

$$Z_j = \sum_{i=1}^{m} L[a_i, j + \sum_{k=i+1}^{m} 2^{a_k}]$$

In fact, we can add any permutation of partial sums that add up to k (we call this modification the LG_R encoding). That is, suppose b_1 to b_m is some permutation of a_1 to a_m. Then we add the constraint:

$$Z_j = \sum_{i=1}^{m} L[b_i, j + \sum_{k=1}^{i-1} 2^{b_k}]$$

It is not hard to show that such additional redundant constraints can help propagation

4 Theoretical comparison

We compare theoretically those encodings on which we may not achieve domain consistency on the SEQUENCE constraint. We will show that we get more propagation with LG than AD, but that AD, CS and LG are otherwise incomparable. It should be noted that during propagation all auxiliary variables in CS, LG, AD and PS encodings will always have ranges as their domains, consequently, bounds consistency is equivalent to domain consistency for them.

Theorem 3. *Bounds consistency on LG is strictly stronger than bounds consistency on AD.*

Proof. Suppose LG is bounds consistent. Consider any AMONG constraint in AD. It is not hard to see how, based on the partial sums in LG, we can construct support for any value assigned to any variable in this AMONG constraint. To show strictness, consider SEQUENCE $(3, 3, 4, [Y_1, \ldots, Y_6], \{1\})$ with $Y_1, Y_2 \in \{0\}$ and $Y_3, \ldots, Y_6 \in \{0, 1\}$. Enforcing bounds consistency on LG fixes $Y_5 = Y_6 = 1$. On the other hand, AD is bounds consistent. □

Theorem 4. *Bounds consistency on CS is incomparable to bounds consistency on AD.*

Proof. Consider SEQUENCE $(1, 1, 3, [Y_1, Y_2, Y_3, Y_4], \{1\})$ with $Y_1 \in \{0\}$ and Y_2, Y_3, $Y_4 \in \{0, 1\}$. Now AD is bounds consistent. In CS, we have $S_0, S_1 \in \{0\}$, $S_2 \in \{0, 1\}$, $S_3, S_4 \in \{1\}$. As S_3 and S_4 are equal, enforcing bounds consistency on CS prunes 1 from the domain of Y_4.

Consider SEQUENCE $(1, 2, 2, [Y_1, Y_2, Y_3, Y_4], \{1\})$ with $Y_3 \in \{0\}$ and $Y_1, Y_2, Y_4 \in \{0, 1\}$. In CS, we have $S_0 \in \{0\}$, $S_1 \in \{0, 1\}$, $S_2, S_3 \in \{1, 2\}$, $S_4 \in \{2, 3\}$. All constraints in CS are bounds consistent. Enforcing bounds consistency on AD prunes 0 from the domains of Y_2 and Y_4. □

From the proof of Theorem 4 it follows that bounds consistency on CS does not enforce domain consistency on SEQUENCE when SEQUENCE is monotone.

Theorem 5. *Bounds consistency on CS is incomparable with bounds consistency on LG.*

Proof. Consider SEQUENCE $(2, 2, 4, [Y_1, Y_2, Y_3, Y_4, Y_5], \{1\})$ with $Y_1 \in \{1\}$ and Y_2, Y_3, Y_4, $Y_5 \in \{0, 1\}$. All constraints in LG are bounds consistent. In CS, we have $S_0 \in \{0\}$, $S_1 \in \{1\}$, $S_2, S_3 \in \{1, 2\}$, $S_4 \in \{2\}$, $S_5 \in \{3\}$. As S_4 and S_5 are ground and $S_5 = S_4 + 1$, Enforcing bounds consistency on CS fixes $Y_5 = 1$.

Consider SEQUENCE $(2, 3, 3, [Y_1, Y_2, Y_3, Y_4], \{1\})$ with $Y_1 = 1$ and Y_2, Y_3, $Y_4 \in \{0, 1\}$. Now CS is bounds consistent. However, enforcing bounds consistency on LG prunes 0 from Y_4. □

Recall that singleton bounds consistency on CS is equivalent to domain consistency on SEQUENCE. We therefore also consider the effect of singleton consistency on the other encodings where propagation is hindered. Unlike CS, singleton bounds consistency on AD or LG may not prune all possible values.

