The Range and Roots Constraints: Specifying Counting and Occurrence Problems*

Christian Bessiere LIRMM-CNRS Montpelier, France bessiere@lirmm.fr

Emmanuel Hebrard NICTA and UNSW Sydney, Australia e.hebrard@nicta.com.au

Brahim Hnich 4C, UCC Ireland brahim@4c.ucc.ie

Zeynep Kiziltan DEIS Univ. di Bologna, Italy zkiziltan@deis.unibo.it

Toby Walsh NICTA and UNSW Sydney, Australia tw@cse.unsw.edu.au

Abstract

We propose a simple declarative language for specifying a wide range of counting and occurrence constraints. This specification language is executable since it immediately provides a polynomial propagation algorithm. To illustrate the capabilities of this language, we specify a dozen global constraints taken from the literature. We observe one of three outcomes: we achieve generalized arc-consistency; we do not achieve generalized arc-consistency, but achieving generalized arc-consistency is NP-hard; we do not achieve generalized arc-consistency, but specialized propagation algorithms can do so in polynomial time. Experiments demonstrate that this specification language is both efficient and effective in practice.

1 Introduction

Global constraints are central to the success of constraint programming. Global constraints allow users to specify patterns that occur in many problems, and to exploit efficient and effective propagation algorithms for pruning the search space. Two common types of global constraints are counting and occurrence constraints. Occurrence constraints place restrictions on the occurrences of particular values. For instance, we may wish to ensure that no value used by one set of variables occurs in a second set. Counting constraints, on the other hand, restrict the number of values or variables meeting some condition. For example, we may want to limit the number of distinct values assigned to a set of variables. Many different counting and occurrences constraints have been proposed to help model a wide range of problems, especially those involving resources (see, for example, [Régin, 1994; Beldiceanu and Contejean, 1994; Régin, 1996; Beldiceanu, 2001; Beldiceanu *et al.*, 2004b]).

We will show that many such constraints can be specified using a simple declarative language. This specification language is executable. It decomposes these constraints into some simple primitives for which there are polynomial propagation algorithms. In some cases, we show that this decomposition does not hinder propagation. In other cases, achieving generalized arc-consistency is NP-hard and decomposition is one method to obtain a polynomial propagation algorithm. In the remaining cases, we show that the decomposition hinders propagation, and there exists some specialized and polynomial propagation algorithm. As not all global constraints are implemented and integrated into every solver, our language may still be attractive in this case. We provide a generic means for propagating counting and occurrence constraints in the absence of a specialized algorithm. Our experiments demonstrate that this is an efficient and effective way to reason about counting and occurrence constraints in practice.

2 Formal Background

A constraint satisfaction problem consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for 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). We write $\mathcal{D}(X)$ for the domain of a variable X. A solution is an assignment of values to the variables satisfying the constraints. A variable is ground when it is assigned a value. We consider both integer and set variables. A constraint can be defined on only integer, or only set variables, or on both. A set variable S can be represented by its lower bound $lb(S)$ which contains the definite elements and an upper bound $ub(S)$ which contains the definite and potential elements. Each set variable is equivalent to a sequence of 0/1 variables representing the characteristic function. A variable in such a sequence is 1 iff the corresponding element is in the set.

Given a constraint C, a *bound support* on C is a tuple that assigns to each integer variable a value between its minimum and maximum, and to each set variable a set between its lower and upper bounds which satisfies C. An *hybrid support* on C is a tuple that assigns to each integer variable a value in its domain, and to each set variable a set between its lower and upper bounds which satisfies C. If C involves only integer variables, a hybrid support is a *support*. A constraint C is *bound consistent* (*BC*) iff for each integer variable X_i, its minimum and maximum values belong to a bound support, and for each set variable S_j, the values in $ub(S_j)$ belong to S_j in at least one bound support and the values in $lb(S_j)$ belong to S_j in all bound supports. A constraint C is *hybrid consistent* (*HC*) iff for each integer variable X_i, every value in $D(X_i)$ belongs to an hybrid support, and for each set vari-

*Brahim Hnich is currently supported by Science Foundation Ireland under Grant No. 00/PI.1/C075.

able S_j, the values in $ub(S_j)$ belong to S_j in at least one hybrid support, and the values in $lb(S_j)$ belong to S_j in all hybrid supports. A constraint C involving only integer variables is *generalized arc consistent* (*GAC*) iff for each variable X_i, every value in $D(X_i)$ belongs to a support.

