The Range Constraint: Algorithms and Implementation

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

[1] LIRMM, CNRS/University of Montpellier, France, bessiere@lirmm.fr
[2] NICTA and UNSW, Sydney, Australia, {ehebrard, tw}@cse.unsw.edu.au
[3] Izmir University of Economics, Izmir, Turkey, brahim.hnich@ieu.edu.tr
[4] University of Bologna, Italy, zkiziltan@deis.unibo.it

Abstract. We recently 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 mapping onto particular values. In order for this specification language to be executable, propagation algorithms for the Range and Roots constraints should be developed. In this paper, we focus on the study of the Range constraint. We propose an efficient algorithm for propagating the Range constraint. We also show that decomposing global counting and occurrence constraints using Range is effective and efficient in practice.

1 Introduction

Constraints that put restrictions on the occurrence of particular values (*occurrence* constraints) or constraints that put restrictions on the number of values or variables meeting some conditions (*counting* constraints) are very useful in many real world problems, especially those involving resources. For instance, we may want to limit the number of distinct values assigned to a set of variables. Many of the global constraints proposed in the past are counting and occurrence constraints (see, for example, [1–5]). In [6], we show that many occurrence and counting constraints can be expressed by means of two new global constraints, Range and Roots, together with some classical elementary constraints. This language also provides us with a method to propagate counting and occurrence constraints. We just need to provide efficient propagation algorithms for the Range and Roots constraints. This paper focuses on the Range constraint. We give an efficient algorithm for propagating the Range constraint based on a flow algorithm. We propose an extension of the Range constraint where we have constraints on the cardinality of the set variables. We also show that decomposing occurence constraints and counting constraints using the Range constraint performs well in practice.

The rest of the paper is organised as follows. Section 2 gives the formal background. Section 3 shows how counting and occurrence constraints can be decomposed using the RANGE constraint. In Section 4, we propose a polynomial algorithm for the RANGE constraint and an extension to the case where the set variables are subject to constraints on their cardinality. Some experimental results are presented in Section 6. Finally, we conclude in Section 7.

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. 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 (that must belong to the set) and an upper bound $ub(S)$ which also contains the potential elements (that may or may not belong to the set).

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 an *hybrid support.* If C involves only integer variables, a hybrid support is a *support.* A value for an integer variable (resp. set variable) is *bound* or *hybrid consistent with* C iff there exists a bound or hybrid support assigning this value (resp. set of values) to this variable. 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 different 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 (value 2 was considered as possible for X_1 because BC deals with bounds). 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.

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 $\mathcal{F}$. Throughout, we will view a set of variables, X_1 to X_n as a function $\mathcal{X} : \{1,..,n\} \longrightarrow \bigcup_{i=1}^{i=n} D(X_i)$. That is, $\mathcal{X}(i)$ is the value of X_i.

3 An Executable Language

One of the simplest ways to propagate a new constraint is to decompose it into existing primitive constraints. We can then use the propagation algorithms associated with these primitives. Of course, such decomposition may reduce the number of domain values pruned. In [6], we show that many global 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 $|S| = k$). Adding Range and Roots and their propagation algorithms to a constraint toolkit thus provides a simple executable language for specifying a wide range of counting and occurrence constraints.

We focus here on the Range constraint. Given the 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.

$$\textsc{Range}([X_1, .., X_n], S, T) \text{ iff } T = \bigcup_{i \in S} \mathcal{X}(i)$$

In an associated report, we present a catalog containing over 70 global constraints from [7] specified with this simple language containing Range and Roots constraints. We present here just two examples of using Range to decompose a global constraint. 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 [8]. $\textsc{NValue}([X_1, .., X_n], N)$ holds iff $N = |\{X_i \mid 1 \leq i \leq n\}|$. This can be decomposed into a Range constraint:

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

Enforcing HC on the decomposition is weaker than GAC on the original NValue constraint. However, it is NP-hard to enforce GAC on a NValue constraint [9].

In [6], the Uses constraint was introduced. Uses is a variant of the UsedBy constraint [5]. $\textsc{Uses}([X_1, .., X_n], [Y_1, .., Y_m])$ holds iff the set of values assigned to $Y_1, .., Y_m$ is a subset of the set of values assigned to $X_1, .., X_n$. This can be decomposed into a Range constraint:

$$\textsc{Uses}([X_1, .., X_n], [Y_1, .., Y_m]) \text{ iff}$$

$$\textsc{Range}([X_1, .., X_n], \{1, .., n\}, T) \ \wedge \textsc{Range}([Y_1, .., Y_m], \{1, .., m\}, T') \ \wedge \ T' \subseteq T$$

Enforcing HC on the decomposition is weaker than GAC on the original USES constraint. However, it is NP-hard to enforce GAC on a USES constraint [6]. Thus, decomposition is a simple method to obtain a polynomial propagation algorithm.