Theorem 6. *Domain consistency on* SEQUENCE *is strictly stronger than singleton bounds consistency on LG.*

Proof. Consider SEQUENCE $(2, 2, 4, [Y_1, Y_2, Y_3, Y_4, Y_5], \{1\})$ with $Y_1 \in \{1\}$ and Y_2, Y_3, Y_4, $Y_5 \in \{0, 1\}$. Consider $Y_5 = 0$ and the LG decomposition. We have $P_{0,1} \in \{1\}$, $P_{0,2}, P_{0,3}, P_{0,4} \in \{0, 1\}$, $P_{0,5} \in \{0\}$, $P_{1,1}, P_{1,2} \in \{1, 2\}$, $P_{1,3}, P_{1,4} \in \{0, 1\}$, $P_{2,1}, P_{2,2} \in \{2\}$. All constraints in LG are bounds consistent. Consequently, we do not detect that $Y_5 = 0$ does not have support. □

Theorem 7. *Domain consistency on* SEQUENCE *is strictly stronger than singleton bounds consistency on AD.*

Proof. By transitivity from Theorems 6 and 3. □

5 The Multiple SEQUENCE constraint (MR)

We often have multiple SEQUENCE constraints applied to the same sequence of variables. For instance, we might insist that at most 1 in 3 cars have the sun roof option and simultaneously that at most 2 in 5 of those cars have electric windows. We propose an encoding for enforcing domain consistency on the conjunction of m such SEQUENCE constraints (we shall refer to this as MR). Suppose that the jth such constraint is SEQUENCE$(l_j, u_j, k_j, [X_1, \ldots, X_n], v_j)$. For simplicity, we suppose also that the values being counted are disjoint. The extension to the non-disjoint case is straightforwards but notationally messy. We channel into a new sequence of variables where $Y_i = j$ if $X_i \in v_j$ else $j = 0$. We now construct an automaton whose states record the last $k' - 1$ values used where k' is the largest k_j. Transitions of the automaton ensure that all SEQUENCE constraints are satisfied. Domain consistency can therefore be enforced using the REGULAR constraint in $O(nm^{k'-1})$ time.

6 The Cyclic SEQUENCE constraint

In rostering problems, we may wish to produce a cyclic schedule which can be repeated, say, every four weeks. We therefore consider a cyclic version of the SEQUENCE constraint. More precisely, $\text{SEQUENCE}_o(l, u, k, [X_1, \ldots, X_n], v)$ ensures that between l and u variables in X_i to $X_{1+(i+k-1 \bmod n)}$ takes values in the set v for $1 \leq i \leq n$.

One simple way to post such a constraint is with a ternary encoding. We introduce state variables Q_i for $1 \leq i \leq n$, each with $O(2^{k-1})$ states to record which of the last $k-1$ values round the cycle are from the set v. We then post ternary constraints of the form $T(X_i, Q_i, Q_{1+(i+1 \bmod n)})$ which hold iff $Q_i = \langle b_1, \ldots b_{k-1} \rangle$, $Q_{1+(i+1 \bmod n)} = \langle (X_i \in v), b_1, \ldots b_{k-2} \rangle$, and:$l \leq \sum_{i=1}^{k-1} b_i + (X_i \in v) \leq u$. This is similar to the ternary encoding of the REGULAR constraint. We can enforce DC on this encoding in $O(n2^{k-1})$ time. However, as the constraint graph of the encoding is cyclic, enforcing domain consistency on the ternary encoding does not achieve domain consistency on SEQUENCE_o in general.

When we have a monotone SEQUENCE_o constraint, decomposition into AMONG (similar to AD) does not hinder propagation. In the non-monotone case, we can enforce domain consistency on SEQUENCE_o using a linear REGULAR constraint with many more states. The states of the automata at index i record which of the last $\min(i-1, k-1)$ values encountered are from the set v, as well as which of the first $\min(i-1, k-1)$ variables from the sequence take values from v. We thus need $O(2^{k-1}2^{k-1}) = O(4^{k-1})$ states. By storing the values used at the start of the sequence, we can test at the end of the sequence if values from v occur with the correct frequency when we wrap around. However, this comes at a considerable cost as enforcing domain consistency takes $O(n4^{k-1})$ time and is unlikely to be useful in practice unless k is very small.

7 Generalizations of SEQUENCE

We consider generalizing the SEQUENCE constraint to replace its parameters l, u, k by integer variables and v by a set variable. A set variable is one which takes a set of values. It is equivalent to a vector of 0/1 variables, representing the characteristic function. Unfortunately enforcing domain consistency on such a SEQUENCE constraint is NP-hard.

