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 a 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 a generalization of the SEQUENCE constraint – the Multiple SEQUENCE constraint. 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, encodings can make it easier to construct nogoods for learning and backjumping. Fourth, 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 CSP is DC/BC iff each constraint is DC/BC. A CSP 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. Consider, for example, a problem with two constraints: $X_1 \neq X_2$, $X_1 + X_2 = X_3$, where $D(X_1) = \{0, 1, 2\}$, $D(X_2) = \{1, 3\}$ and $D(X_3) = \{1, 2, 3\}$. All constraints are domain consistent, but enforcing SDC on these constraints removes the value 1 from the domain of X_1 and the value 2 from the domain of X_3. A CSP is *singleton bounds consistent* (*SBC*) iff for each variable X_i, we can assign the maximum (minimum) value of X_i and make the resulting subproblem bounds consistent. In the previous example enforcing SBC does not prune any values. A constraint is *monotone* iff there exists a total ordering $\prec$ of the domain values such that for any two values v, w if $v \prec w$ then v can be replaced by w in any support for C.

We will compare local consistency properties applied to sets of constraints, c_1 and c_2, that 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. For the complexity results, we assume the propagation engine wakes each propagator whose variables are changed in constant time per propagator, and that the propagation cost for a linear constraint on n variables is $O(n)$. Modern propagation engines respect this.

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 $l \leq |\{i \mid X_i \in v\}| \leq u$. 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 we will simplify notation and consider AMONG (and SEQUENCE) on Boolean 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 AMONG decomposition (*AD*). 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]. An 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. Régin and Puget propose a filtering algorithm for the Global Sequencing constraint (GSC) that combines a SEQUENCE and a Global Cardinality constraint (GCC) [7]. GSC$([X_1, \ldots, X_n], l, u, v, k, [l_1, \ldots, l_m], [u_1, \ldots, u_m])$ is satisfied iff for each $i \in \{1, \ldots, m\}$, $l_i \leq |\{j \mid X_j = i\}| \leq u_i$ and for each $p \in \{1, \ldots, n-k\}$, $l \leq |\{j \mid X_j \in v \,\&\, p \leq j \leq p+k-1\}| \leq u$. 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]. Régin proposes decomposing GSC into a set of variable disjoint AMONG and GCC constraints [9]. Again, this decomposition hinders propagation. Bessière *et al.* [5] encode SEQUENCE using a SLIDE constraint, and give a domain consistency propagator that runs in $O(nd^{k-1})$ time (d is the maximal domain size). 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, 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. The REGULAR $([X_1, \ldots, X_n], \mathcal{A})$ constraint ensures that the string defined by the sequence

of variables $X_1, \ldots, X_n$ is accepted by the deterministic finite automaton (DFA) $\mathcal{A}$. The encoding 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). Second, in many problems the lower and upper bounds are typically small (e.g. in all data files in Prob001 in CSPLib, $u \leq 2$ and $k \leq 5$).

Suppose we wish to ensure that at most 1 in k Boolean variables Y_i take the value 1. Consider an automaton whose states record the minimum of k and the distance back to the last occurrence of 1. If 1 has not 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 (Figure 1(a)). A similar automaton can be constructed for $u > 1$, but we need states to record the distances back to the last u occurrences of value 1. Now, suppose we wish to ensure at least 1 in k variables take the value 1. The states of the automaton record the distance back to the last occurrence of 1. If 1 has not yet occurred, the distance is taken to be the number of variables seen so far. The transition function from the state q on seeing Y_i is $t(q, Y_i) = q + 1$ if $Y_i = 0$ and $q < k$, and $t(q, Y_i) = 1$ if $q \leq k$ and $Y_i = 1$. The initial state of the automaton is 1 and any state is accepting (Figure 1(b)).

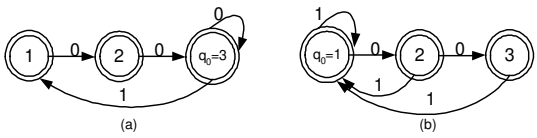

Fig. 1. (a) An automaton for the ATMOSTSEQ constraint with $u = 1$ and $k = 3$. (b) An automaton for the ATLEASTSEQ constraint with $l = 1$ and $k = 3$.

