The ROOTS Constraint

Christian Bessiere[1], Emmanuel Hebrard[2], Brahim Hnich[3], Zeynep Kiziltan[4], and Toby Walsh[5]

[1] LIRMM, CNRS/University of Montpellier, France, `bessiere@lirmm.fr`
[2] 4C and UCC, Cork, Ireland, `e.hebrard@4c.ucc.ie`
[3] Izmir University of Economics, Izmir, Turkey, `brahim.hnich@ieu.edu.tr`
[4] University of Bologna, Italy, `zkiziltan@deis.unibo.it`
[5] NICTA and UNSW, Sydney, Australia, `tw@cse.unsw.edu.au`

Abstract. A wide range of counting and occurrence constraints can be specified with just two global primitives: the RANGE constraint, which computes the range of values used by a sequence of variables, and the ROOTS constraint, which computes the variables mapping onto a set of values. We focus here on the ROOTS constraint. We show that propagating the ROOTS constraint completely is intractable. We therefore propose a decomposition which can be used to propagate the constraint in linear time. Interestingly, for all uses of the ROOTS constraint we have met, this decomposition does not destroy the global nature of the constraint as we still prune all possible values. In addition, even when the ROOTS constraint is intractable to propagate completely, we can enforce bound consistency in linear time simply by enforcing bound consistency on the decomposition. Finally, we show that specifying counting and occurrence constraints using ROOTS is effective and efficient in practice on two benchmark problems from CSPLib.

1 Introduction

Global constraints on the occurrence of particular values (*occurrence* constraints) or on the number of values or variables satisfying some condition (*counting* constraints) occur in many real world problems. They are especially useful in problems involving resources. For instance, if values represent resources, we may wish to count the number of occurrences of the different values used. Many global constraints proposed in the past are counting and occurrence constraints (see, for example, [13, 3, 14, 1, 4]). Bessiere et al. showed [5] that many such constraints can be specified with two new global constraints, ROOTS and RANGE, together with some simple elementary constraints like subset and set cardinality.

As we show here, specifying a global constraint using ROOTS and RANGE is also in many cases a way to provide an *efficient* propagator. There are three possible situations. In the first, we do not lose the "global" nature of our counting or occurrence constraint by specifying it with ROOTS and RANGE. The global nature of the ROOTS and RANGE constraint is enough to capture the global nature of the given counting or occurrence constraint, and propagation is not hindered. In the second situation, completely propagating the counting or occurrence constraint is NP-hard. We must accept some loss of globality if we are to make propagation tractable. Using ROOTS and RANGE is then one means to propagate the counting or occurrence constraint partially. In the third situation, the global constraint can be propagated completely in polynomial time but using ROOTS and RANGE hinders propagation. In this case, we need to develop a specialized propagation algorithm.

In [7], we focused on the RANGE constraint. This paper therefore concentrates on the ROOTS constraint. We prove that it is intractable to propagate the ROOTS constraint completely. We therefore propose a decomposition of the ROOTS constraint that can propagate it partially in linear time. This decomposition does not destroy the global nature of the ROOTS constraint as in many situations met in practice, it prunes all possible values. This decomposition can also easily be incorporated into a new constraint toolkit. We show experimentally the efficiency of using the ROOTS constraint on two real world problems from CSPLib. The rest of the paper is organised as follows. Section 2 gives the formal background. Section 3 gives many examples of counting and occurrence constraints that can be specified using the ROOTS constraint. In Section 4, we give a complete theoretical analysis of the ROOTS constraint and our decomposition of it. In Section 5, we discuss implementation details. Experimental results are presented in Section 6. Finally, we end with conclusions.

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 capitals for variables (e.g. X, Y and S), and lower case for values (e.g. v and w). We write $D(X)$ for the domain of a variable X. For totally ordered domains, we write $min(X)$ and $max(X)$ for the minimum and maximum values. 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 set variable S is represented by its lower bound $lb(S)$ which contains the definite elements and an upper bound $ub(S)$ which also contains the potential elements.

Constraint solvers typically explore partial assignments enforcing a local consistency property using either specialized or general purpose propagation algorithms. 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. A bound support in which each integer variable is assigned a value in its domain is called a *hybrid support*. 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 a hybrid support, and for each set variable 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. 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.

To illustrate these concepts, consider the constraint $C(X_1, X_2, S)$ that holds iff the set variable S is assigned exactly the values used by the integer variables X_1 and X_2. Let $D(X_1) = \{1, 3\}$, $D(X_2) = \{2, 4\}$, $lb(S) = \{2\}$ and $ub(S) = \{1, 2, 3, 4\}$. BC does not remove any value since all domains are already bound consistent. On the other hand, HC removes 4 from $D(X_2)$ and from $ub(S)$ as there does not exist any tuple satisfying C in which X_2 does not take value 2. Note that as BC deals with bounds, value 2 was considered as possible for X_1.

3 Counting and Occurrence Constraints

Counting constraints limit the number of values or variables satisfying some condition (e.g. the global cardinality constraint [14] counts the number of variables using particular values). Occurrence constraints limit the occurrence of particular values (e.g. the all different constraint [13] ensures no value occurs twice). We previously showed [5] that many counting and occurrence constraints can be decomposed into two new global constraints, Range and Roots, together with simple non-global constraints over integer variables (like $X \leq m$) and simple non-global constraints over set variables (like $S_1 \subseteq S_2$ or $|S| = k$). We focus here on the Roots constraint. Given a sequence of variables X_1 to X_n, the Roots constraint holds iff a set variable S is the set of indices of variables which map to a value belonging to a second set variable, T.

$$\textsc{Roots}([X_1, \ldots, X_n], S, T) \text{ iff } S = \{i \mid X_i \in T\}$$

Note that elements in T may not be used by any integer variable X_i. For example, $\textsc{Roots}([1, 3, 1, 2, 3], S, T)$ is satisfied by $S = \{1, 3\}$ and $T = \{1\}$, $S = \{4\}$ and $T = \{2, 7\}$, or $S = \{2, 4, 5\}$ and $T = \{2, 3, 8\}$. We now list some of the uses of the Roots constraint for specifying other more complex global constraints.