Theorem 8. *Enforcing domain consistency on* $\text{SEQUENCE}(l, u, k, [X_1, \ldots, X_n], V)$ *is NP-hard when V is a set variable, and X_i, l, u and k are ground.*

Proof. We reduce 1-in-3 SAT on positive clauses to deciding if a SEQUENCE constraint has a solution. As the X_i in this constraint are ground (but the set variable V is not), enforcing domain consistency on the SEQUENCE constraint is NP-hard. Consider a 1-in-3 SAT problem in N variables (labelled 1 to N) and M positive clauses. We set $\{0\} \subseteq V \subseteq \{0, 1, \ldots, n\}$. V will contain 0 and the set of variables which are true in a satisfying assignment. We set $n = 15M$, $l = 1$, $u = 3$ and $k = 6$. If the ith clause is $x \vee y \vee z$ then we set $\langle X_{15(i-1)+1}, \ldots, X_{15i} \rangle$ to $\langle x, y, z, x, y, z, -1, -1, -1, 0, 0, 0, -1, -1, -1 \rangle$. $\square$

Generalizing l and u by integer variables L and U is tractable if k remains ground and bounded. This would be useful, say, if we wished to minimize the number of night shifts worked in any seven day period. Similarly, generalizing k with an integer variable K is tractable if l and u are fixed. This would be useful, say, if we wished to maximize the minimum period containing more than two night shifts. In both cases, we can encode the resulting constraint using REGULAR. However, it is an open question in both cases if enforcing domain consistency is tractable if we let $k = O(n)$.

8 Experimental Results

Table 1. Randomly generated instances with a single SEQUENCE constraint and $\Delta = 1$. Number of instances solved in 100 sec / average time to solve.

n	k	PS	$HPRS$	ST	AD	GSC	LG	LG_R	CS
50	7	**20** / 0.003	**20** / 0.002	**20** / 0.005	**20** / 0.133	**20** / 0.538	**20** / 0.044	**20** / **0.001**	**20** / 0.002
	15	**20** / 0.023	**20** / **0.001**	**20** / 0.005	**20** / 0.004	**20** / 0.018	**20** / 0.003	**20** / 0.004	**20** / **0.001**
	25	**20** / 0.094	**20** / 0.003	**20** / 0.005	19 / 0.066	19 / 0.396	19 / 0.034	19 / 0.008	**20** / **0.001**
200	7	**20** / 0.016	**20** / 0.030	**20** / 0.242	15 / 2.517	14 / 5.423	17 / **0.003**	**20** / 0.005	**20** / 0.020
	15	**20** / 0.120	**20** / 0.030	**20** / 0.235	7 / 1.850	6 / 0.106	9 / 0.083	17 / 0.021	**20** / **0.016**
	25	**20** / 0.661	**20** / 0.027	**20** / 0.235	3 / 0.005	3 / 0.039	3 / **0.004**	9 / 0.012	**20** / 0.016
	50	**20** / 5.423	**20** / 0.028	**20** / 0.232	4 / 18.255	3 / 1.361	6 / 5.926	10 / 0.113	**20** / **0.014**
500	7	**20** / 0.043	**20** / 0.336	**20** / 4.086	9 / 6.756	8 / 1.046	13 / **0.009**	**20** / 0.017	**20** / 0.150
	15	**20** / 0.320	**20** / 0.334	**20** / 4.130	4 / 13.442	3 / 0.121	6 / **0.012**	14 / 0.059	**20** / 0.100
	25	**20** / 1.816	**20** / 0.279	**20** / 4.017	1 / **0**	1 / **0**	3 / 0.013	5 / 0.020	**20** / 0.085
	50	**20** / 16.762	**20** / 0.290	**20** / 4.032	0 / **0**	0 / **0**	2 / 11.847	4 / 0.031	**20** / 0.086
TOTALS									
solved in 100 sec/total		**220** /220	**220** /220	**220** /220	102 /220	97 /220	118 /220	158 /220	**220** /220
avg time for solved in 100 sec		2.298	0.124	1.566	2.376	1.115	0.524	**0.021**	0.045
avg bt for solved in 100 sec		**0**	**0**	**0**	42830	4319	3239	25	33

Table 2. Randomly generated instances with a single SEQUENCE constraint and $\Delta = 3$. Number of instances solved in 100 sec / average time to solve.