Thus, to encode a SEQUENCE constraint, we convert it into a sequence of ATLEAST and ATMOST constraints. We can convert the sequence of ATLEAST constraints into a sequence of ATMOST constraints (or vice versa depending on which representation gives smaller complexity) by inverting the value being counted. For example,the constraint that at least 3 in any 5 days must be work days is equivalent to at most 2 in 5 days are rest days. Finally, we construct the product of the automata for the two sequences of ATMOST or ATLEAST 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 $[0, j]$. We encode this linearly as $S_0 = 0$ and $S_i = Y_i + S_{i-1}$ for $1 \leq i \leq n$. We then post $S_j \leq S_{j+k} - l$ and $S_{j+k} \leq S_j + u$ 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. *Singleton bounds consistency on CS enforces domain consistency on* SEQUENCE *and takes $O(n^3)$ time to enforce down a 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 $O(n^3)$. Using shadow variables for the Y_i we can disconnect the SBC propagation of the *CS* encoding from the rest of the problem. Since copying from the Y_i to their shadows and back is $O(n)$ the implementation of SBC 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. □

Consider, for example, SEQUENCE$(1, 2, 2, [Y_1, Y_2, Y_3, Y_4], \{1\})$, $D(Y_3) = \{0\}$ and $D(Y_i) = \{0, 1\}, i \in \{1, 2, 4\}$. Corresponding cumulative sum variables S have the following domains: $S_0 \in \{0\}$, $S_1 \in \{0, 1\}$, $S_2, S_3 \in \{1, 2\}$, $S_4 \in \{2, 3\}$. All constraints in *CS* are bounds consistent. However, enforcing singleton bounds consistency on *CS* prunes the value 0 from the domains of Y_2 and Y_4. We can see 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 difference constraints (*CD*)

The key constraints in the *CS* encoding are difference constraints of the form $S \leq S' + d$, a well studied class with connections to shortest path algorithms [10]. We can modify the encoding to use only reified difference 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 *CD* encoding.

We can convert a conjunction of difference 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 well known:

Proposition 1. *(Theorem 1 from [10]) 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 difference 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 difference constraints propagator on the CD encoding enforces domain consistency of* SEQUENCE *in $O(n^2 \log n)$ time down a 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 *CD*, there can be no solution to *CD* with $Y_i = 1$. Hence the difference 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 *CD* 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 *CD* 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 $s_{ij} = \min\{s_{ij}, s_{ik} + c + s_{lj}\}$. The cost for adding a single arc is then $O(n^2)$ and hence $O(n^3)$ down a branch of the search tree.

Consider, for example, SEQUENCE$(1, 2, 2, [Y_1, Y_2, Y_3, Y_4], \{1\})$, $D(Y_i) = \{0, 1\}$, $i \in \{1, \ldots, 4\}$. The initial constraint graph and the corresponding transitive closure of its adjacency matrix are presented in Figure 2(a). Assigning Y_3 to 0 causes addition of the edge from S_3 to S_2 with cost 0. After updating all shortest path variables s, we get that $s_{1,2}$ and $s_{3,4}$ are equal to -1. Consequently, value 0 can be pruned from domains of Y_2 and Y_4 (Figure 2(b)).

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

The fourth 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: $P_{i,j} = \sum_{l=i}^{j} Y_l$ each with domain $[0, \min(u, j - i + 1)]$. 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 a branch of a search tree.*