The PERMUTATION constraint is an ALLDIFFERENT constraint where we additionally know $\Re$, the set of values to be taken. That is, the sequence of variables is a permutation of the values in $\Re$ where $|\Re| = n$. In [6], the PERMUTATION constraint is decomposed using a single RANGE constraint:

$$\begin{array}{l}\text{PERMUTATION}([X_1, \ldots, X_n], \Re) \ \text{ iff} \\ \text{RANGE}([X_1, \ldots, X_n], \{1, \ldots, n\}, \Re)\end{array}$$

This decomposition of the PERMUTATION constraint into a single RANGE constraint does not hinder constraint propagation [6].

4 Propagating the Range Constraint

Enforcing hybrid consistency on the RANGE constraint is polynomial. This can be done using a maximum network flow problem. In fact, the RANGE constraint can be decomposed using a global cardinality constraint (GCC) for which propagators based on flow problems already exist [3, 10]. This will be shown in Section 5. But the RANGE constraint does not need the whole power of maximum network flow problems, and thus HC can be enforced on it at a lower cost than that of calling a GCC propagator. In this section, we propose an efficient way to enforce HC on RANGE. To simplify the presentation, the use of the flow is limited to a constraint that performs only part of the work needed for enforcing HC on RANGE. This constraint, that we name OCCURS$([X_1, \ldots, X_n], T)$, ensures that all the values in the set variable T are used by the integer variables X_1 to X_n:

$$\text{OCCURS}([X_1, \ldots, X_n], T) \text{ iff } T \subseteq \bigcup_{i \in 1..n} \mathcal{X}(i)$$

We first present an algorithm for achieving HC on OCCURS (Section 4.1), and then use this to propagate the RANGE constraint (Section 4.2).

4.1 Occurs Constraint

We achieve hybrid consistency on OCCURS$([X_1, \ldots, X_n], T)$ using a network flow. We use a unit capacity network [11] in which capacities between two nodes can only be 0 or 1. This is represented by a directed graph where an arc from node x to node y means that a maximum flow of 1 is allowed between x and y while the absence of an arc means that the maximum flow allowed is 0. The unit capacity network $G_C = (N, E)$ of the constraint $C =$ OCCURS$([X_1, \ldots, X_n], T)$ is built in the following way. $N = \{s\} \cup N_1 \cup N_2 \cup \{t\}$, where s is a source node, t is a sink node, $N_1 = \{v \mid v \in \bigcup D(X_i)\}$ and $N_2 = \{z_v \mid v \in \bigcup D(X_i)\} \cup \{x_i \mid i \in [1..n]\}$. The set of arcs E is as follows:

$$E = (\{s\} \times N_1) \cup \{(v, z_v), \forall v \notin lb(T)\} \cup \{(v, x_i) \mid v \in D(X_i)\} \cup (N_2 \times \{t\})$$

G_C is quadripartite, i.e., $E \subseteq (\{s\} \times N_1) \cup (N_1 \times N_2) \cup (N_2 \times \{t\})$. In Fig. 1, we depict the network G_C of the constraint $C = \text{Occurs}([X_1, X_2, X_3], T)$ with $D(X_1) = \{1, 2\}$, $D(X_2) = \{2, 3, 4\}$, $D(X_3) = \{3, 4\}$, $lb(T) = \{3, 4\}$ and $ub(T) = \{1, 2, 3, 4\}$. The intuition behind this graph is that when a flow uses an arc from a node v to a node x_i this means that X_i is assigned v, and when a flow uses the arc (v, z_v) this means that v is not necessarily used by the X_i's.[5] In Fig. 1 nodes 3 and 4 are linked only to nodes x_2 and x_3, which means that values 3 and 4 must necessarily be taken by one of the variables X_i (3 and 4 belong to $lb(T)$). On the contrary, nodes 1 and 2 are also linked to nodes z_1 and z_2 because values 1 and 2 do not have to be taken by a X_i (they are not in $lb(T)$).