n	k	PS	$HPRS$	ST	AD	GSC	LG	LG_R	CS
50	7	**20** / 0.004	**20** / **0.001**	**20** / 0.006	**20** / **0.001**	**20** / 0.003	**20** / **0.001**	**20** / **0.001**	**20** / 0.002
	15	**20** / 0.022	**20** / 0.001	**20** / 0.006	**20** / 0.002	**20** / 0.004	**20** / **0**	**20** / 0.005	**20** / 0.001
	25	**20** / 0.097	**20** / 0.002	**20** / 0.005	**20** / **0.001**	**20** / 0.007	**20** / **0.001**	**20** / 0.003	**20** / 0.002
200	7	**20** / 0.017	**20** / 0.028	**20** / 0.251	**20** / **0.003**	**20** / 0.016	**20** / 0.004	**20** / 0.007	**20** / 0.028
	15	**20** / 0.127	**20** / 0.036	**20** / 0.242	19 / **0.004**	19 / 0.025	19 / 0.005	**20** / 0.027	**20** / 0.026
	25	**20** / 0.652	**20** / 0.031	**20** / 0.240	18 / 0.252	18 / 3.160	19 / 0.126	**20** / **0.012**	**20** / 0.019
	50	**20** / 5.441	**20** / 0.036	**20** / 0.237	13 / 0.063	13 / 0.571	16 / 4.556	19 / 4.178	**20** / **0.016**
500	7	**20** / 0.047	**20** / 0.414	**20** / 4.226	**20** / **0.010**	**20** / 0.082	**20** / 0.012	**20** / 0.022	**20** / 0.278
	15	**20** / 0.351	**20** / 0.377	**20** / 4.159	17 / **0.013**	17 / 0.113	18 / 0.016	**20** / 0.090	**20** / 0.216
	25	**20** / 1.812	**20** / 0.445	**20** / 4.116	8 / **0.015**	8 / 0.139	11 / 0.019	**20** / 0.040	**20** / 0.141
	50	**20** / 16.711	**20** / 0.383	**20** / 4.056	6 / 0.022	6 / 0.201	7 / **0.019**	12 / 0.123	**20** / 0.099
TOTALS									
solved in 100 sec/total		**220** /220	**220** /220	**220** /220	181 /220	181 /220	190 /220	211 /220	**220** /220
avg time for solved in 100 sec		2.298	0.160	1.595	**0.035**	0.394	0.402	0.403	0.075
avg bt for solved in 100 sec		**0**	**0**	**0**	352	352	1964	970	30

To compare performance with the different encodings, we carried out a series of experiments. The first series used randomly generated instances so we could control the parameters precisely. The second series used some nurse rostering benchmarks to test more realistic situations.

Table 3. Randomly generated instances with a single SEQUENCE constraint and $\varDelta = 5$. Number of instances solved in 100 sec / average time to solve.

n	k	PS	$HPRS$	ST	AD	GSC	LG	LG_R	CS
50	7	**20** / 0.004	**20** / **0.001**	**20** / 0.005	**20** / **0.001**	**20** / 0.003	**20** / 0.002	**20** / **0.001**	**20** / 0.002
	15	**20** / 0.025	**20** / **0.001**	**20** / 0.006	**20** / **0.001**	**20** / 0.005	**20** / **0.001**	**20** / 0.005	**20** / 0.002
	25	**20** / 0.100	**20** / **0.001**	**20** / 0.006	**20** / **0.001**	**20** / 0.008	**20** / 0.002	**20** / 0.003	**20** / 0.002
200	7	**20** / 0.016	**20** / 0.020	**20** / 0.262	**20** / **0.003**	**20** / 0.017	**20** / 0.005	**20** / 0.007	**20** / 0.024
	15	**20** / 0.133	**20** / 0.031	**20** / 0.251	**20** / **0.005**	**20** / 0.026	**20** / 0.005	**20** / 0.034	**20** / 0.028
	25	**20** / 0.665	**20** / 0.030	**20** / 0.242	**20** / 0.007	**20** / 0.038	**20** / **0.006**	**20** / 0.014	**20** / 0.020
	50	**20** / 5.538	**20** / 0.039	**20** / 0.242	**20** / 0.012	**20** / 0.073	**20** / **0.010**	**20** / 0.018	**20** / 0.019
500	7	**20** / 0.047	**20** / 0.195	**20** / 4.362	**20** / **0.012**	**20** / 0.085	**20** / **0.012**	**20** / 0.023	**20** / 0.154
	15	**20** / 0.358	**20** / 0.383	**20** / 4.183	**20** / **0.015**	**20** / 0.119	**20** / 0.017	**20** / 0.101	**20** / 0.235
	25	**20** / 1.786	**20** / 0.411	**20** / 4.127	**20** / **0.019**	**20** / 0.146	**20** / 0.021	**20** / 0.045	**20** / 0.201
	50	**20** / 17.016	**20** / 0.342	**20** / 4.077	11 / 0.034	11 / 0.298	12 / **0.033**	17 / 4.869	**20** / 0.120
TOTALS									
solved in 100 sec/total		**220** /220	**220** /220	**220** /220	211 /220	211 /220	212 /220	217 /220	**220** /220
avg time for solved in 100 sec		2.335	0.132	1.615	**0.009**	0.065	**0.009**	0.405	0.073
avg bt for solved in 100 sec		**0**	**0**	**0**	2	2	1	1268	19