Enforcing hybrid consistency on a constraint C is equivalent to enforcing GAC if the set variables are represented by their characteristic function. If all variables in C are integer variables, hybrid consistency reduces to generalized arc-consistency, and if all variables in C are set variables, hybrid consistency reduces to bound consistency.

We will compare local consistency properties applied to (sets of) logically equivalent constraints, c_1 and c_2. As in [Debruyne and Bessière, 1997], 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; Φ on c_1 is incomparable to Ψ on c_2 iff Φ on c_1 is not as strong as Ψ on c_2 and vice versa.

A total function $\mathcal{F}$ from a set $\mathcal{S}$ into a set $\mathcal{T}$ is denoted by $\mathcal{F}: \mathcal{S} \longrightarrow \mathcal{T}$ where $\mathcal{S}$ is the domain of $\mathcal{F}$ and $\mathcal{T}$ is the range of $\mathcal{F}$. The set of all elements in the domain of $\mathcal{F}$ that have the same image $j \in \mathcal{T}$ is $\mathcal{F}^{-1}(j) = \{i : \mathcal{F}(i) = j\}$. The image of a set $S \subseteq \mathcal{S}$ under $\mathcal{F}$ is $\mathcal{F}(S) = \bigcup_{i \in S} \mathcal{F}(i)$, whilst the domain of a set $T \subseteq \mathcal{T}$ under $\mathcal{F}$ is $\mathcal{F}^{-1}(T) = \bigcup_{j \in T} \mathcal{F}^{-1}(j)$. Throughout, we will view a set of variables, X_1 to X_n as a function $\mathcal{X} : \{1, .., n\} \longrightarrow \bigcup_{i=1}^{i=n} \mathcal{D}(X_i)$. That is, $\mathcal{X}(i)$ is the value of X_i.

3 Specification Language

We propose a simple declarative language in which we can specify many different counting and occurrence constraints. This specification language is executable. It permits us to decompose each global constraint into more primitive constraints, each of which has an associated polynomial propagation algorithm. In some cases, this decomposition does not hinder propagation. In other cases, enforcing local consistency on the global constraint is intractable, and decomposition is one method to obtain a polynomial propagation algorithm. The specification language consists of simple non-global constraints over integer variables (like $X \leq m$), simple non-global constraints over set variables (like $S_1 \subseteq S_2$ or $|S| = k$) and two special global constraints acting on sequences of variables: ROOTS and RANGE.

Given a function $\mathcal{X}$ representing a set of variables, X_1 to X_n, the RANGE constraint holds iff a set variable, T is the range of this function restricted to the indices belonging to a second set variable, S.

$$\text{RANGE}([X_1, .., X_n], S, T) \text{ iff } \mathcal{X}(S) = T$$

The ROOTS constraint holds iff S is the set of indices which map to an element in T.

$$\text{ROOTS}([X_1, .., X_n], S, T) \text{ iff } S = \mathcal{X}^{-1}(T)$$

ROOTS and RANGE are not exact inverses. A RANGE constraint can hold, but the corresponding ROOTS constraint may not, and vice versa. For instance, $\text{RANGE}([1, 1], \{1\}, \{1\})$ holds but not $\text{ROOTS}([1, 1], \{1\}, \{1\})$ since $\mathcal{X}^{-1}(1) = \{1, 2\}$, and $\text{ROOTS}([1, 1, 1], \{1, 2, 3\}, \{1, 2\})$ holds but not $\text{RANGE}([1, 1, 1], \{1, 2, 3\}, \{1, 2\})$ as no X_i is assigned to 2.

In an associated report, we discuss how to propagate these two global constraints. Enforcing HC on the RANGE constraint is polynomial and give an $O(nd + n \cdot |lb(T)|^{3/2})$ algorithm where d is the maximum domain size of the X_i. Enforcing BC on the ROOTS constraint is $O(nd)$. Unfortunately, enforcing HC on the ROOTS constraint is NP-hard. Nevertheless, all the cases needed here are polynomial. For example, if all X_i are ground, or T is ground then enforcing HC on $\text{ROOTS}([X_1, .., X_n], S, T)$ is $O(nd)$.

4 Range Constraints

In an associated report, we present a catalog containing over 70 global constraints from [Beldiceanu, 2000] specified with this simple language. We have room here to present just a few of the more important constraints. In this and the subsequent three sections, we list some global constraints which can be specified using RANGE constraints, using ROOTS constraints, and using both RANGE and ROOTS constraints.