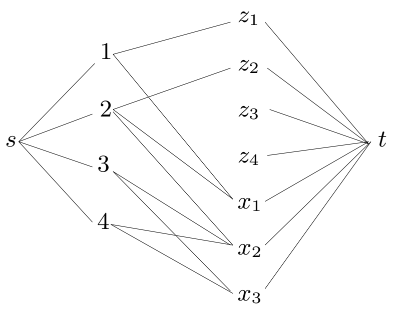


Fig. 1. Unit capacity network of the constraint $C = \text{Occurs}([X_1, X_2, X_3], T)$ with $D(X_1) = \{1, 2\}$, $D(X_2) = \{2, 3, 4\}$, $D(X_3) = \{3, 4\}$, $lb(T) = \{3, 4\}$ and $ub(T) = \{1, 2, 3, 4\}$. Arcs are directed from left to right.

In the particular case of unit capacity networks, a flow is any set $E' \subseteq E$: any arc in E' is assigned 1 and the arcs in $E \setminus E'$ are assigned 0. A *feasible* flow from s to t in G_C is a subset E_f of E such that $\forall n \in N \setminus \{s, t\}$ the number of arcs of E_f entering n is equal to the number of arcs of E_f going out of n, that is, $|\{(n', n) \in E_f\}| = |\{(n, n'') \in E_f\}|$. The value of the flow E_f from s to t, denoted $val(E_f, s, t)$, is $val(E_f, s, t) = |\{n \mid (s, n) \in E_f\}|$. A *maximum* flow from s to t in G_C is a feasible flow E_M such that there does not exist a feasible flow E_f, with $val(E_f, s, t) > val(E_M, s, t)$. A maximum flow for the network of Fig. 1 is given in Fig. 2. By construction a feasible flow cannot have a value greater than $|N_1|$. In addition, a feasible flow cannot contain two arcs entering

[5] Note that the edges go from the nodes representing the values to the nodes representing the variables. This is the opposite as the direction in the network flow problems used in the propagators of the Alldiff or Gcc constraints [1, 3].

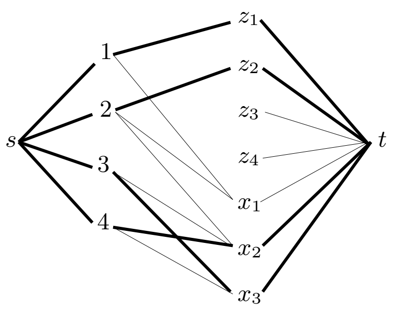


Fig. 2. A maximum flow for the network of Fig. 1. Bold arcs are those that belong to the flow. Arcs are directed from left to right.

a node x_i from N_2. Hence, we can define a function φ linking feasible flows and partial instantiations on the X_i's. Given any feasible flow E_f from s to t in G_C, $\varphi(E_f) = \{(X_i, v) \mid (v, x_i) \in E_f\}$. The maximum flow in Fig. 2 corresponds to the instantiation $X_2 = 4, X_3 = 3$. The way G_C is built induces the following:

Theorem 1. *Let $G_C = (N, E)$ be the capacity network of a constraint $C =$ Occurs$([X_1, \dots, X_n], T)$.*

1. *A value v in the domain $D(X_i)$ for some $i \in [1..n]$ is HC iff there exists a flow E_f from s to t in G_C with $val(E_f, s, t) = |N_1|$ and $(v, x_i) \in E_f$*
2. *If the X_i's are HC, T is HC iff $ub(T) \subseteq \bigcup_i D(X_i)$*

Proof. (1.$\Rightarrow$) Let I be a solution for C with $(X_i, v) \in I$. Build the following flow H: Put (v, x_i) in H; $\forall w \in I[T], w \neq v$, take a variable X_j such that $(X_j, w) \in I$ (we know there is at least one since I is solution), and put (w, x_j) in H; $\forall w' \notin I[T], w' \neq v$, add $(w', z_{w'})$ to H. Add to H the edges from s to N_1 and from N_2 to t so that we obtain a feasible flow. By construction, all $w \in N_1$ belong to an edge of H. So, $val(H, s, t) = |N_1|$ and H is a maximum flow with $(v, x_i) \in H$.

(1.$\Leftarrow$) Let E_M be a flow from s to t in G_C with $(v, x_i) \in E_M$ and $val(E_M, s, t) = |N_1|$. By construction of G_C, we are guaranteed that all nodes in N_1 belong to an arc in $E_M \cap (N_1 \times N_2)$, and that for every value $w \in lb(T)$, $\{n \mid (w, n) \in E\} \subseteq \{x_i \mid i \in [1..n]\}$. Thus, for each $w \in lb(T), \exists X_j \mid (X_j, w) \in \varphi(E_M)$. Hence, any extension of $\varphi(E_M)$ where each unassigned X_j takes any value in $D(X_j)$ and $T = lb(T)$ is a solution of C with $X_i = v$.