8.1 Random instance

For each possible combination of $n \in \{50, 200, 500\}$, $k \in \{7, 15, 25, 50\}$, $\varDelta = u - l \in \{1, 3, 5\}$, we generated twenty instances with random lower bounds in the interval $[0, k - \varDelta)$. We used a random value and variable ordering and a time-out of 100 sec. Results for different values of $\varDelta$ are presented in Tables 1– 3. Instances can be partitioned into 2 groups. In the first group, $n > 50$ and $\varDelta < 3$. On these instances, assignment of one variable has a strong impact on other variables. In the extreme case when $\varDelta = 0$ instantiation of one variable assigns on average another n/k variables. So, we expect DC propagators to significantly shrink variables domains and reduce the search tree. As can be seen from Tables 1, DC propagators outperform non-DC propagators. Surprisingly, CS has the best time of all combinations and solved all instances. Whilst it takes more backtracks compared to the DC propagators which solve problems without search, it is much faster. The ST algorithm is an order of magnitude slower compared to the $HPRS$ propagator but in the current implementation we use a naive $O(n^2)$ algorithm to update the adjacency matrix. The PS algorithm is much slower compared to other DC algorithms and it relative performance decays for larger k's.

In the second group, $n \leq 50$ or $\varDelta \geq 3$. On these instances, assignment of one variable does not have big influence on other variables. The overhead of using DC propagators to achieve better pruning outweighs the reduction in the search space. The clear winner in this case are those propagators which do not achieve DC. When $k < 25$ AD is best. When k gets larger, LG and LG_R solve more instances and work faster due to their better propagation.

8.2 Nurse Rostering Problems

All instances are taken from `http://www.projectmanagement.ugent.be/nsp.php`. The basic model includes the following three constraints: each shift has a minimum required number of nurses, each nurse should have at least 12 hours of break between 2 shifts, each nurse should have at least two consecutive days on any shift. In addition, each model has additional SEQUENCE constraints. For each day in the scheduling period, a nurse is assigned to a day (D), evening (E), or night (N) shift or takes a day-off (O). We introduce one variable $X[i,j]$ for each nurse, $i = 1 \ldots p$ and each day, $j = 1 \ldots n$, where p is the number of nurses, n is the number of days in the scheduling period. Each model was run on 50 instances. The number of nurses in each instance was set to the maximal number of nurses required for any day over the period of 14 days. The time limit for all instances was 100 sec. For variable ordering, we branched on the smallest domain.

Table 4 gives results for those instances that were solved by each propagator. In these experiments $n < 50$. As expected from the random experiments , the AD decomposition outperforms all other decompositions. The only exception are instances with $\Delta = 0$ and non-DC propagators lose to DC algorithms and the CS decomposition.

Table 4. Models of the nurse rostering problem using the SEQUENCE constraint. Number of instances solved in 100 sec / average time to solve.

SEQUENCE	RE	PS	LO	$HPRS$	ST	AD	GSC	LG	LG_R	CS
$(1,3,3,\{O\})$	**43** / 0.71	**43** / 0.47	**43** / 0.64	**43** / 0.54	**43** / 0.63	**43** / **0.45**	**43** / 0.60	**43** / **0.45**	**43** / 0.47	39 / 2.49
$(3,5,5,\{O\})$	**44** / 2.32	**44** / 2.43	**44** / 2.11	**44** / 2.28	**44** / 2.61	**44** / **1.84**	**44** / 2.84	**44** / 1.87	**44** / 1.92	40 / 3.53
$(2,2,5,\{O\})$	39 / **4.48**	39 / 5.41	39 / 4.76	**40** / 7.41	38 / 4.97	36 / 7.50	35 / 7.73	36 / 8.56	**40** / 5.43	36 / 5.36
$(2,2,7,\{O\})$	**23** / 7.35	**23** / 9.09	**23** / 7.92	**23** / 5.64	**23** / 7.50	22 / 11.16	22 / 18.21	22 / 11.66	**23** / 8.10	**23** / **4.53**
$(2,3,5,\{O\})$	26 / 4.44	26 / 4.65	26 / 4.92	**27** / 6.77	26 / **3.91**	**27** / 5.47	26 / 4.27	**27** / 5.81	**27** / 6.19	26 / 5.77
$(2,5,7,\{O\})$	22 / 2.77	22 / 3.45	**23** / 6.90	22 / 2.22	22 / 2.43	**23** / 6.06	22 / 3.28	22 / **2.10**	22 / 2.45	22 / 2.28
$(1,3,4,\{O\})$	26 / 4.82	**27** / 7.02	26 / 5.25	**27** / 6.75	26 / **4.35**	**27** / 5.80	26 / 4.69	**27** / 6.05	**27** / 6.05	25 / 6.18
TOTALS										
solved in 100 sec/total	223 /350	224 /350	224 /350	**226** /350	222 /350	222 /350	218 /350	221 /350	**226** /350	211 /350
avg time for solved in 100 sec	3.49	4.17	4.07	4.26	**3.47**	4.78	5.17	4.68	3.95	4.22
avg bt for solved in 100 sec	8527	11045	9905	13804	**8017**	20063	12939	18715	14205	17747