3.1 Among constraint

The Among constraint was introduced in CHIP to model resource allocation problems like car sequencing [3]. It counts the number of variables using values from a given set. $\textsc{Among}([X_1, \ldots, X_n], [d_1, \ldots, d_m], N)$ holds iff $N = |\{i \mid X_i \in \{d_1, \ldots, d_m\}\}|$. It can be decomposed using a Roots constraint:

$$\begin{aligned}&\textsc{Among}([X_1, \ldots, X_n], [d_1, \ldots, d_m], N) \text{ iff}\\ &\textsc{Roots}([X_1, \ldots, X_n], S, \{d_1, \ldots, d_m\}) \ \wedge\ |S| = N\end{aligned}$$

GAC on AMONG is equivalent to HC on this decomposition [5]. As we show later, since the third argument of ROOTS is ground, we can achieve HC on the ROOTS constraint in linear time. We note that ROOTS is more than a set version of AMONG. With AMONG, we just count the number of variables using particular values. However, with ROOTS, we collect the set of variables using particular values. As we see later, having this set and not just its cardinality permits us to specify global constraints like COMMON which go beyond what can be expressed with AMONG.

3.2 COUNT constraint

The COUNT constraint [2] is closely related to the AMONG constraint. The COUNT constraint permits us to constrain the number of variables using a particular value. More precisely, $\text{COUNT}([X_1, \ldots, X_n], d, op, N)$ where $op \in \{\leq, \geq, <, >, =, \neq\}$ holds iff $|\{i \mid X_i = d\}|\ op\ N$. The ATMOST and ATLEAST constraints are instances of COUNT where $op \in \{\leq, \geq\}$. The COUNT constraint can be decomposed into a ROOTS constraint:

$$\begin{aligned}&\text{COUNT}([X_1, \ldots, X_n], d, op, N) \text{ iff}\\&\text{ROOTS}([X_1, \ldots, X_n], S, \{d\})\ \&\ |S|\ op\ N\end{aligned}$$

This decomposition does not hinder propagation and, as we will show later, it takes linear time to enforce HC on such an instance of the ROOTS constraint.

3.3 DOMAIN constraint

We may wish to channel between a variable and a sequence of 0/1 variables representing the possible values taken by the variable. The $\text{DOMAIN}(X, [X_1, \ldots, X_m])$ constraint introduced in [12] ensures $X = i$ iff $X_i = 1$. This can be decomposed into a ROOTS constraint:

$$\begin{aligned}&\text{DOMAIN}(X, [X_1, \ldots, X_m]) \text{ iff}\\&\text{ROOTS}([X_1, \ldots, X_m], S, \{1\})\ \&\ |S| = 1\ \&\ X \in S\end{aligned}$$

Enforcing HC on this specification again takes linear time and it is equivalent to enforcing GAC on the original global DOMAIN constraint.

3.4 LINKSET2BOOLEANS constraint

We may also wish to channel between a set variable and a sequence of 0/1 variables representing the characteristic function of this set. The global constraint $\text{LINKSET2BOOLEANS}(S, [X_1, \ldots, X_m])$ introduced in [2] ensures $i \in S$ iff $X_i = 1$. This can also be decomposed into a ROOTS constraint:

$$\begin{aligned}&\text{LINKSET2BOOLEANS}(S, [X_1, \ldots, X_m]) \text{ iff}\\&\text{ROOTS}([X_1, \ldots, X_m], S, \{1\})\end{aligned}$$

Enforcing HC (or BC) on this specification again takes linear time and it is equivalent to enforcing HC (or BC) on the original global LINKSET2BOOLEANS constraint.

3.5 Gcc constraint

The global cardinality constraint [14] constrains the number of times values are used. We consider a generalization in which the number of occurrences of a value is an integer variable. That is, $\text{Gcc}([X_1, \ldots, X_n], [d_1, \ldots, d_m], [O_1, \ldots, O_m])$ 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:

$$\begin{array}{l} \text{Gcc}([X_1, \ldots, X_n], [d_1, \ldots, d_m], [O_1, \ldots, O_m]) \;\text{ iff} \\ \forall j \;.\; \text{Roots}([X_1, \ldots, X_n], S_j, \{d_j\}) \;\&\; |S_j| = O_j \end{array}$$

Enforcing GAC on such 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 [11]. This is more than is achieved in general by enforcing HC on the specification using Roots [5].