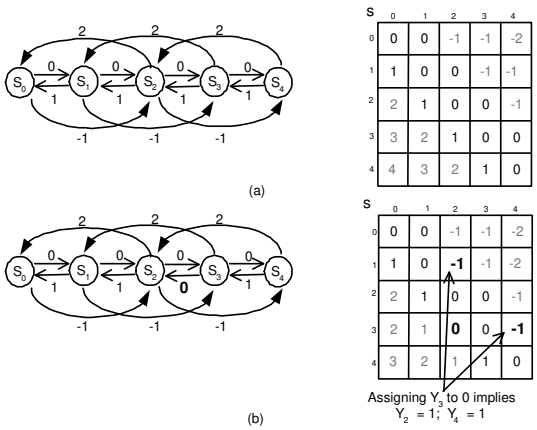

Fig. 2. The *CD* encoding for SEQUENCE$(1, 2, 2, [Y_1, Y_2, Y_3, Y_4], \{1\})$, $D(Y_i) = \{0, 1\}$, $i \in \{1, \ldots, 4\}$. (a) The initial constraint graph and the transitive closure of its adjacency matrix. (b) The constraint graph and the transitive closure of its adjacency matrix after assigning Y_3 to 0.

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 $O(nk^2u)$. □

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

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 variables $L_{i,j}$ with domain $[0, \min(u, 2^{i-1})]$ 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

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

Figure 3 shows the intra-variable dependencies for an example.

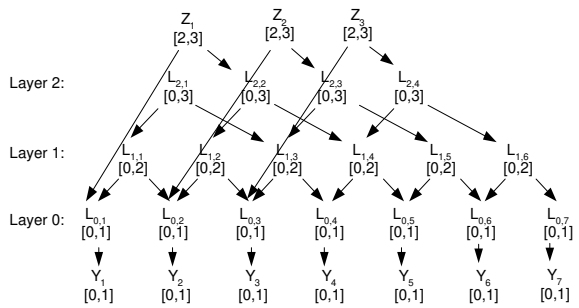

Fig. 3. Dependencies between partial sum variables L and Y and their initial domains for the SEQUENCE$(2, 3, 5, [Y_1, \ldots, Y_7])$ constraint.

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 arity $O(\log k)$. The constraint propagation cost equals the number of invocations of the filtering algorithm for this constraint times the cost of one invocation. The number of invocations is bounded by the number of values in the domains of the variables. Hence, the propagation cost of a ternary constraint is $O(u)$. For all ternary constraints we have a cost of $O(u)O(n \log k) = O(nu \log k)$. Also, we have $O(n)$ variables Z that are subject to $O(n)$ linear constraints. We split linear constraints of the form $a = b_1 + b_2 +, \ldots, +b_{p-2} + b_{p-1} + b_p$ into ternary constraints as follows: $a = b_1 + (b_2 + (, \ldots, (b_{p-2} + (b_{p-1} + b_p))))$. Each parenthesised expression creates an additional variable. Instead of having $O(n)$ linear constraints of arity $O(\log k)$, we have $O(n \log k)$ ternary constraints. The cardinality of each variable domain in these constraints is $O(u)$. Consequently, the propagation cost for the original linear constraints is $O(u)O(n \log k) = O(nu \log k)$. Therefore we can enforce bounds consistency on this encoding in $O(nu \log k) + O(nu \log k) = O(nu \log k)$ time down a branch of a search tree. However, this may not achieve domain consistency on the SEQUENCE constraint.

There are a number of redundant constraints we can add to improve propagation. In fact, we can add any permutation of partial sums that add up to 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. 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. □

We summarise relations among the decompositions in Figure 4.

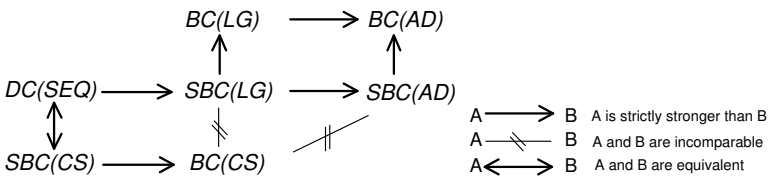

Fig. 4. Relations among decompositions of the SEQUENCE constraint.

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, $j \geq 1$, is SEQUENCE$(l_j, u_j, k_j, [X_1, \ldots, X_n], v_j)$. We suppose that the values being counted are disjoint. We channel into a new sequence of variables Y_i where $Y_i = j$ if $X_i \in v_j$ else $Y_i = 0$. 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. The automaton for the Multiple SEQUENCE with 2 SEQUENCEs is presented in Figure 5.