8.3 Multiple SEQUENCE constraints

Table 5. Randomly generated instances with 2 SEQUENCE constraints and $\Delta = 1$. Number of instances solved in 100 sec / average time to solve.

n	k	MR	RE	PS	LO	$HPRS$	ST	AD	GSC	LG	LG_R	CS
50	5	**20** / **0**	18 / 0.22	18 / 0.18	18 / 0.27	18 / 0.34	18 / 1.44	18 / 0.65	18 / 2.30	18 / 0.19	18 / 0.24	5 / 36.45
	7	**20** / **0.05**	14 / 3.01	14 / 3.97	14 / 3.58	14 / 3.06	13 / 0.36	13 / 0.85	13 / 2.62	13 / 0.24	14 / 3.42	8 / 11.02
100	5	**20** / 0.01	15 / 0.01	15 / **0**	15 / 0.01	15 / 0.01	15 / 0.09	15 / 1.38	15 / 6.21	15 / **0**	15 / 0.18	0 / **0**
	7	**20** / 0.10	11 / 0.02	11 / 0.02	11 / 0.02	11 / 0.01	11 / 0.08	7 / 8.89	6 / 0.51	9 / 10.16	10 / **0**	1 / **0**
TOTALS												
solved in 100 sec/total		**80** /80	58 /80	58 /80	58 /80	58 /80	57 /80	53 /80	52 /80	55 /80	57 /80	14 /80
avg time for solved in 100 sec		**0.04**	0.80	1.02	0.96	0.85	0.57	1.99	3.30	1.78	0.96	19.31
avg bt for solved in 100 sec		**0**	4771	4771	4771	4771	710	43390	22915	28894	8887	553515

We also evaluated performance of the different propagators on problems with multiple SEQUENCE constraints. We again used randomly generated instances and nurse rostering problems. For each possible combination of $n \in \{50, 100\}$, $k \in \{5, 7\}$, $\Delta = 1$ and $m \in \{2, 3, 4\}$ (where m is the number of SEQUENCE constraints), we generated twenty random instances. All variables had domains of size 5. An instance was obtained by selecting random lower bounds in the interval $[0, k - \Delta]$. We excluded

Table 6. Randomly generated instances with 3 SEQUENCE constraints and $\Delta = 1$. Number of instances solved in 100 sec / average time to solve.

n	k	MR	RE	PS	LO	$HPRS$	ST	AD	Gsc	LG	LG_R	CS
50	5	**20** / 0.02	10 / 0.63	10 / 0.50	10 / 0.72	10 / 0.67	10 / 4.01	11 / 9.45	10 / 1.32	11 / 6.59	11 / 7.92	1 / **0**
	7	**20** / 0.35	8 / 0.36	8 / 0.33	8 / 0.45	8 / **0.25**	8 / 1.79	9 / 7.67	8 / 4.33	9 / 12.47	9 / 6.02	3 / 33.39
100	5	**20** / 0.04	1 / **0**	1 / **0**	1 / **0**	1 / **0**	1 / **0**	1 / **0**	1 / **0**	1 / **0**	1 / **0**	0 / **0**
	7	**20** / 0.42	3 / 0.33	3 / 0.47	3 / 0.38	3 / 0.46	3 / 7.85	1 / **0**	1 / **0**	1 / **0**	2 / 1.22	2 / 1.91
TOTALS												
solved in 100 sec/total		**80** /80	22 /80	22 /80	22 /80	22 /80	22 /80	22 /80	20 /80	22 /80	23 /80	6 /80
avg time for solved in 100 sec		**0.21**	0.46	0.41	0.54	0.46	3.54	7.86	2.40	8.40	6.25	17.33
avg bt for solved in 100 sec		**0**	3512	3512	3512	3512	3512	140670	11815	131464	90217	369357