3.6 Common constraint

A generalization of the Among and AllDifferent constraints introduced in [2] is the Common constraint. $\text{Common}(N, M, [X_1, \ldots, X_n], [Y_1, \ldots, Y_m])$ ensures $N = |\{i \mid X_i = Y_j\}|$ and $M = |\{j \mid X_i = Y_j\}|$. That is, N variables in $X_1, \ldots, X_n$ take values in common with $Y_1, \ldots, Y_m$ and M variables in $Y_1, \ldots, Y_m$ take values in common with $X_1, \ldots, X_n$. We cannot expect to enforce GAC on such a constraint in general as it is NP-hard to do so [5]. One way to propagate a Common constraint is to decompose it into Range and Roots constraints:

$$\begin{array}{l} \text{Common}(N, M, [X_1, \ldots, X_n], [Y_1, \ldots, Y_m]) \;\text{ iff} \\ \text{Range}([Y_1, \ldots, Y_m], \{1, \ldots, m\}, T) \;\& \\ \text{Roots}([X_1, \ldots, X_n], S, T) \;\&\; |S| = N \;\& \\ \text{Range}([X_1, \ldots, X_n], \{1, \ldots, n\}, V) \;\& \\ \text{Roots}([Y_1, \ldots, Y_m], U, V) \;\&\; |U| = M \end{array}$$

where the Range constraint holds iff a set variable T equals the set of values used by those variables, X_1 to X_n whose index is in the set S.

$$\text{Range}([X_1, \ldots, X_n], S, T) \text{ iff } T = \{X_i \mid i \in S\}$$

Enforcing HC on this specification of the Common constraint again takes linear time. As no specialized propagation algorithm has yet been proposed for the Common constraint, Roots and Range provide a simple and promising means to propagate the constraint.

4 The Roots constraint

We now give a thorough theoretical analysis of the Roots constraint. In Section 4.1, we provide a proof for the first time of the claim made in [5] that enforcing

HC on ROOTS is NP-hard in general. Section 4.2 presents a decomposition of the ROOTS constraint that permits us to propagate the ROOTS constraint partially in linear time. Section 4.3 shows that in many cases this decomposition does not destroy the global nature of the ROOTS constraint as enforcing HC on the decomposition achieves HC on the ROOTS constraint. Finally, Section 4.4 shows that we can obtain BC on the ROOTS constraint by enforcing BC on its decomposition.

4.1 Complete propagation

Unfortunately, propagating the ROOTS constraint completely is intractable in general. Whilst we made this claim in [5], a proof has not yet been published. For this reason, we give one here.

Theorem 1. *Enforcing HC on the* ROOTS *constraint is NP-hard.*

Proof. We transform 3SAT into the problem of the existence of a solution for ROOTS. Finding a hybrid support is thus NP-hard. Hence enforcing HC on ROOTS is NP-hard. Let $\varphi = \{c_1, \ldots, c_m\}$ be a 3CNF on the Boolean variables $x_1, \ldots, x_n$. We build the constraint ROOTS$([X_1, \ldots, X_{n+m}], S, T)$ as follows. Each Boolean variable x_i is represented by the variable X_i with domain $D(X_i) = \{i, -i\}$. Each clause $c_p = x_i \vee \neg x_j \vee x_k$ is represented by the variable X_{n+p} with domain $D(X_{n+p}) = \{i, -j, k\}$. We build S and T in such a way that it is impossible for both the index i of a Boolean variable x_i and its complement $-i$ to belong to T. We set $lb(T) = \emptyset$ and $ub(T) = \bigcup_{i=1}^{n}\{i, -i\}$, and $lb(S) = ub(S) = \{n+1, \ldots, n+m\}$. An interpretation M on the Boolean variables $x_1, \ldots, x_n$ is a model of φ iff the tuple τ in which $\tau[X_i] = i$ iff $M[x_i] = 0$ can be extended to a solution of ROOTS. (This extension puts in T value i iff $M[x_i] = 1$ and assigns X_{n+p} with the value corresponding to the literal satisfying c_p in M.) □

We thus have to look for a lesser level of consistency for ROOTS or for particular cases on which HC is polynomial. We will show that bound consistency is tractable and that, under conditions often met in practice (e.g. one of the last two arguments of ROOTS is ground), enforcing HC is also.

4.2 A decomposition of ROOTS

To show that ROOTS can be propagated tractably, we will give a straightforward decomposition into ternary constraints that can be propagated in linear time. This decomposition does not destroy the global nature of the ROOTS constraint since enforcing HC on the decomposition will, in many cases, achieve HC on the original ROOTS constraint, and since in all cases, enforcing BC on the decomposition achieves BC on the original ROOTS constraint. Given ROOTS$([X_1, \ldots, X_n], S, T)$, we decompose it into the implications:

$$i \in S \rightarrow X_i \in T$$
$$X_i \in T \rightarrow i \in S$$

where $i \in [1..n]$. We have to be careful how we implement such a decomposition in a constraint solver. First, some solvers will not achieve HC on such constraints (see Sec 5 for more details). Second, we need an efficient algorithm to be able to propagate the decomposition in linear time. As we explain in more detail in Sec 5, a constraint solver could easily take quadratic time if it is not incremental.

We first show that this decomposition prevents us from propagating the ROOTS constraint completely. However, this is to be expected as propagating ROOTS completely is NP-hard and this decomposition is linear to propagate. In addition, as we later show, in many circumstances met in practice, the decomposition does not in fact hinder propagation.

Theorem 2. *HC on* ROOTS$([X_1, \ldots, X_n], S, T)$ *is strictly stronger than HC on* $i \in S \rightarrow X_i \in T$*, and* $X_i \in T \rightarrow i \in S$ *for all* $i \in [1..n]$.

Proof. Consider $X_1 \in \{1, 2\}$, $X_2 \in \{3, 4\}$, $X_3 \in \{1, 3\}$, $X_4 \in \{2, 3\}$, $lb(S) = ub(S) = \{3, 4\}$, $lb(T) = \emptyset$, and $ub(T) = \{1, 2, 3, 4\}$. The decomposition is HC. However, enforcing HC on ROOTS will prune 3 from $D(X_2)$. □

In fact, enforcing HC on the decompostion achieves a level of consistency between BC and HC on the original ROOTS constraint. In the next section, we identify exactly when it achieves HC on ROOTS.