4.1 All Different

The ALLDIFFERENT constraint forces a sequence of variables to take different values from each other. Such a constraint is useful in a wide range of problems (e.g. allocation of activities to different slots in a timetabling problem). It can be propagated efficiently [Régin, 1994]. It can also be decomposed with a single RANGE constraint:

$$\text{ALLDIFFERENT}([X_1, .., X_n]) \text{ iff}$$
$$\text{RANGE}([X_1, .., X_n], \{1, .., n\}, T) \ \wedge \ |T| = n$$

A special but nevertheless important case of this constraint is the PERMUTATION constraint. This is an ALLDIFFERENT constraint where we additionally know $\mathcal{R}$, the set of values to be taken. That is, the sequence of variables is a permutation of the values in $\mathcal{R}$ where $|\mathcal{R}| = n$. This also can be decomposed using a single RANGE constraint:

$$\text{PERMUTATION}([X_1, .., X_n], \mathcal{R}) \text{ iff}$$
$$\text{RANGE}([X_1, .., X_n], \{1, .., n\}, \mathcal{R})$$

Such a decomposition of the PERMUTATION constraint obviously does not hinder propagation. However, decomposition of ALLDIFFERENT into a RANGE constraint does. In addition, HC on the RANGE decomposition is incomparable to AC on the decomposition of ALLDIFFERENT which uses a clique of binary inequality constraints. Thus, we may be able to obtain more pruning by using both decompositions.

Theorem 1 *GAC on* PERMUTATION *is equivalent to HC on the decomposition with* RANGE. *GAC on* ALLDIFFERENT *is stronger than HC on the decomposition with* RANGE. *AC on the decomposition of* ALLDIFFERENT *into binary inequalities is incomparable to HC on the decomposition with* RANGE.

Proof: Consider X_1, $X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3, 4\}$, and $\{1, 2\} \subseteq T \subseteq \{1, 2, 3, 4\}$. Then

RANGE($[X_1, X_2, X_3], \{1, 2, 3\}, T$) and $|T| = 3$ are both HC, but ALLDIFFERENT($[X_1, X_2, X_3]$) is not GAC. Consider X_1, $X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3\}$, and $T = \{1, 2, 3\}$. Then $X_1 \neq X_2$, $X_1 \neq X_3$ and $X_2 \neq X_3$ are AC but RANGE($[X_1, X_2, X_3], \{1, 2, 3\}, T$) is not HC. Consider X_1, $X_2 \in \{1, 2, 3, 4\}$, $X_3 \in \{2\}$, and $\{2\} \subseteq T \subseteq \{1, 2, 3, 4\}$. Then RANGE($[X_1, X_2, X_3], \{1, 2, 3\}, T$) and $|T| = 3$ are HC. But $X_1 \neq X_3$ and $X_2 \neq X_3$ are not AC. The other result holds immediately. □

4.2 Number of Values

The NVALUE constraint is useful in a wide range of problems involving resources since it counts the number of distinct values used by a sequence of variables [Pachet and Roy, 1999]. NVALUE($[X_1, .., X_n], N$) holds iff $N = |\{X_i \mid 1 \leq i \leq n\}|$. The ALLDIFFERENT constraint is a special case of the NVALUE constraint in which $N = n$. Unfortunately, it is NP-hard in general to enforce GAC on a NVALUE constraint [Bessiere *et al.*, 2004]. However, there is an $O(n \log(n))$ algorithm to enforce a level of consistency similar to BC [Beldiceanu, 2001]. An alternative and even simpler way to implement this constraint is with a RANGE constraint:

$$\text{NVALUE}([X_1, .., X_n], N) \text{ iff}$$
$$\text{RANGE}([X_1, .., X_n], \{1, .., n\}, T) \wedge |T| = N$$

HC on this decomposition is incomparable to BC on the NVALUE constraint.

Theorem 2 *BC on* NVALUE *is incomparable to HC on the decomposition.*

Proof: Consider $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3, 4\}$, $N \in \{3\}$ and $\{\} \subseteq T \subseteq \{1, 2, 3, 4\}$. Then RANGE($[X_1, X_2, X_3], \{1, 2, 3\}, T$) and $|T| = N$ are both HC. However, enforcing BC on NVALUE($[X_1, X_2, X_3], N$) prunes 1 and 2 from X_3.