Table 7. Randomly generated instances with 4 SEQUENCE constraints and $\Delta = 1$. Number of instances solved in 100 sec / average time to solve.

n	k	MR	RE	PS	LO	$HPRS$	ST	AD	Gsc	LG	LG_R	CS
50	5	**20** / 0.05	6 / 14.93	6 / 12.58	6 / 17.03	5 / 0.81	5 / 4.76	6 / 13.75	5 / 10.59	7 / 15.05	6 / 6.86	0 / **0**
	7	**20** / 0.86	7 / 25.76	6 / 20.85	6 / 16.89	7 / 23.99	4 / **0.15**	6 / 14.02	5 / 15.90	8 / 26.81	6 / 27.34	2 / 5.80
100	5	**20** / 0.11	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**
	7	**20** / 1.83	2 / 6.54	2 / 8.98	2 / 7.52	2 / 10.48	1 / **0**	1 / **0**	1 / **0**	1 / **0**	2 / 0.01	0 / **0**
TOTALS												
solved in 100 sec/total		**80** /80	15 /80	14 /80	14 /80	14 /80	10 /80	13 /80	11 /80	16 /80	14 /80	2 /80
avg time for solved in 100 sec		**0.71**	18.86	15.61	15.61	13.78	2.44	12.81	12.04	19.99	14.66	5.80
avg bt for solved in 100 sec		**0**	91795	72078	72078	64579	2214	185747	51413	257831	120511	106008

instances where $\sum_{i=1}^{m} l_i \geq k$ to avoid unsatisfiable instances. We used a random variable and value ordering, and a time-out of 100 sec. All SEQUENCE constraints were enforced on disjoint sets of cardinality one.

Experimental results for instances with $\Delta = 1$ and different number of SEQUENCE constraints are presented in Tables 5– 7. They show that the multiple SEQUENCE propagator significantly outperforms other propagators in both metrics, namely time to find a valid sequence and the number of solved instances. For bigger values of n, the multiple SEQUENCE is the only filtering algorithm that successfully solved all instances. However, it should be noted that, due to its space complexity, to use this propagator successfully, k and m should be relatively small and n should be less than 100.

In the second series of experiments we used nurse scheduling problems benchmarks. We removed the last constraint from the basic model described in the previous section and added two sets of non-monotone SEQUENCE constraints to give two different models. In the first model, each nurse has to work one or two night shifts in 7 consecutive days, one or two evening shifts, one to five day shifts and two to five days-off. In the second model, each nurse has to work one or two night shifts in 7 consecutive days, and one or two days-off in 5 days. The number of nurses was equal to maximal number of nurses required for any day over the period multiplied by 1.5. Table 8 shows the number of instances solved by each propagator. The multiple SEQUENCE propagator solved more instances compared to the other propagators.

In a final set of experiments we compared performance of MR, LO and AD propagators on monotone SEQUENCE constraints. We used the same basic model for nurse rostering but augmented it with 5 monotone constraints. The first model includes the following SEQUENCE constraints: at least one day-off in 4 consecutive days, at least one night shift in 7 days, at most two night shifts in 7 days, at least one evening shift in 7 days, and at least one day shift in 7 days. In the second model, we replaced the at

Table 8. Models of the nurse rostering problem using the SEQUENCE constraint. Number of instances solved in 100 sec / average time to solve.

	MR	RE	PS	LO	$HPRS$	ST	AD	GSC	LG	LG_R	CS
Model 1	**16** / 8.52	7 / 0.48	7 / 0.52	7 / 0.44	7 / 0.28	7 / 0.42	7 / 0.24	7 / 0.82	7 / 0.24	7 / 0.33	7 / **0.20**
Model 2	**21** / 0.59	9 / 0.05	9 / 0.04	9 / 0.05	9 / **0.02**	9 / **0.02**	10 / 9.18	9 / 0.04	10 / 8.72	9 / **0.02**	10 / 8.71
TOTALS											
solved in 100 sec/total	**37** /100	16 /100	16 /100	16 /100	16 /100	16 /100	17 /100	16 /100	17 /100	16 /100	17 /100
avg time for solved in 100 sec	4.02	0.24	0.25	0.22	**0.13**	0.20	5.50	0.38	5.22	0.16	5.21
avg bt for solved in 100 sec	11446	**884**	**884**	**884**	**884**	**884**	59688	**884**	59686	**884**	59687

least one day shift in 7 consecutive days constraint with at most two days shifts in 5 days which has a similar tightness. The third model is identical to the second model, but at least one evening shift in 7 consecutive days constraint is replaced by at most two evening shifts in 5 days. Experimental results for all models are presented in the Table 9. All models have approximately the same tightness, but as can be seen from Table 9, the use of the multiple SEQUENCE propagator gives larger performance gains when the model includes several sequences of at least constraints.