4.3 Some special cases

Many of the counting and occurrence constraints do not use the ROOTS constraint in its more general form, but have some restrictions on the variables S, T or X_i's. For example, it is often the case that T or S are ground. We select four important cases that cover many of these uses of ROOTS and show that enforcing HC on ROOTS is then tractable.

C1. $\forall i \in lb(S), D(X_i) \subseteq lb(T)$
C2. $\forall i \notin ub(S), D(X_i) \cap ub(T) = \emptyset$
C3. $X_1 \ldots X_n$ are ground
C4. T is ground

We will show that in any of these cases, we can achieve HC on ROOTS simply by propagating the decomposition.

Theorem 3. *If one of the conditions $C1$ to $C4$ holds, then enforcing HC on* $i \in S \rightarrow X_i \in T$*, and* $X_i \in T \rightarrow i \in S$ *for all* $i \in [1..n]$ *achieves HC on* ROOTS$([X_1, \ldots, X_n], S, T)$.

Proof. Soundness. Immediate.

Completeness. We observe that if the ROOTS constraint is unsatisfiable then enforing HC on the decomposition will also fail. We assume therefore that the ROOTS constraint is satisfiable. We have to prove that, for any X_i, all the values in $D(X_i)$ belong to a solution of ROOTS, and that the bounds on S and T are as tight as possible. Our proof will exploit the following properties that are guaranteed to hold when we have enforced HC on the decomposition.

P1 if $D(X_i) \subseteq lb(T)$ then $i \in lb(S)$
P2 if $D(X_i) \cap ub(T) = \emptyset$ then $i \notin ub(S)$
P3 if $i \in lb(S)$ then $D(X_i) \subseteq ub(T)$
P4 if $i \notin ub(S)$ then $D(X_i) \cap lb(T) = \emptyset$
P5 if $D(X_i) = \{v\}$ and $i \in lb(S)$ then $v \in lb(T)$
P6 if $D(X_i) = \{v\}$ and $i \notin ub(S)$ then $v \notin ub(T)$
P7 if i is added to $lb(S)$ by the constraint $X_i \in T \to i \in S$ then $D(X_i) \subseteq lb(T)$
P8 if i is deleted from $ub(S)$ by the constraint $i \in S \to X_i \in T$ then $D(X_i) \cap ub(T) = \emptyset$

Let us prove that $lb(T)$ is tight. Suppose the tuple τ is a solution of the ROOTS constraint. Let $v \notin lb(T)$ and $v \in \tau[T]$. We show that there exists a solution with $v \notin \tau[T]$. (Remark that this case is irrelevant to condition C4.) We remove v from $\tau[T]$. For each $i \notin lb(S)$ such that $\tau[X_i] = v$ we remove i from $\tau[S]$. With C1 we are sure that none of the i in $lb(S)$ have $\tau[X_i] = v$, thanks to property P7 and the fact that $v \notin lb(T)$. With C3 we are sure that none of the i in $lb(S)$ have $\tau[X_i] = v$, thanks to property P5 and the fact that $v \notin lb(T)$. There remains to check C2. For each $i \in lb(S)$, we know that $\exists v' \neq v, v' \in D(X_i) \cap ub(T)$, thanks to properties P3 and P5. We set X_i to v' in τ, we add v' to $\tau[T]$ and add all k with $\tau[X_k] = v'$ to $\tau[S]$. We are sure that $k \in ub(S)$ because $v' \in ub(T)$ plus condition C2 and property P8.

Completeness on $ub(T)$, $lb(S)$, $ub(S)$ and X_i's are shown with similar proofs. Let $v \in ub(T) \setminus \tau[T]$. (Again C4 is irrelevant.) We show that there exists a solution with $v \in \tau[T]$. Add v to $\tau[T]$ and for each $i \in ub(S)$, if $\tau[X_i] = v$, put i in $\tau[S]$. C2 is solved thanks to property P8 and the fact that $v \in ub(T)$. C3 is solved thanks to property P6 and the fact that $v \in ub(T)$. There remains to check C1. For each $i \notin ub(S)$ and $\tau[X_i] = v$, we know that $\exists v' \neq v, v' \in D(X_i) \setminus lb(T)$ (thanks to properties P4 and P6). We set X_i to v' in τ and remove v' from $\tau[T]$. Each k with $\tau[X_k] = v'$ is removed from $\tau[S]$, and this is possible because we are in condition C1, $v' \notin lb(T)$, and thanks to property P7.

Let $v \in D(X_i)$ and $\tau[X_i] = v', v' \neq v$. (C3 is irrelevant.) Assign v to X_i in τ. If both v and v' or none of them are in $\tau[T]$, we are done. There remain two cases. First, if $v \in \tau[T]$ and $v' \notin \tau[T]$, the two alternatives to satisfy ROOTS are to add i in $\tau[S]$ or to remove v from $\tau[T]$. If $i \in ub(S)$, we add i to $\tau[S]$ and we are done. If $i \notin ub(S)$, we know that $v \notin lb(T)$ thanks to property P4. So, v is removed from $\tau[T]$ and we are sure that the X_j's can be updated consistently for the same reason as in the proof of $lb(T)$. Second, if $v \notin \tau[T]$ and $v' \in \tau[T]$, the two alternatives to satisfy ROOTS are to remove i from $\tau[S]$ or to add v to $\tau[T]$. If $i \notin lb(S)$, we remove i from $\tau[S]$ and we are done. If $i \in lb(S)$, we know that $v \in ub(T)$ thanks to property P3. So, v is added to $\tau[T]$ and we are sure that the X_j's can be updated consistently for the same reason as in the proof of $ub(T) \setminus \tau[T]$.