Consider $X_1, X_2, X_3 \in \{1, 3\}$ and $N \in \{3\}$. Then NVALUE($[X_1, X_2, X_3], N$) is BC. However, enforcing HC on RANGE($[X_1, X_2, X_3], \{1, 2, 3\}, T$) makes $\{\} \subseteq T \subseteq \{1, 3\}$ which will cause $|T| = 3$ to fail. □

4.3 Uses

In [Beldiceanu *et al.*, 2004b], propagation algorithms achieving GAC and BC are proposed for the USEDBY constraint. USEDBY($[X_1, .., X_n], [Y_1, .., Y_m]$) holds iff the *multiset* of values assigned to $Y_1, .., Y_m$ is a subset of the *multiset* of values assigned to $X_1, .., X_n$. We now introduce a variant of the USEDBY constraint called the USES constraint. USES($[X_1, .., X_n], [Y_1, .., Y_m]$) holds iff the *set* of values assigned to $Y_1, .., Y_m$ is a subset of the *set* of values assigned to $X_1, .., X_n$. That is, USEDBY takes into account the number of times a value is used while USES does not. Unlike the USEDBY constraint, enforcing GAC on USES is NP-hard.

Theorem 3 *Enforcing GAC on* USES *is NP-hard.*

Proof: We reduce 3-SAT to the problem of deciding if a USES constraint has a solution. Finding support is therefore NP-hard. Consider a formula φ with n Boolean variables and m clauses. For each Boolean variable i, we introduce a variable $X_i \in \{i, -i\}$. For each clause $c_j = x \vee \neg y \vee z$, we introduce $Y_j \in \{x, -y, z\}$. Then φ has a model iff the USES constraint has a satisfying assignment, and i is true iff $X_i = i$. □

One way to propagate a USES constraint is to decompose it using RANGE constraints:

$$\text{USES}([X_1, .., X_n], [Y_1, .., Y_m]) \text{ iff}$$
$$\text{RANGE}([X_1, .., X_n], \{1, .., n\}, T) \wedge$$
$$\text{RANGE}([Y_1, .., Y_m], \{1, .., m\}, T') \wedge T' \subseteq T$$

Enforcing HC on this decomposition is polynomial. Not surprisingly, this hinders propagation (otherwise we would have a polynomial algorithm for a NP-hard problem).

Theorem 4 *GAC on* USES *is stronger than HC on the decomposition.*

Proof: Consider $X_1 \in \{1, 2, 3, 4\}$, $X_2 \in \{1, 2, 3, 5\}$, $X_3, X_4 \in \{4, 5, 6\}$, $Y_1 \in \{1, 2\}$, $Y_2 \in \{1, 3\}$, and $Y_3 \in \{2, 3\}$. The decomposition is HC while GAC on USES prunes 4 from the domain of X_1 and 5 from the domain of X_2. □

Thus, decomposition is a simple method to obtain a polynomial propagation algorithm.

5 Roots Constraints

RANGE constraints are often useful to specify constraints on the values used by a sequence of variables. ROOTS constraint, on the other hand, are useful to specify constraints on the variables taking particular values.

5.1 Global Cardinality

The global cardinality constraint introduced in [Régin, 1996] constrains the number of times values are used. We consider a generalization in which the number of occurrences of a value may itself be an integer variable. That is, GCC($[X_1, .., X_n], [d_1, .., d_m], [O_1, .., O_m]$) holds iff $|\{i \mid X_i = d_j\}| = O_j$ for all j. Such a GCC constraint can be decomposed into a set of ROOTS constraints:

$$\text{GCC}([X_1, .., X_n], [d_1, .., d_m], [O_1, .., O_m]) \text{ iff}$$
$$\forall i \,.\, \text{ROOTS}([X_1, .., X_n], S_i, \{d_i\}) \wedge |S_i| = O_i$$

Enforcing HC on these ROOTS constraints is polynomial since the sets $\{d_i\}$ are ground. Enforcing GAC on a generalized GCC constraint is NP-hard, but we can enforce GAC on the X_i and BC on the O_j in polynomial time using a specialized algorithm [Quimper *et al.*, 2004]. This is more than is achieved by the decomposition.

Theorem 5 *GAC on the X_i and BC on the O_j of a* GCC *constraint is stronger than HC on the decomposition using* ROOTS *constraints.*

Proof: Sets being represented by their bounds, HC on the decomposition cannot prune more on the O_j than BC does on the GCC. To show strictness, consider $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3\}$, $d_i = i$ and $O_1, O_2, O_3 \in \{0, 1\}$. The decomposition is HC (with $\{\} \subseteq S_1, S_2 \subseteq \{1, 2, 3\}$ and $\{\} \subseteq S_3 \subseteq \{3\}$). However, enforcing GAC on the X_i and BC on the O_j of the GCC constraint will prune 1 and 2 from X_3 and 0 from O_1, O_2 and O_3. □

This example illustrates that, whilst many global constraints can be expressed in terms of ROOTS and RANGE, there are some global constraints like GCC for which it is worth developing specialized propagation algorithms. Nevertheless, ROOTS and RANGE provide a means of propagation for such constraints in the absence of specialised algorithms.