(2.$\Rightarrow$) If T is HC, all values in $ub(T)$ appear in at least one solution tuple. Since C ensures that $T \subseteq \bigcup_i \{X_i\}$, $ub(T)$ cannot contain a value appearing in none of the $D(X_i)$.

(2.$\Leftarrow$) Let $ub(T) \subseteq \bigcup_i D(X_i)$. Since all X_i's are HC, we know that each value v in $\bigcup_i D(X_i)$ is taken by some X_i in at least one solution tuple I. Build the tuple I' so that $I'[X_i] = I[X_i]$ for each $i \in [1..n]$ and $I'[T] = I[T] \cup \{v\}$. I' is still solution of C. So, $ub(T)$ is as tight as it can be wrt HC. In addition, since all X_i's are HC, this means that in every solution tuple I, for each $v \in lb(T)$ there exists i such that $I[X_i] = v$. So, $lb(T)$ is HC. □

Following Theorem 1, we need a way to check which edges belong to a maximum flow. *Residual graphs* are useful for this task. Given a unit capacity network G_C and a maximal flow E_M from s to t in G_C, the residual graph $R_{G_C}(E_M) = (N, E_R)$ is the directed graph obtained from G_C by reversing all arcs belonging to the maximum flow E_M; that is, $E_R = \{(x,y) \in E \setminus E_M\} \cup \{(y,x) \mid (x,y) \in E \cap E_M\}$. Given the network G_C of Fig. 1 and the maximum flow E_M of Fig. 2, $R_{G_C}(E_M)$ is depicted in Fig. 3. Given a maximum flow E_M from s to t in G_C, given $(x,y) \in N_1 \times N_2 \cap E \setminus E_M$, there exists a maximum flow containing (x,y) iff (x,y) belongs to a cycle in $R_{G_C}(E_M)$ [12]. Furthermore, finding all the arcs (x,y) that do not belong to a cycle in a graph can be performed by building the strongly connected components of the graph. We see in Fig. 3 that the arcs $(1, x_1)$ and $(2, x_1)$ belong to a cycle. So, they belong to some maximum flow and $(X_1, 1)$ and $(X_1, 2)$ are hybrid consistent. $(2, x_2)$ does not belong to any cycle. So, $(X_2, 2)$ is not HC.

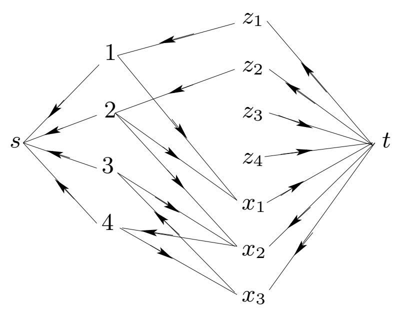


Fig. 3. Residual graph obtained from the network in Fig. 1 and the maximum flow in Fig. 2.

We now have all the tools for achieving HC on any OCCURS constraint. We first build G_C. We compute a maximum flow E_M from s to t in G_C; if $val(E_M, s, t) < |N_1|$, we fail. Otherwise we compute $R_{G_C}(E_M)$, build the strongly connected components in $R_{G_C}(E_M)$, and remove from $D(X_i)$ any value v such that (v, x_i) belongs to neither E_M nor to a strongly connected component in $R_{G_C}(E_M)$. Finally, we set $ub(T)$ to $ub(T) \cap \bigcup_i D(X_i)$. Following Theorem 1 and properties of residual graphs, this algorithm enforces HC on OCCURS$([X_1, .., X_n], T)$.

Building G_C is in $O(nd)$. We need then to find a maximum flow E_M in G_C. This can be done in two sub-steps. First, we use the arc (v, z_v) for each $v \notin lb(T)$ (in $O(|\bigcup_i D(X_i)|)$). Afterwards, we compute a maximum flow on the subgraph composed of all paths traversing nodes w with $w \in lb(T)$ (because there is no arc (w, z_w) in G_C for such w). The complexity of finding a maximum flow in a unit capacity network is in $O(\sqrt{k} \cdot e)$ if k is the number of nodes and e the number of

Algorithm 1: `Hybrid consistency on Range`

```
procedure Propag-Range($[X_1, \dots, X_n], S, T$);
1  Introduce the set of integer variables $Y = \{Y_i \mid i \in ub(S)\}$,
   with $D(Y_i) = D(X_i) \cup \{dummy\}$;
2  Achieve hybrid consistency on the constraint OCCURS($Y, T$);
3  Achieve hybrid consistency on the constraints $i \in S \leftrightarrow Y_i \in T$, for all $Y_i \in Y$;
4  Achieve GAC on the constraints $(Y_i = dummy) \vee (Y_i = X_i)$, for all $Y_i \in Y$;
```

edges. This gives a complexity in $O(\sqrt{|lb(T)|} \cdot n \cdot |lb(T)|)$ for this second sub-step. Building the residual graph and computing the strongly connected components is in $O(nd)$. Extracting the HC domains for the X_i's is direct. There remains to compute BC on T, which takes $O(nd)$. Therefore, the total complexity is in $O(nd + n \cdot |lb(T)|^{3/2})$.