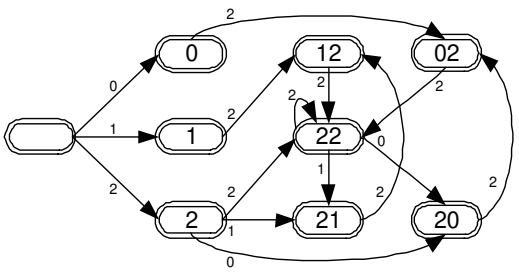

Fig. 5. The automaton for the Multiple SEQUENCE constraint with two SEQUENCEs: SEQUENCE$(0, 1, 2, [X_1, \ldots, X_n], \{1\})$ and SEQUENCE$(2, 3, 3, [X_1, \ldots, X_n], \{2\})$ with $D(X_i) = \{0, 1, 2\}$, for $i \in \{1, \ldots, n\}$.

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

6 Experimental Results

To compare performance of 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 and car sequencing benchmarks to test more realistic situations. Experiments were run with ILOG Solver 6.1 on an Intel Pentium 4 CPU 3.20Ghz, 1GB RAM.

6.1 Random instance

For each possible combination of $n \in \{50, 200, 500\}$, $k \in \{7, 15, 25, 50\}$, $\Delta = u - l \in \{1, 5\}$, we generated twenty instances with random lower bounds in the interval $[0, k - \Delta)$. We used a random value and variable ordering and a time-out of 100 sec. Results are given in Tables 1–2.

Instances can be partitioned into two groups. In the first group, $n > 50$ and $\Delta < 3$. On these instances, assignment of one variable has a strong impact on other variables. In

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

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

the extreme case when $\Delta = 0$ instantiation of one variable assigns on average another n/k variables. So, we expect DC propagators to significantly shrink variable domains and reduce the search tree. As can be seen from Table 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 *CD* algorithm is an order of magnitude slower compared to the *HPRS* propagator but in the current implementation we use incremental all-pairs shortest path algorithms, rather than the single-source shortest path algorithms of [10]. The *PS* algorithm is much slower compared to other DC algorithms and its relative performance decays for larger k.

In the second group, $n \leq 50$ or $\Delta \geq 3$. On these instances, assignment of a variable does not have a 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 that do not achieve DC. When $k < 25$ *AD* is best. When k gets larger, *LG* solves more instances and is faster.

6.2 Nurse Rostering Problems

Instances come from `www.projectmanagement.ugent.be/nsp.php`. For each day in the scheduling period, a nurse is assigned to a day, evening, or night shift or takes a day off. The original benchmarks specify minimal required staff allocation for each shift and individual preferences for each nurse. We ignore these preference and replace them with a set of constraints that model common workload restrictions for all nurses. 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. Each model was run on 50 instances. The scheduling period is 14 days. 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

Table 3. Models of the nurse rostering problem using the SEQUENCE constraint. Number of instances solved in 100 sec / average time to solve.

SEQUENCE	*PS*	*LO*	*HPRS*	*CD*	*AD*	GSC	*LG*	*CS*
$(1, 3, 3, \{O\})$	**43** / 0.47	**43** / 0.64	**43** / 0.54	**43** / 0.63	**43** / **0.45**	**43** / 0.60	**43** / **0.45**	39 / 2.49
$(3, 5, 5, \{O\})$	**44** / 2.43	**44** / 2.11	**44** / 2.28	**44** / 2.61	**44** / **1.84**	**44** / 2.84	**44** / 1.87	40 / 3.53
$(2, 2, 5, \{O\})$	39 / 5.41	39 / **4.76**	**40** / 7.41	38 / 4.97	36 / 7.50	35 / 7.73	36 / 8.56	36 / 5.36
$(2, 2, 7, \{O\})$	**23** / 9.09	**23** / 7.92	**23** / 5.64	**23** / 7.50	22 / 11.16	22 / 18.21	22 / 11.66	**23** / **4.53**
$(2, 3, 5, \{O\})$	26 / 4.65	26 / 4.92	**27** / 6.77	26 / **3.91**	**27** / 5.47	26 / 4.27	**27** / 5.81	26 / 5.77
$(2, 5, 7, \{O\})$	22 / 3.45	**23** / 6.90	22 / 2.22	22 / 2.43	**23** / 6.06	22 / 3.28	22 / **2.10**	22 / 2.28
$(1, 3, 4, \{O\})$	**27** / 7.02	26 / 5.25	**27** / 6.75	26 / **4.35**	**27** / 5.80	26 / 4.69	**27** / 6.05	25 / 6.18
TOTALS								
solved/total	224 /350	224 /350	**226** /350	222 /350	222 /350	218 /350	221 /350	211 /350
avg time for solved	4.169	4.068	4.263	**3.475**	4.776	5.170	4.676	4.218
avg bt for solved	11045	9905	13804	**8017**	20063	12939	18715	17747