5.2 Among

The AMONG constraint was introduced in CHIP to help model resource allocation problems like car sequencing [Beldiceanu and Contejean, 1994]. It counts the number of variables using values from a given set. AMONG$([X_1,..,X_n], [d_1,..,d_m], N)$ holds iff $N = |\{i \mid X_i \in \{d_1,..,d_m\}\}|$.

One decomposition using the global cardinality constraint GCC introduced in [Régin, 1996] is as follows:

$$\text{AMONG}([X_1,..,X_n],[d_1,..,d_m],N) \text{ iff}$$
$$\text{GCC}([X_1,..,X_n],[d_1,..,d_m],[O_1,..,O_m]) \wedge \sum O_i = N$$

Unfortunately, this decomposition hinders propagation.

Theorem 6 *GAC on* AMONG *is stronger than GAC on the decomposition using* GCC.

Proof: Consider $X_1, X_2 \in \{1,2\}$, $X_3 \in \{1,2,3\}$, $d_1 = 1$, $d_2 = 2$, and $N \in \{1,2\}$. The decomposition is GAC, but enforcing GAC on AMONG$([X_1,X_2,X_3],[d_1,d_2],N)$ prunes 1 and 2 from X_3 as well as 1 from N. □

An alternative way to propagate the AMONG constraint is to decompose it using a ROOTS constraint:

$$\text{AMONG}([X_1,..,X_n],[d_1,..,d_m],N) \text{ iff}$$
$$\text{ROOTS}([X_1,..,X_n],S,\{d_1,..,d_m\}) \wedge |S| = N$$

It is polynomial to enforce HC on this case of the ROOTS constraint since the target set is ground. This decomposition also does not hinder propagation. It is therefore a potentially attractive method to implement the AMONG constraint.

Theorem 7 *GAC on* AMONG *is equivalent to HC on the decomposition using* ROOTS.

Proof: (Sketch) Suppose the decomposition into ROOTS$([X_1,..,X_n],S,\{d_1,..,d_m\})$ and $|S| = N$ is HC. The variables X_i divide into three categories: those whose domain only contains elements from $\{d_1,..,d_m\}$ (at most $\min(N)$ such vars); those whose domain do not contain any such elements (at most $n - \max(N)$ such vars); those whose domain contains both elements from this set and from outside. Consider any value for a variable X_i in the first such category. To construct support for this value, we assign the remaining variables in the first category with values from $\{d_1,..,d_m\}$. If the total number of assigned values is less than $\min(N)$, we assign a sufficient number of variables from the second category with values from $\{d_1,..,d_m\}$ to bring up the count to $\min(N)$. We then assign all the remaining unassigned X_j with values outside $\{d_1,..,d_m\}$. Finally, we assign $\min(N)$ to N. Support can be constructed for variables in the other two categories in a similar way, as well as for any value of N between $\min(N)$ and $\max(N)$. □

The ATMOST and ATLEAST constraints are closely related. The ATMOST constraint puts an upper bound on the number of variables using a particular value, whilst the ATLEAST puts a lower bound. Both can be decomposed using a ROOTS constraint without hindering propagation.

6 Range and Roots Constraints

Some global constraints need both ROOTS and RANGE constraints in their specifications.

6.1 Assign and NValues

In bin packing and knapsack problems, we may wish to assign both a value and a bin to each item, and place constraints on the values appearing in each bin. For instance, in the steel mill slab design problem (prob038 in CSPLib), we assign colors and slabs to orders so that there are a limited number of colors on each slab. ASSIGN&NVALUES$([X_1,..,X_n],[Y_1,..,Y_n],N)$ holds iff $|\{Y_i \mid X_i = j\}| \leq N$ for each j [Beldiceanu, 2000]. It can be decomposed into a set of ROOTS and RANGE constraints:

$$\text{ASSIGN\&NVALUES}([X_1,..,X_n],[Y_1,..,Y_n],N) \text{ iff}$$
$$\forall j \,.\, \text{ROOTS}([X_1,..,X_n],S_j,\{j\}) \wedge$$
$$\text{RANGE}([Y_1,..,Y_n],S_j,T_j) \wedge |T_j| \leq N$$

Decomposition hinders propagation as it considers the values separately. However, since this constraint generalizes NVALUE, enforcing GAC is NP-hard. Decomposition is thus one method to obtain a polynomial propagation algorithm.