4.2 Hybrid Consistency on Range

Enforcing HC on RANGE($[X_1, \dots, X_n], S, T$) can be done by decomposing it as an OCCURS constraint on new variables Y_i and some channelling constraints ([13]) linking T and the Y_i's to S and the X_i's. But the interesting point is that we do not need to *maintain* HC on the decomposition but we just need to propagate the constraints in *one pass*.

The algorithm *Propag-Range*, enforcing HC on the RANGE constraint, is presented in Algorithm 1. In line 1, a special encoding is built, where a Y_i is introduced for each X_i with index in $ub(S)$. The domain of a Y_i is the same as that of X_i plus a dummy value. The dummy value works as a flag. If OCCURS prunes it from $D(Y_i)$ this means that Y_i is necessary in OCCURS to cover $lb(T)$. Then, X_i is also necessary to cover $lb(T)$ in RANGE. In line 2, HC on OCCURS removes a value from a Y_i each time it contains other values that are necessary to cover $lb(T)$ in every solution tuple. HC also removes values from $ub(T)$ that cannot be covered by any Y_i in a solution. Line 3 updates the bounds of S and the domain of Y_i's. Finally, in line 4, the channelling constraints between Y_i and X_i propagate removals on X_i for each i which belongs to S in all solutions.

Theorem 2. *The algorithm Propag-Range is a correct algorithm for enforcing HC on* RANGE, *that runs in* $O(nd + n \cdot |lb(T)|^{3/2})$ *time, where d is the maximal size of* X_i *domains.*

Proof. Soundness. A value v is removed from $D(X_i)$ in line 4 if it is removed from Y_i together with *dummy* in lines 2 or 3. If a value v is removed from Y_i in line 2, this means that any tuple on variables in Y covering $lb(T)$ requires that Y_i takes a value from $D(Y_i)$ other than v. So, we cannot find a solution of RANGE in which $X_i = v$ since $lb(T)$ must be covered as well. A value v is removed from $D(Y_i)$ in line 3 if $i \in lb(S)$ and $v \notin ub(T)$. In this case, RANGE cannot be satisfied by a tuple where $X_i = v$. If a value v is removed from $ub(T)$ in line 2, none of the tuples of values for variables in Y covering $lb(T)$ can cover

v as well. Since variables in Y duplicate variables X_i with index in $ub(S)$, there is no hope to satisfy RANGE if v is in T. Note that $ub(T)$ cannot be modified in line 3 since Y contains only variables Y_i for which i was in $ub(S)$. If a value v is added to $lb(T)$ in line 3, this is because there exists i in $lb(S)$ such that $D(Y_i) \cap ub(T) = \{v\}$. Hence, v is necessarily in T in all solutions of RANGE. An index i can be removed from $ub(S)$ only in line 3. This happens when the domain of Y_i does not intersect $ub(T)$. In such a case, this is evident that a tuple where $i \in S$ could not satisfy RANGE since X_i could not take a value in T. Finally, if an index i is added to $lb(S)$ in line 3, this is because $D(Y_i)$ is included in $lb(T)$, which means that the dummy value has been removed from $D(Y_i)$ in line 2. This means that Y_i takes a value from $lb(T)$ in all solutions of OCCURS. X_i also has to take a value from $lb(T)$ in all solutions of RANGE.
Completeness (Sketch). Suppose that a value v is not pruned from $D(X_i)$ after line 4 of *Propag-Range*. If $Y_i \in Y$, we know that after line 2 there was an instantiation I on Y and T, solution of OCCURS with $I[Y_i] = v$ or with $Y_i = dummy$ (thanks to the channelling constraints in line 4). We can build the tuple I' on $X_1, ..X_n, S, T$ where X_i takes value v, every X_j with $j \in ub(S)$ and $I[Y_j] \in I[T]$ takes $I[Y_j]$, and the remaining X_j's take any value in their domain. T is set to $I[T]$ plus the values taken by X_j's with $j \in lb(S)$. These values are in $ub(T)$ thanks to line 3. Finally, S is set to $lb(S)$ plus the indices of the Y_j's with $I[Y_j] \in I[T]$. These indices are in $ub(S)$ since the only j's removed from $ub(S)$ in line 3 are such that $D(Y_j) \cap ub(T) = \emptyset$, which prevents $I[Y_j]$ from taking a value in $I[T]$. Thus I' is a solution of RANGE with $I'[X_i] = v$. We have proved that the X_i's are hybrid consistent after *Propag-Range*.