Let $i \notin lb(S)$ and $i \in \tau[S]$. We show that there exists a solution with $i \notin \tau[S]$. We remove i from $\tau[S]$. Thanks to property P1, we know that $D(X_i) \not\subseteq lb(T)$. So, we set X_i to a value $v' \in D(X_i) \setminus lb(T)$. With C4 we are done because we are sure $v' \notin \tau[T]$. With conditions C1, C2, and C3, if $v' \in \tau[T]$, we remove it from $\tau[T]$ and we are sure that the X_j's can be updated consistently for the same reason as in the proof of $lb(T)$.

Let $i \in ub(S) \setminus \tau[S]$. We show that there exists a solution with $i \in \tau[S]$. We add i to $\tau[S]$. Thanks to property P2, we know that $D(X_i) \cap ub(T) \neq \emptyset$. So, we set X_i to a value $v' \in D(X_i) \cap ub(T)$. With condition C4 we are done because we are sure $v' \in \tau[T]$. With conditions C1, C2, and C3, if $v' \notin \tau[T]$, we add it to $\tau[T]$ and we are sure that the X_j's can be updated consistently for the same reason as in the proof of $ub(T) \setminus \tau[T]$. □

4.4 Bound consistency

In addition to being able to enforce HC on Roots in some special cases, enforcing HC on the decomposition always enforces a level of consistency at least as strong as BC. In fact, in any situation (even those where enforcing HC in intractable), enforcing BC on the decomposition enforces BC on the Roots constraint.

Theorem 4. *Enforcing BC on $i \in S \rightarrow X_i \in T$, and $X_i \in T \rightarrow i \in S$ for all $i \in [1..n]$ achieves BC on* Roots$([X_1, \ldots, X_n], S, T)$.

Proof. Soundness. Immediate.
Completeness. The proof follows the same structure as that in Theorem 3. We relax the properties P1–P4 into properties P1'–P4'.

P1' if $[min(X_i), max(X_i)] \subseteq lb(T)$ then $i \in lb(S)$
P2' if $[min(X_i), max(X_i)] \cap ub(T) = \emptyset$ then $i \notin ub(S)$
P3' if $i \in lb(S)$ then the bounds of X_i are included in $ub(T)$
P4' if $i \notin ub(S)$ then the bounds of X_i are outside $lb(T)$

Let us prove that $lb(T)$ and $ub(T)$ are tight. Let o be the total ordering on $D = \bigcup_i D(X_i) \cup ub(T)$. Build the tuples σ and τ as follows: For each $v \in lb(T)$: put v in $\sigma[T]$ and $\tau[T]$. For each $v \in ub(T) \setminus lb(T)$, following o, do: put v in $\sigma[T]$ or $\tau[T]$ alternately. For each $i \in lb(S)$, P3' guarantees that both $min(X_i)$ and $max(X_i)$ are in $ub(T)$. By construction of $\sigma[T]$ (and $\tau[T]$) with alternation of values, if $min(X_i) \neq max(X_i)$, we are sure that there exists a value in $\sigma[T]$ (in $\tau[T]$) between $min(X_i)$ and $max(X_i)$. In the case $|D(X_i)| = 1$, P5 guarantees that the only value is in $\sigma[T]$ (in $\tau[T]$). Thus, we assign X_i in σ (in τ) with such a value in $\sigma[T]$ (in $\tau[T]$). For each $i \notin ub(S)$, we assign X_i in σ with a value in $[min(X_i), max(X_i)] \setminus \sigma[T]$ (the same for τ). We know that such a value exists with the same reasoning as for $i \in lb(S)$ on alternation of values, and thanks to P4' and P6. We complete σ and τ by building $\sigma[S]$ and $\tau[S]$ consistently with the assignments of X_i and T. The resulting tuples satisfy Roots. From this we deduce that $lb(T)$ and $ub(T)$ are BC as all values in $ub(T) \setminus lb(T)$ are either in σ or in τ, but not both.

We show that the X_i are BC. Take any X_i and its lower bound $min(X_i)$. If $i \in lb(S)$ we know that $min(X_i)$ is in T either in σ or in τ thanks to P3' and by construction of σ and τ. We assign $min(X_i)$ to X_i in the relevant tuple. This remains a solution of Roots. If $i \notin ub(S)$, we know that $min(X_i)$ is outside

T either in σ or in τ thanks to P4' and by construction of σ and τ. We assign $min(X_i)$ to X_i in the relevant tuple. This remains a solution of ROOTS. If $i \in ub(S) \setminus lb(S)$, assign X_i to $min(X_i)$ in σ. If $min(X_i) \notin \sigma[T]$, remove i from $\sigma[S]$ else add i to $\sigma[S]$. The tuple obtained is a solution of ROOTS using the lower bound of X_i. By the same reasoning, we show that the upper bound of X_i is BC also, and therefore, all X_i's are BC.

We prove that $lb(S)$ and $ub(S)$ are BC with similar proofs. Let us show that $ub(S)$ is BC. Take any X_i with $i \in ub(S)$ and $i \notin \sigma[S]$. Since X_i was assigned any value from $[min(X_i), max(X_i)]$ when σ was built, and since we know that $[min(X_i), max(X_i)] \cap ub(T) \neq \emptyset$ thanks to P2', we can modify σ by assigning X_i a value in $ub(T)$, putting the value in T if not already there, and adding i into S. The tuple obtained satisfies ROOTS. So $ub(S)$ is BC.