6.2 Common

A generalization of the AMONG and ALLDIFFERENT constraints introduced in [Beldiceanu, 2000] is the COMMON constraint. COMMON$(N,M,[X_1,..,X_n],[Y_1,..,Y_m])$ ensures $N = |\{i \mid \exists j, X_i = Y_j\}|$ and $M = |\{j \mid \exists i, X_i = Y_j\}|$. That is, N variables in X_i take values in common with Y_j and M variables in Y_j takes values in common with X_i. We prove that we cannot expect to enforce GAC on such a constraint as it is NP-hard to do so in general.

Theorem 8 *Enforcing GAC on* COMMON *is NP-hard.*

Proof: We again use a transformation from 3-SAT. Consider a formula φ with n Boolean variables and m clauses. For each Boolean variable i, we introduce a variable $X_i \in \{i, -i\}$. For each clause $c_j = x \vee \neg y \vee z$, we introduce $Y_j \in \{x, -y, z\}$. We let $N \in \{0,..,n\}$ and $M = m$. φ has a model iff the COMMON constraint has a solution in which the X_i take the literals true in this model. □

One way to propagate a COMMON constraint is to decompose it into RANGE and ROOTS constraints:

$$\text{COMMON}(N,M,[X_1,..,X_n],[Y_1,..,Y_m]) \text{ iff}$$
$$\text{RANGE}([Y_1,..,Y_m],\{1,..,m\},T) \wedge$$
$$\text{ROOTS}([X_1,..,X_n],S,T) \wedge |S| = N \wedge$$
$$\text{RANGE}([X_1,..,X_n],\{1,..,n\},V) \wedge$$
$$\text{ROOTS}([Y_1,..,Y_m],U,V) \wedge |U| = M$$

Enforcing HC on this decomposition is polynomial. Decomposition thus offers a simple and promising method to propagate a COMMON constraint. Not surprisingly, the decomposition hinders propagation.

Theorem 9 *GAC on* COMMON *is stronger than HC on the decomposition.*

Proof: Consider $N = M = 0$, $X_1, Y_1 \in \{1, 2\}$, $X_2, Y_2 \in \{1, 3\}$, $Y_3 \in \{2, 3\}$. Hybrid consistency on the decomposition enforces $\{\} \subseteq T, V \subseteq \{1, 2, 3\}$, and $S = U = \{\}$ but no pruning on the X_i and Y_j. However, enforcing GAC on COMMON$(N, M, [X_1, X_2], [Y_1, Y_2, Y_3])$ prunes 2 from X_1, 3 from X_2 and 1 from both Y_1 and Y_2. $\Box$

6.3 Symmetric All Different

In certain domains, we may need to find symmetric solutions. For example, in sports scheduling problems, if one team is assigned to play another then the second team should also be assigned to play the first. SYMALLDIFF$([X_1, .., X_n])$ ensures $X_i = j$ iff $X_j = i$ [Régin, 1999]. It can be decomposed into a set of ROOTS and RANGE constraints:

$$\text{SYMALLDIFF}([X_1, .., X_n]) \text{ iff}$$
$$\text{RANGE}([X_1, .., X_n], \{1, .., n\}, \{1, .., n\}) \wedge$$
$$\forall i \, . \, \text{ROOTS}([X_1, .., X_n], S_i, \{i\}) \wedge X_i \in S_i \wedge |S_i| = 1$$

It is polynomial to enforce HC on these cases of the ROOTS constraint. However, as with the ALLDIFFERENT constraint, it is more effective to use a specialized propagation algorithm like that in [Régin, 1999].

Theorem 10 *GAC on* SYMALLDIFF *is stronger than HC on the decomposition.*

Proof: Consider $X_1 \in \{2, 3\}$, $X_2 \in \{1, 3\}$, $X_3 \in \{1, 2\}$, $\{\} \subseteq S_1 \subseteq \{2, 3\}$, $\{\} \subseteq S_2 \subseteq \{1, 3\}$, and $\{\} \subseteq S_3 \subseteq \{1, 2\}$. Then the decomposition is HC. However, enforcing GAC on SYMALLDIFF$([X_1, X_2, X_3])$ will detect unsatisfiability. $\Box$

To our knowledge, this constraint has not been integrated into any constraint solver. Thus, this decomposition provides a means of propagation for the SYMALLDIFF constraint.

7 Beyond Counting and Occurrence

The RANGE and ROOTS constraints are useful for specifying a wide range of counting and occurence constraints. Nevertheless, their expressive power permits their use to specify many other constraints.