Suppose a value $i \in ub(S)$ after line 4. Thanks to constraint in line 3 we know there exists v in $D(Y_i) \cap ub(T)$, and so, $v \in D(X_i) \cap ub(T)$. Now, X_i is hybrid consistent after line 4. Thus $X_i = v$ belongs to a solution of RANGE. If we modify this solution by putting i in S and v in T (if not already there), we keep a solution.

Completeness on $lb(S)$, $lb(T)$ and $ub(T)$ is proved in a similar way.
Complexity. The important thing to notice in *Propag-Range* is that constraints in lines 2–4 are propagated in sequence. Thus, OCCURS is propagated only once, for a complexity in $O(nd + n \cdot |lb(T)|^{3/2})$. Lines 1, 3, and 4 are in $O(nd)$. Thus, the complexity of *Propag-Range* is in $O(nd+n \cdot |lb(T)|^{3/2})$. This reduces to linear time complexity when $lb(T)$ is empty. □

As we will show in the next section, the Range constraint can be decomposed using the GCC constraint. However, propagation on such a decomposition is in $O(n^2d + n^{2.66})$ time complexity (see [10]). *Propag-Range* is thus significantly cheaper.

5 Range and Cardinality

Constraint toolkits like [14] additionally represent an interval on the cardinality of each set variable. This extra information is not taken into account by RANGE$([X_1, \ldots, X_n], S, T)$ whereas it could improve propagation. We can easily extend the RANGE constraint to a constraint RANGE-CARD that involves this cardinality information. RANGE-CARD$([X_1, \ldots, X_n], S, M, T, N)$ holds iff RANGE$([X_1, \ldots, X_n], S, T)$ & $|S| = M$ & $|T| = N$. Unfortunately, enforcing HC on RANGE-CARD is NP-hard because it subsumes the NVALUE constraint (RANGE-CARD$([X_1, \ldots, X_n], \{1..n\}, n, T, N) \equiv$ NVALUE$(N, [X_1, \ldots, X_n])$) and NVALUE is itself NP-hard to propagate [9]. However, we can partially take into account such cardinality information. RANGE-CARD$([X_1, \ldots, X_n], S, M, T, N)$ can be decomposed using a GCC constraint:

RANGE-CARD$([X_1, \ldots, X_n], S, M, T, N)$ iff

$$\begin{array}{lll} & \textsc{Gcc}([Y_1, \ldots, Y_n], [1, \ldots, m+1], [B_1, \ldots, B_{m+1}]) & \wedge \\ \forall i \in [1..n] & i \in S \leftrightarrow Y_i \in T & \wedge \\ \forall i \in [1..n] & (X_i = Y_i) \vee (Y_i = m+1) & \wedge \\ \forall v \in [1..m+1] & v \in T \leftrightarrow B_v \neq 0 & \wedge \\ \forall v \in [1..m+1] & B_v \leq M - N + 1 & \wedge \\ & \sum_{v \in [1..m]} B_v = N & \end{array}$$

where $m = |\bigcup_{i \in [1..n]}(D(X_i))|$, $m+1$ is a dummy value, and GCC is the global cardinality constraint [3]. A GCC$([X_1, \ldots, X_n], [d_1, \ldots, d_m], [O_1, \ldots, O_m])$ constraint holds iff the value d_i is used O_i times in $X_1, \ldots, X_n$.

For sake of clarity we suppose that values are consecutive in the interval $[1..m]$ but this is not a restriction.

We have $\forall i \in [1..n],\ D(Y_i) = D(X_i) \cup \{m+1\}$ and $\forall v \in [1..m+1],\ D(B_v) = [0..n]$. We enforce GAC on the X's and Y's and BC on S, T and the B's. This algorithm has $O(n^2 d + n^{2.66})$ complexity (see [10]), which is typically worse than *Propag-Range* which ignores such cardinality information. It remains an open problem if we can extend *Propag-Range* to include some cardinality information, and if we can do so without changing its complexity.