Table 4. The Car Sequencing Problem. Number of instances solved in 100 sec / average time to solve.

	The basic model from the Ilog distribution +							
	PS	*LO*	*HPRS*	*CD*	*AD*		*LG*	*CS*
CarSequencing	**36** / 8.21	**36** / 8.26	35 / 7.59	34 / 7.85	**36** / 8.15	**36** / 7.95	**36** / 8.10	35 / **5.91**
TOTALS								
solved/total	**36** /78	**36** /78	35 /78	34 /78	**36** /78	**36** /78	**36** /78	35 /78
avg time for solved	8.21	8.26	7.59	7.85	8.15	7.95	8.10	**5.91**
avg bt for solved	518	518	265	**211**	518	518	518	265

ordering, we branched on the smallest domain. Table 3 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.

6.3 Car Sequencing Problems

In this series of experiments we used car sequencing problems benchmarks. All instances are taken from CSPLib (the first set of benchmarks). We used the car sequencing model from the Ilog distribution as the basic model and added each encoding as a redundant constraint to this model. The time limit for all instances was 100 sec. All SEQUENCE constraints in these benchmarks are monotone. Hence, GSC enforces DC on the SEQUENCE part of it, and an introduction of redundant constraints does not give extra pruning. As Table 4 shows, the basic model gives the best performance.

6.4 Multiple SEQUENCE constraints

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 $n \in \{50, 100\}$, $k \in \{5, 7\}$, $\Delta = 1$, we generated twenty random instances with four SEQUENCE constraints. All variables had domains of size 5. An instance was obtained by selecting random lower bounds in the

Table 5. Randomly generated instances with 4 SEQUENCE constraints and $\Delta = 1$. Number of instances solved in 100 sec / average time to solve.

n	k	*MR*	*PS*	*LO*	*HPRS*	*CD*	*AD*	GSC	*LG*	*CS*
50	5	**20** / 0.05	6 / 12.58	6 / 17.03	5 / 0.81	5 / 4.76	6 / 13.75	5 / 10.59	7 / 15.05	0 / **0**
	7	**20** / 0.86	6 / 20.85	6 / 16.89	7 / 23.99	4 / **0.15**	6 / 14.02	5 / 15.90	8 / 26.81	2 / 5.80
100	5	**20** / 0.11	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**	0 / **0**
	7	**20** / 1.83	2 / 8.98	2 / 7.52	2 / 10.48	1 / **0**	1 / **0**	1 / **0**	1 / **0**	0 / **0**
TOTALS										
solved/total		**80** /80	14 /80	14 /80	14 /80	10 /80	13 /80	11 /80	16 /80	2 /80
avg time for solved		**0.71**	15.61	15.61	13.78	2.44	12.81	12.04	19.99	5.80
avg bt for solved		**0**	72078	72078	64579	2214	185747	51413	257831	106008

Table 6. Models of the nurse rostering problem using the SEQUENCE constraint. Number of instances solved in 100 sec / average time to solve.