Table 9. Models of the nurse rostering problem using the SEQUENCE constraint. Number of instances solved in 100 sec / average time to solve.

	MR	LO	AD
Model 1	**25** / 9.33	11 / 5.55	2 / **0.01**
Model 2	**15** / 5.93	14 / **3.29**	10 / 8.37
Model 3	18 / 1.40	18 / **1.01**	**21** / 3.85
TOTALS			
solved in 100 sec/total	**58** /150	43 /150	33 /150
avg time for solved in 100 sec	5.991	**2.916**	4.989
avg bt for solved in 100 sec	7192	**4423**	47348

8.4 Cyclic SEQUENCE constraint

We end with experiments on the cyclic SEQUENCE constraint. As might be expected, we could not use the *DC* propagator because it requires too much memory. We use instead the first approach described in Section 6 to propagate this constraint. As can be seen from the Table 10, results are similar to the non-cyclic case. The multiple SEQUENCE propagator outperforms other propagators when the problem includes several at least constraints. It should be noted that in the cyclic case, problem instances are much harder. We solved only half the instances compared to the non-cyclic case.

Table 10. Models of the nurse rostering problem using the cyclic SEQUENCE constraint. Number of instances solved in 100 sec / average time to solve.

	MR	LO	AD
Model 1	**12** / 9.02	7 / 0.56	2 / **0.14**
Model 2	**6** / 2.31	**6** / 1.25	2 / **0.03**
Model 3	13 / 5.41	13 / 8.25	**14** / **1.74**
TOTALS			
solved in 100 sec/total	**31** /150	26 /150	18 /150
avg time for solved in 100 sec	6.205	4.562	**1.374**
avg bt for solved in 100 sec	5178	**2705**	8201

9 Conclusion

The SEQUENCE constraint is useful in modelling a range of rostering, scheduling and car sequencing problems. We proved that down the whole branch of a search tree domain consistency can be enforced on the SEQUENCE constraint in just $O(n^2 \log n)$ time. This improves upon the previous bound of $O(n^3)$ for each call down the branch [6]. To propagate the SEQUENCE constraint, we introduced half a dozen new encodings, some of which do not hinder propagation. We also considered some generalizations including multiple SEQUENCE constraints, and cyclic SEQUENCE constraints. Our experiments suggest that, on very large and tight problems, the existing domain consistency algorithm is best. However, on smaller or looser problems, much simpler encodings are better, even though these encodings hinder propagation. When there are multiple SEQUENCE constraints, especially when we are forcing values to occur, a more expensive propagator shows promise. This study raises a number of questions. For example, what other global constraints can be efficiently and effectively propagated using simple encodings? As a second example, can we design heuristics to choose an effective encoding automatically?

References

1. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Proceedings of 10th Int. Conf. on Principles and Practice of Constraint Programming (CP'04). (2004)
2. Quimper, C.G., Walsh, T.: Global grammar constraints. In: In Proceedings of the 12th Int. Conf. on Principles and Practice of Constraint Programming. (2006)
3. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling **12** (1994)
4. Debruyne, R., Bessière, C.: Some practicable filtering techniques for the constraint satisfaction problem. In: Proceedings of the 15th IJCAI. (1997) 412–417
5. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: The slide meta-constraint. Technical report (2007)
6. Hoeve, W.J.v., Pesant, G., Rousseau, L.M., Sabharwal, A.: Revisiting the sequence constraint. In: Proceedings of the 12th Int. Conf. on Principles and Practice of Constraint Programming (CP '06). (2006)
7. Régin, J.C., Puget, J.F.: A filtering algorithm for global sequencing constraints. In: Proceedings of the 3th Int. Conf. on Principles and Practice of Constraint Programming (CP '97). (1997)
8. Beldiceanu, N., Carlsson, M.: Revisiting the cardinality operator and introducing cardinality-path constraint family. In Codognet, P., ed.: Proceedings of the Int. Conf. on Logic Programming (ICLP 2001), (2001)
9. Régin, J.C.: Combination of among and cardinality constraints. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second Int. Conf.. (2005)
10. Cotton, S., Maler, O.: Fast and flexible difference constraint propagation for DPLL(T). In: Proceedings of Theory and Applications of Satisfiability Testing (SAT-2006). (2006)