The ELEMENT constraint introduced in [Van Hentenryck and Carillon, 1988] indexes into an array with a variable. More precisely, ELEMENT$(I, [X_1, .., X_n], J)$ holds iff $X_I = J$. We can use such a constraint to look up the price of a component included in a configuration problem. The ELEMENT constraint can be decomposed into a RANGE constraint without hindering propagation:

$$\text{ELEMENT}(I, [X_1, .., X_n], J) \text{ iff } |S| = |T| = 1 \wedge$$
$$I \in S \wedge J \in T \wedge \text{RANGE}([X_1, .., X_n], S, T)$$

We may wish to channel between a variable and the sequence of 0/1 variables representing the possible values taken by the variable. The DOMAIN$(X, [X_1, .., X_m])$ constraint introduced in [Refalo, 2000] ensures $X = i$ iff $X_i = 1$. This can be decomposed into a ROOTS constraint:

$$\text{DOMAIN}(X, [X_1, .., X_m]) \text{ iff}$$
$$\text{ROOTS}([X_1, .., X_m], S, \{1\}) \wedge |S| = 1 \wedge X \in S$$

Enforcing HC on this decomposition is polynomial ($\{1\}$ is ground) and it is equivalent to enforcing GAC on the DOMAIN constraint.

The CONTIGUITY constraint ensures that, in a sequence of 0/1 variables, those taking the value 1 appear contiguously. This is a discrete form of convexity. The constraint was introduced in [Maher, 2002] to model a hardware configuration problem. It can be decomposed into a ROOTS constraint:

$$\text{CONTIGUITY}([X_1, .., X_n]) \text{ iff}$$
$$\text{ROOTS}([X_1, .., X_n], S, \{1\}) \wedge$$
$$X = \max(S) \wedge Y = \min(S) \wedge |S| = X - Y + 1$$

Again it is polynomial to enforce HC on this case of the ROOTS constraint. Unfortunately, decomposition hinders propagation. Consider $X_1, X_3 \in \{0, 1\}$, $X_2, X_4 \in \{1\}$. HC on the decomposition will enforce $\{2, 4\} \subseteq S \subseteq \{1, 2, 3, 4\}$, $X \in \{4\}$, $Y \in \{1, 2\}$ and $|S|$ to be in $\{3, 4\}$ but no pruning will happen on the X_i. However, enforcing GAC on CONTIGUITY$([X_1, .., X_n])$ will prune 0 from X_3.

Whilst ROOTS and RANGE can specify concepts quite distant from counting and occurrences like convexity and discrete derivatives, it seems that we may need other algorithmic ideas to propagate them effectively.

8 Experiments

The last three sections demonstrate that many global constraints can be specified in terms of ROOTS and RANGE constraints. We also argued that these specifications are executable and in certain cases provide efficient and effective propagation algorithms. In this section, we support this argument by means of some experiments.

	2GCC+SUM			GCC+ROOTS			ROOTS		
Size	f	t	#s	f	t	#s	f	t	#s
10	6	0.02	9/10	6	0.01	9/10	335	0.41	10/10
15	6475	1.56	29/52	6468	1.38	29/52	23	0.10	52/52
20	2852	1.37	20/35	2852	0.97	20/35	1093	3.54	32/35
25	71	0.22	13/20	71	0.09	13/20	2776	13.62	19/20
30	12472	11.87	6/10	12472	6.80	7/10	4279	17.23	9/10
35	16	0.44	20/56	14	0.15	21/56	5107	36.29	49/56

Table 1: Mystery Shopper. Instance are run with a 5 minutes time limit. #fails (f) and cpu time (t) are averaged on the #instances solved (#s) by each method.

We used a model for the Mystery Shopper problem [Cheng *et al.*, 1998] due to Helmut Simonis that appears in CSPLib. This model contains a GCC constraint and a number of AMONG constraints. ILOG's Solver contains the GCC but not the AMONG constraint. We implement AMONG either with a GCC and a SUM constraint or directly with ROOTS. This gives us two models (2GCC+SUM and GCC+ROOTS, resp.). We also consider the model (ROOTS) which is a variation of GCC+ROOTS where we decompose the GCC constraint using ROOTS. In order to generate a range of instances, we used the following protocol. The duration in weeks as well as the number of varieties of shoppers, visits per salelady and number of areas are all set to 4 (as in the original problem). The number of saleladies takes its value in $\{10, 15, 20, 25, 30, 35\}$, and the number of shoppers is the next multiple of 4 that is not consecutive. All the possible ways of partitioning the saleladies into 4 areas (that do not trivially violate the constraints) are solved. As expected, the decomposition of the AMONG constraint using ROOTS is more efficient than the decomposition with a GCC and a SUM constraint (see Table 1). Surprisingly, however, the ROOTS model is the most efficient strategy (see the '#s' column). This is because we were able to branch on the 'extra' set variables, which proved to be the most effective method. Finally, the inference using GCC (2GCC+SUM) is faster (better ratio t/f) than using its decomposition (ROOTS). This suggests that it will be worth investing time and effort into optimizing this new ROOTS constraint.