6 Experimental Results

The purpose of this section is twofold. We demonstrate that decomposing global counting and occurrence constraints using RANGE is effective and efficient in practice. We show that propagating RANGE using the algorithm introduced in this paper is more effective than propagating it using the straightforward decomposition:

$$\textsc{Range}([X_1, \ldots, X_n], S, T) \text{ iff}$$

$$i \in S \rightarrow X_i \in T \ \wedge \ j \in T \rightarrow \exists i \in S . X_i = j \qquad (1)$$

In order to isolate the effect of the RANGE constraint from other modelling issues, we used the following protocol: we randomly generated instances of binary CSPs and we added USES$([X_1, .., X_n], [Y_1, .., Y_n])$ constraints. Note that, it is NP-hard to achieve GAC on USES and there is no propagator available for this constraint in the literature. So, in all our experiments, we encode USES in three different ways:

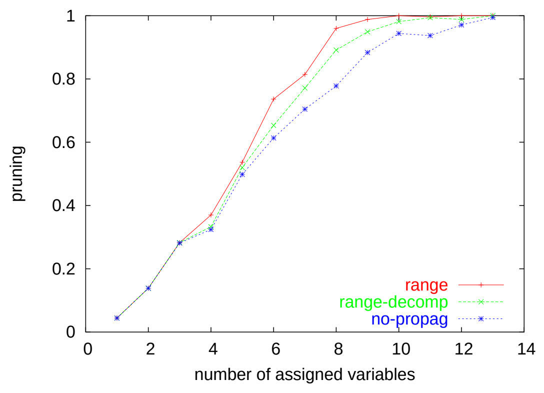


Fig. 4. Propagating random binary constraint satisfaction problems with three overlapping USES constraints (class A).

`[no-propag]`**:** by putting the USES constraint in the model with no propagator but just a checker testing if it is satisfied or not,

`[range]`**:** by decomposing USES using RANGE as described in Section 3 and using the algorithm *Propag-Range* presented in Section 4,

`[range-decomp]`**:** by decomposing the RANGE constraints of the previous model using primitive constraints as in decomposition (1).

The problem instances are generated according to model B in [15], and can be described with the following parameters: the number of X and Y variables nx and ny in USES constraints, the total number of variables nz, the domain size d, the number of binary constraint m_1, the number of forbidden tuples t per binary constraint, and the number of USES constraints m_2. Note that the USES constraints can have overlapping or disjoint scopes of variables. We distinguish the two cases. All reported results are averages on 100 instances.

Our first experiment shows the effectiveness of decomposing USES with RANGE for propagation *alone* (not solving). We compared the number of values removed by propagation on the models obtained by representing USES constraints in the three different ways: `no-propag`, `range`, and `range-decomp`. (Note that in the `no-propag` model, the values are pruned only because of the binary constraints.) To simulate what happens inside a backtrack search, we randomly selected a subset of the variables and randomly assigned them values before propagation. Hence, in the experiments, the constraints are exposed to a wide range of different variable domains. We report the ratio of values removed by propagation on the following classes of problems:

$$class\ A : \langle nx = 5, ny = 10, nz = 35, d = 20, m_1 = 70, t = 150, m_2 = 3\ (overlap)\rangle$$
$$class\ B : \langle nx = 5, ny = 10, nz = 45, d = 20, m_1 = 90, t = 150, m_2 = 3\ (disjoint)\rangle$$

in which the number of assigned variables varies between 1 and 15.

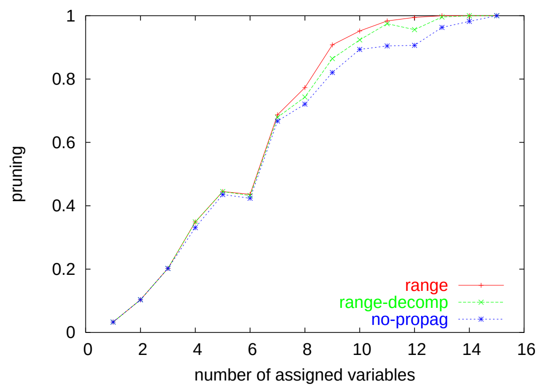


Fig. 5. Propagating random binary constraint satisfaction problems with three disjoint USES constraints (class B).

We observe in Figures 4 and 5 that domains can be reduced significantly using RANGE when propagating problems containing USES constraints. We also observe that propagating the RANGE constraint directly (`range` model) is more effective than propagating its decomposition (`range-decomp` model). The differences are greater when the USES constraints of the original problem overlap (Fig. 4) than when they are all disjoint (Fig. 5).

Our second experiment shows the efficiency of decomposing USES with RANGE when *solving* the problems. Our solver used the *smallest-domain-first* variable ordering heuristic with the lexicographical value ordering and a cutoff at 600 seconds. We compared the cost of solving the three types of models: `no-propag`, `range`, and `range-decomp`. We report the number of fails and the cpu-time needed to find the first solution on the following classes of problems:

$$\textit{class } C : \langle nx = 5, ny = 10, nz = 25, d = 10, m_1 = 40, t, m_2 = 2\rangle$$
$$\textit{class } D : \langle nx = 5, ny = 10, nz = 30, d = 10, m_1 = 60, t, m_2 = 2\rangle$$

in which t varies between 30 and 80.