There remains to show that $lb(S)$ is BC. Thanks to P1', we know that values $i \in ub(S) \setminus lb(S)$ are such that $[min(X_i), max(X_i)] \setminus lb(T) \neq \emptyset$. Take $v \in [min(X_i), max(X_i)] \setminus lb(T)$. Thus, either σ or τ is such that $v \notin T$. Take the corresponding tuple, assign X_i to v and remove i from S. The modified tuple is still a solution of ROOTS and $lb(S)$ is BC. $\square$

5 Implementation Details

This decomposition of the ROOTS constraint can be implemented in many solvers using disjunctions of membership constraints: $\texttt{or}(\texttt{member}(i, S), \texttt{notmember}(X_i, T))$ and $\texttt{or}(\texttt{notmember}(i, S), \texttt{member}(X_i, T))$. However, this requires a little care. Unfortunately, some existing solvers (like Ilog Solver) may not achieve HC on such disjunctions of primitives. For instance, the negated membership constraint $\texttt{notmember}(X_i, T)$ is activated only if X_i is instantiated with a value of T (whereas it should be as soon as $D(X_i) \subseteq lb(T)$). We have to ensure that the solver wakes up when it should to ensure we achieve HC. As we explain in the complexity proof, we also have to be careful that the solver doesn't wake up too often or we will lose the optimal $O(nd)$ time complexity which can be achieved.

Theorem 5. *It is possible to enforce HC (or BC) on the decomposition of* ROOTS$([X_1, \ldots, X_n], S, T)$ *in* $O(nd)$ *time, where* $d = max(\forall i.|D(X_i)|, |ub(T)|)$.

Proof. The decomposition of ROOTS is composed of $2n$ constraints. To obtain an overall complexity in $O(nd)$, the total amount of work spent propagating each of these constraints must be in $O(d)$.

First, it is necessary that each of the $2n$ constraints of the decomposition is not called for propagation more than d times. Since S can be modified up to n times (n can be larger than d) it is important that not all constraints are called for propagation at each change in $lb(S)$ or $ub(S)$. By implementing 'propagating events' as described in [10, 15], we can ensure that when a value i is added to $lb(S)$ or removed from $ub(S)$, constraints $j \in S \rightarrow X_j \in T$ and $X_j \in T \rightarrow j \in S$, $j \neq i$, are not called for propagation.

Second, we show that enforcing HC on contraint $i \in S \rightarrow X_i \in T$ in $O(d)$. Testing the precondition (does i belong to $lb(S)$?) is constant time. If true, removing from $D(X_i)$ all values not in $ub(T)$ is in $O(d)$ and updating $lb(T)$ (if $|D(X_i)| = 1$) is constant time. Testing that the postcondidtion is false (is $D(X_i)$ disjoint from $ub(T)$?) is in $O(d)$. If false, updating $ub(S)$ is constant time. Thus HC on $i \in S \rightarrow X_i \in T$ is in $O(d)$. Enforcing HC on $X_i \in T \rightarrow i \in S$ is in $O(d)$ as well because testing the precondition ($D(X_i) \subseteq lb(T)$?) is in $O(d)$, updating $lb(S)$ is constant time, testing that the postcondition is false ($i \notin ub(S)$?) is constant time, and removing from $D(X_i)$ all values in $lb(T)$ is in $O(d)$ and updating $ub(T)$ (if $|D(X_i)| = 1$) is constant time.

When T is modified, all constraints are potentially concerned. Since T can be modified up to d times, we can have d calls of the propagation in $O(d)$ for each of the $2n$ constraints. It is thus important that the propagation of the $2n$ constraints is *incremental* to avoid an $O(nd^2)$ overall complexity. An algorithm for $i \in S \rightarrow X_i \in T$ is incremental if the complexity of calling the propagation of the constraint $i \in S \rightarrow X_i \in T$ up to d times (once for each change in T or $D(X_i)$) is the same as propagating the constraint once. This can be achieved by an AC2001-like algorithm that stores the last value found in $D(X_i) \cap ub(T)$, which is a witness that the postcondition can be true. (Similarly, the last value found in $D(X_i) \setminus lb(T)$ is a witness that the precondition of the constraint $X_i \in T \rightarrow i \in S$ can be false.) Finally, each time $lb(T)$ (resp. $ub(T)$) is modified, $D(X_i)$ must be updated for each i outside $ub(S)$ (resp. inside $lb(S)$). If the propagation mechanism of the solver provides the values that have been added to $lb(T)$ or removed from $ub(T)$ to the propagator of the $2n$ constraints (as described in [16]), updating a given $D(X_i)$ has a total complexity in $O(d)$ for the d possible changes in T. The proof that BC can also be enforced in linear time follows a similar argument. □

6 Experimental Results

We now demonstrate that specifying global counting and occurrence constraints using Roots is effective and efficient in practice using two benchmark problems.

6.1 Balanced Academic Curriculum Problem

We implemented in Ilog Solver the constraint model of the Balanced Academic Curriculum Problem (BACP) (prob030 in CSPLib) proposed in [9] and compared it against a model using Roots. In this problem, we need to design a balanced academic curriculum by assigning periods to courses so that the academic load of each period is balanced, i.e., as similar as possible. The goal is to assign a period to every course so that the constraints on the minimum and maximum academic load for each period, the minimum and maximum number of courses for each period, and the prerequisite relationships are satisfied. An optimal balanced curriculum minimises the maximum academic load for all periods.