	MR	*PS*	*LO*	*HPRS*	*CD*	*AD*	GSC	*LG*	*CS*
Model 1	**9** / 10.62	4 / 12.61	4 / 14.78	4 / 7.38	4 / 19.00	4 / 6.41	4 / 16.87	4 / **5.58**	4 / 5.64
Model 2	**8** / 1.40	4 / 0.08	4 / 0.11	4 / **0.04**	4 / 0.06	4 / **0.04**	4 / 0.08	4 / **0.04**	4 / **0.04**
TOTALS									
solved/total	**17** /100	8 /100	8 /100	8 /100	8 /100	8 /100	8 /100	8 /100	8 /100
avg time for solved	6.28	6.35	7.45	3.71	9.53	3.22	8.47	**2.81**	2.84
avg bt for solved	**3470**	30696	30696	30696	30696	30232	30232	30430	30749

interval $[0, k - \Delta]$. We excluded 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 are given in Table 5. The Multiple SEQUENCE propagator significantly outperforms other propagators in both the time to find a valid sequence and the number of solved instances. For bigger values of n, the Multiple SEQUENCE propagator is the only one able to solve all instances. However, due to its space complexity, to use this propagator, k and m need to be relatively small and $n < 100$.

In the second series of experiments we used nurse scheduling 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 1 or 2 night shifts in 7 consecutive days, 1 or 2 evening shifts, 1 to 5 day shifts and 2 to 5 days-off. In the second model, each nurse has to work 1 or 2 night shifts in 7 consecutive days, and has 1 or 2 days off in 5 days. In order to test the performance of the Multiple SEQUENCE constraint on large problems, we built a schedule over a 28 days period. The number of nurses was equal to the maximum number of nurses required for any day over the period multiplied by 1.5. The total number of variables in an instance is about 500. Table 6 shows the number of instances solved by each propagator. The Multiple SEQUENCE propagator again solved the most instances.

7 Conclusion

The SEQUENCE constraint is useful in modelling a range of rostering, scheduling and car sequencing problems. We proved that down a branch of a search tree domain con-

sistency 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 a generalization of SEQUENCE constraint – the Multiple SEQUENCE constraint. 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?

8 Acknowledgements

We would like to thank Willem-Jan van Hoeve for providing us with the implementation of the *HPRS* algorithm, as well as the reviewers for useful feedback.

References

1. Pesant, G.: A regular language membership constraint for finite sequences of variables. In Wallace, M., ed.: 10th Int. Conf. on Principles and Practice of Constraint Programming (CP'04). Volume 3258 of LNCS., Springer (2004) 482–495
2. Quimper, C.G., Walsh, T.: Global grammar constraints. [11] 751–755
3. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling **12** (1994) 97–123
4. Debruyne, R., Bessière, C.: Some practicable filtering techniques for the constraint satisfaction problem. In: 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97), Morgan Kaufmann (1997) 412–417
5. Bessière, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: The SLIDE-meta constraint. Technical report (2007)
6. van Hoeve, W.J., Pesant, G., Rousseau, L.M., Sabharwal, A.: Revisiting the Sequence constraint. [11] 620–634
7. R´egin, J.C., Puget, J.F.: A filtering algorithm for global sequencing constraints. In Smolka, G., ed.: 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP'97). Volume 1330 of LNCS., Springer (1997) 32–46
8. Beldiceanu, N., Carlsson, M.: Revisiting the cardinality operator and introducing the cardinality-path constraint family. In Codognet, P., ed.: 17th Int. Conf. on Logic Programming (ICLP'01). Volume 2237 of LNCS., Springer (2001) 59–73
9. R´egin, J.C.: Combination of Among and Cardinality constraints. In Bart´ak, R., Milano, M., eds.: 2nd Int. Conf. on Integr. of AI and OR Techniques in Constraint Progr. for Comb. Optimization Problems, (CPAIOR'05). Volume 3524 of LNCS., Springer (2005) 288–303
10. Cotton, S., Maler, O.: Fast and flexible difference constraint propagation for DPLL(T). In Biere, A., Gomes, C.P., eds.: 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06). Volume 4121 of LNCS., Springer (2006) 170–183
11. Benhamou, F., ed.: Proc. of 12th Int. Conf. on Principles and Practice of Constraint Programming (CP'06). Volume 4204 of LNCS., Springer (2006)