9 Related Work

Beldiceanu has specified a wide range of global constraints using simple properties of suitably constructed graphs [Beldiceanu, 2000]. There are several important differences between this approach and the framework presented here. For instance, our specification language is executable. Given a specification, we immediately have a polynomial propagation algorithm. This is not the case with Beldiceanu's framework. On the other hand, Beldiceanu's framework is more general since the ROOTS and RANGE constraints can easily be specified in terms of some simple graph properties.

A number of researchers have specified global constraints over sequences of variables using automaton. Beldiceanu *et al.* use an extended polynomial sized automaton, and show how to extract automatically a GAC propagation algorithm [Beldiceanu *et al.*, 2004a]. Pesant uses a deterministic finite automaton, and develops a GAC propagation algorithm for membership of the corresponding regular language [Pesant, 2004]. These approaches are complementary to the framework presented here. For example, automaton can easily specify constraints like the CONTIGUITY constraint and thus provide us with a GAC propagation algorithm. This does not seem possible with just ROOTS and RANGE constraints. On the other hand, deterministic or extended automaton cannot specify a simple PERMUTATION constraint.

10 Conclusions

We have proposed a simple declarative language for specifying a wide range of counting and occurrence constraints. The language uses just two global primitives: the RANGE constraint which computes the range of values used by a set of variables, and the ROOTS constraint which computes the variables in a set mapping onto particular values. This specification language is executable. It immediately provides a polynomial propagation algorithm for any constraint that can be specified. In some cases, this propagation algorithm achieves GAC (e.g. the PERMUTATION and AMONG constraints). In other cases, this propagation algorithm may not make the constraint GAC, but achieving GAC is NP-hard (e.g. the NVALUE and COMMON constraints). Decomposition is then one method to obtain a polynomial algorithm. In the remaining cases, the propagation algorithm may not make the constraint GAC, and specialized propagation algorithms can do so in polynomial time (e.g. the SYMALLDIFF constraint). Our method can still be attractive in this last case as it provides a generic means of propagation for counting and occurrence constraints in the absence of a specialized algorithm. Experiments demonstrate that such propagation algorithms can be both efficient and effective in practice.

References

[Beldiceanu and Contejean, 1994] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20:97–123, no. 12 1994.

[Beldiceanu *et al.*, 2004a] N. Beldiceanu, M. Carlsson, and T. Petit. Deriving filtering algorithms from constraint checkers. In *Proc. of CP'2004*.

[Beldiceanu *et al.*, 2004b] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the *same* and *usedby* constraints. MPI Technical Report MPI-I-2004-1-001, 2004.

[Beldiceanu, 2000] N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. SICS TR T2000/01.

[Beldiceanu, 2001] N. Beldiceanu. Pruning for the minimum constraint family and for the number of distinct values constraint family. In *Proc. of CP'2001*.

[Bessiere *et al.*, 2004] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In *Proc. of AAAI'2004*.

[Cheng *et al.*, 1998] B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee, and J.C.K. Wu. Increasing Constraint Propagation by Redundant Modeling: an Experience Report. *Constraints*, 1998.

[Debruyne and Bessière, 1997] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proc. of IJCAI'97*.

[Maher, 2002] M. Maher. Analysis of a global contiguity constraint. In *Proc. of the CP'2002 Workshop on Rule Based Constraint Reasoning and Programming*, 2002.

[Pachet and Roy, 1999] F. Pachet and P. Roy. Automatic generation of music programs. In *Proc. of CP'99*.

[Pesant, 2004] G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proc. of CP'2004*.

[Quimper *et al.*, 2004] C.-G. Quimper, A. Lopez-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the global cardinality constraint. In *Proc. of CP'2004*.

[Refalo, 2000] P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Proc. of CP'2000*.

[Régin, 1994] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. of AAAI'94*.

[Régin, 1996] J-C. Régin. Generalized arc consistency for global cardinality constraints. In *Proc. of AAAI'96*.

[Régin, 1999] J-C. Régin. The symmetric alldiff constraint. In *Proc. of IJCAI'99*. Morgan Kaufmann, 1999.

[Van Hentenryck and Carillon, 1988] P. Van Hentenryck and J.-P. Carillon. Generality versus specificity: An experience with AI and OR techniques. In *Proc. of AAAI'88*.