We used two models from [9] (Figures 1 and 2) and compared them against a model using Roots (Figure 3). In the Roots model, the curriculum is encoded with integer variables mapping courses to periods, as in the primal-dual model

Variables	Encoding
curriculum:	CURMATRIX[1..#courses][1..#periods] in $\{0,1\}$
academic load:	LOAD[1..#periods] in $[a..b]$
Constraints	**Encoding**
exactly one period per course	$\forall i \in [1..\#courses]$ $\sum_{j=1}^{\#periods}$CURMATRIX$[i][j] = 1$
academic load	$\forall j \in [1..\#periods]$ LOAD$[j] = \sum_{i=1}^{\#courses}$(CURMATRIX$[i][j]$ * $credit[i]$)
prerequisites	$\forall (i \prec j) \in$ prerequisites, $\forall k \in$ [1..#periods] $\sum_{r=1}^{k-1}$(CURMATRIX$[i][r]$) $\geq$CURMATRIX$[i][k]$
number of courses	$\forall j \in [1..\#periods]$ $c \leq \sum_{i=1}^{\#courses}$CURMATRIX$[i][j] \leq d$

Fig. 1. Boolean model.

Variables	Encoding
curriculum:	CURRICULUM[1..#courses] in [1..#periods] CURMATRIX[1..#courses][1..#periods] in $\{0,1\}$
academic load:	LOAD[1..#periods] in $[a..b]$
Constraints	**Encoding**
channeling	$\forall i \in$ [1..#courses] CURMATRIX$[i]$[CURRICULUM$[i]$] = 1
academic load	$\forall j \in$ [1..#periods] LOAD$[j] = \sum_{i=1}^{\#\text{courses}}$(CURMATRIX$[i][j]$ * $credit[i]$)
prerequisites	$\forall (i \prec j) \in$ prerequisites CURRICULUM$[i]$ <CURRICULUM$[j]$
number of courses	Gcc($[c..d]$, ..$[c..d]$, $\{1, 2, ..\#\text{periods}\}$,CURRICULUM)

Fig. 2. Primal-dual model.

of Figure 2. However, instead of using a Gcc constraint to restrict the number of courses per periods, we use the Roots constraint to link these variables to set variables standing for periods. We then restrict the number of courses per periods with cardinality constraints on these sets. The constants a, b, c, d correspond respectively to the minimum and maximum academic load, and the minimum and maximum number of courses per period. The array $credit$[1..#courses] map courses to their academic credits. We added to all models the implied constraint $\sum_{j=1}^{\#\text{periods}}(\text{LOAD}[j]) = \sum_{i=1}^{\#\text{courses}}(credit[i])$

We report in Table 1 the number of fails, cpu time for finding the best solution and the maximum academic load on 6 different instances. The 3 first instances, involving 8, 10 and 12 periods, are those solved in [9]. The 3 next instances were created by simply duplicating and renaming courses. The number of periods and courses are doubled, and for each prerequisite relation $(i \prec j)$ in the initial instance, we add $(i \prec j')$ and $(i' \prec j')$ where i' and j' are the duplicated courses for respectively i and j.

Variables	Encoding
curriculum:	CURRICULUM[1..#courses] in [1..#periods] CURSET[1..#periods] $\subseteq$ {1..#courses}
academic load:	LOAD[1..#periods] in $[a..b]$

Constraints	Encoding
channeling	$\forall j \in$ [1..#periods] Roots(CURRICULUM,CURSET[j], $\{j\}$)
academic load	$\forall j \in$ [1..#periods] $\mathrm{LOAD}[j] = \sum_{i=1}^{\#\mathrm{courses}} ((i \in \mathrm{CURSET}[j]) * \mathit{credit}[i])$
prerequisites	$\forall (i \prec j) \in$ prerequisites CURRICULUM[i] <CURRICULUM[j]
number of courses	$\forall j \in$ [1..#periods] $c \leq \lvert \mathrm{CURSET}[j] \rvert \leq d$

Fig. 3. Roots model.

Table 1. Balanced Academic Curriculum Problem.

	Boolean model			Prima-dual model			Roots model		
Size	#fails	time (s)	max load	#fails	time (s)	max load	#fails	time (s)	max load
8	413,418	18.44	17(*)	294	**0.04**	17(*)	**75**	0.09	17(*)
10	-	-	14(*)	170	**0.02**	14(*)	**121**	0.15	14(*)
12	1251	0.05	17(*)	255	**0.05**	17(*)	**194**	0.51	17(*)
16	**-**	**-**	**-**	429	**0.15**	17(*)	**263**	1.58	17(*)
20	-	-	-	410	**0.21**	19	**406**	3.18	19
20	-	-	-	701	**0.41**	18	**510**	13.12	18

When the maximum academic load is followed by a star (*), it means that this is optimal and is proved so with a few more backtracks. The time cutoff was set to 300 seconds. An entry marked as "-" means no answer was obtained by the cut-off time. We observe that the model using Roots is the most efficient in terms of size of the search tree by a small margin. However the most efficient model in cpu time is the primal-dual model which uses the highly optimized Gcc constraint. Both clearly dominate the simple Boolean model despite the fact that this model only has linear constraints.

6.2 Mystery Shopper Problem

We used a model for the Mystery Shopper problem [8] due to Helmut Simonis that appears in CSPLib (prob004). We used the same problem instances as in [5] but perform a more thorough and extensive analysis. We partition the constraints of this problem into three groups:

Temporal and geographical: All visits for any week are made by different shoppers. Similarly, a particular area cannot be visited more than once by the same shopper.

Shopper: Each shopper makes exactly the required number of visits.

Saleslady: A saleslady must be visited by some shoppers from at least 2 different groups (the shoppers are partitioned into groups).

Table 2. Mystery Shopper, branching on the integer variable with minimum domain.

	`Alld-Gcc-Sum`			`Alld-Gcc`-ROOTS			`Alld`-ROOTS-ROOTS		
Size	#fails	time (s)	#solved	#fails	time (s)	#solved	#fails	time (s)	#solved
10	6	0.01	**9/10**	6	0.01	**9/10**	7	0.03	**9/10**
15	6,566	0.76	**29/52**	6,468	1.38	**29/52**	10,749	19.47	28/52
20	98,497	14.52	**21/35**	2,425	0.83	20/35	2,429	7.30	20/35
25	317	0.13	**13/20**	317	0.20	**13/20**	285	1.37	11/20
30	93,461	26.09	**7/10**	93,461	43.89	**7/10**	7,239	42.00	5/10
35	52,435	16.33	**22/56**	23,094	14.25	21/56	13	1.10	18/56

Table 3. Mystery Shopper, branching on set variables when possible.

	`Alld-Gcc-Sum`			`Alld-Gcc`-ROOTS			`Alld`-ROOTS-ROOTS		
Size	#fails	time (s)	#solved	#fails	time (s)	#solved	#fails	time (s)	#solved
10	6	0.01	9/10	4247	0.83	3/10	318	0.38	**10/10**
15	6566	0.76	29/52	17210	4.31	16/52	102	0.25	**52/52**
20	98497	14.52	21/35	150473	49.95	7/35	930	2.95	**32/35**
25	317	0.13	13/20	265219	124.49	2/20	2334	11.17	**19/20**
30	93461	26.09	7/10	37	0.08	1/10	6766	39.63	**9/10**
35	52435	16.33	22/56	1216	0.53	4/56	4798	35.60	**49/56**

Whilst the first group of constraints can be modelled by using ALLDIFFERENT constraints [13], the second can be modelled by GCC [14] and the third by AMONG constraints [3]. We experimented with several models using Ilog Solver where these constraints are either implemented as their Ilog Solver primitives (respectively, `IloAllDiff`, `IloDistribute`, and a decomposition using `IloSum` on Boolean variables) or as their decompositions with ROOTS. Note that the Boolean decomposition of the AMONG constraint maintains GAC [6]. Due to space limitation, we report results for just the following models: `Alld-Gcc-Sum` (only Ilog Solver primitives), `Alld-Gcc`-ROOTS (AMONG encoded as ROOTS), and `Alld`-ROOTS-ROOTS (AMONG and GCC encoded as ROOTS). AMONG encoded as ROOTS uses the decomposition presented in Section 3.1 and GCC uses the decomposition presented in Section 3.5. All instances solved in the experiments use a time limit of 5 minutes. The cpu time reported for a method on a class of problems is averaged on the instances solved (#solved) by the method.

When branching on the integer variables (Table 2), the `Alld-Gcc-Sum` model tends to perform better than the other models (bold numbers). However, we obtain the best results by branching on the set variables introduced for modelling with ROOTS (see Table 3). By encoding the second and the third groups of constraints using ROOTS (the `Alld`-ROOTS-ROOTS model) and branching on the set variables, we are able to solve more instances. These results are primarily due to the better branching strategy. However, such a strategy would not be easily implementable without ROOTS since the extra set variables are part of it.

7 Conclusion

We have presented a comprehensive study of ROOTS, a global constraint that can specify many other global constraints, such as occurrence and counting constraints. We proved that propagating completely the ROOTS constraint is intractable in general. We therefore proposed a decomposition to propagate it partially. This decomposition achieves hybrid consistency on the global ROOTS constraint under some simple conditions often met in practice. In addition, enforcing bound consistency on the decomposition achieves bound consistency on the global ROOTS constraint whatever conditions hold. Our experiments show that this is practical method to implement many global constraints. We hope that by presenting these results, developers of the many different constraint toolkits will be encouraged to include the ROOTS constraint into their solvers. In our future work, we intend to consider other classes of global constraints (e.g. sequencing constraints) and to identify the primitives needed to specify and propagate these.

References

1. N. Beldiceanu. Pruning for the minimum constraint family and for the number of distinct values constraint family. In *Proc. CP'01*, pp. 211–224, Springer, 2001.
2. N. Beldiceanu, M. Carlsson, and J.X. Rampon. Global constraint catalog. Technical Report T2005:08, SICS, 2005.
3. N. Beldiceanu and E. Contejean. Introducing global constraints in chip. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.
4. N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the *same* and *usedby* constraints. In *MPI Technical Report MPI-I-2004-1-001*, 2004.
5. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. The Range and Roots constraints: Specifying counting and occurrence problems. In *Proc. of IJCAI'05*, pp 60–65, 2005.
6. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh, 'Among, Common and Disjoint Constraints, to appear in LNAI of Springer.
7. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. The Range constraint: Algorithms and implementation, to appear in *Proc. of CPAIOR'06*.
8. 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*, 4:167–192, 1999.
9. B. Hnich, Z. Kiziltan and T. Walsh. Modelling a Balanced Academic Curriculum Problem. In *Proc. of CPAIOR'02*, pp. 121–131, 2002.
10. F. Laburthe. Choco: implementing a CP kernel. In *Proc. of CP'00 Workshop TRICS: Techniques foR Implementing Constraint programming Systems*, 2000.
11. C.G. Quimper, P. van Beek, A. Lopez-Ortiz, A. Golynski and S.B. Sadjad. An efficient bounds consistency algorithm for the global cardinality constraint. In *Proc. of CP'03*, Springer, 2003.
12. P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Proc. of CP'00*, pp. 369–383, Springer, 2000.
13. J.C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. of AAAI'94*, pp 362–367, 1994.
14. J.C. Régin. Generalized arc consistency for global cardinality constraint. In *Proc. of AAAI'96*, pp. 209–215, 1996.
15. C. Schulte and P.J. Stuckey. Speeding up constraint propagation. In *Proc. of CP'04*, pp. 619–633, Springer, 2004.
16. P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.