We observe in Figures 6 and 7 that using the decomposition of RANGE (`range-decomp` model) is costly. This is due to the disjunction in the implementation of $\exists$. Note that the instances solved here (classes C and D) are much smaller than those used for propagation (classes A and B). Solving larger instances was impractical. Note also that we do not present the results where the RANGE constraint is not used (`no-propag` model) because they reached the cutoff in most of the instances not trivially over-constrained. So, this second experiment shows how efficiently RANGE can solve problems containing USES constraints. It also shows the clear benefit of using our algorithm in preference to the decomposition of RANGE over the under-constrained region. As the problems get over-constrained, the binary constraints dominate the pruning, and the

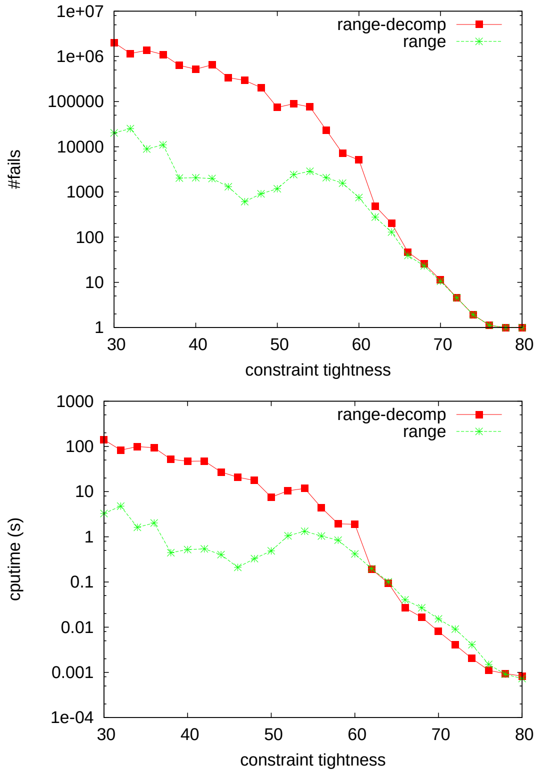
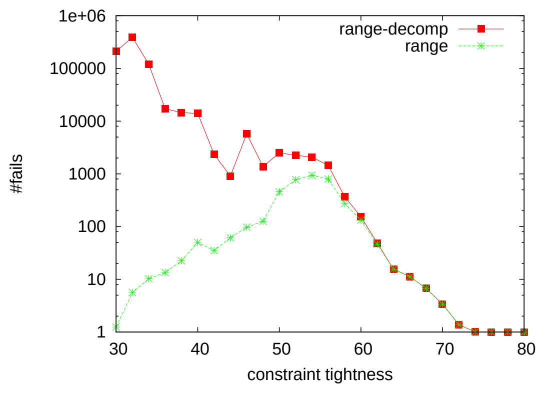


Fig. 6. Solving random binary constraint satisfaction problems with two overlapping Uses constraints (class C).

algorithm gives a slight overhead in run-time, pruning equally with the decomposition of Range.

7 Conclusion

Range and Roots are two global constraints that can express many other global constraints, such as occurrence and counting constraints [6]. We have presented a comprehensive study of the Range constraint. We proposed an algorithm for enforcing hybrid consistency on Range. We proposed a way to partially propagate Range-Card, a constraint that combines Range with constraints on the cardinality of the set variables. Our experiments show the benefit we can obtain by incorporating the Range constraint in a constraint toolkit.



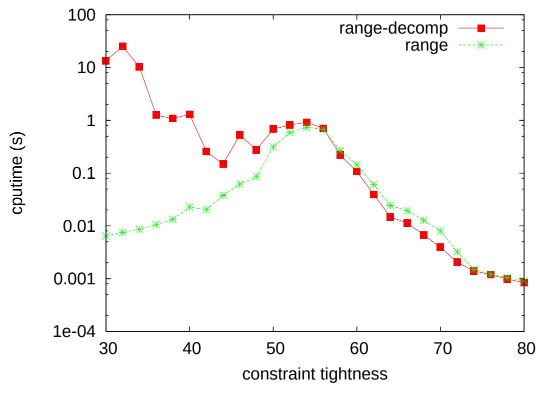


Fig. 7. Solving random binary constraint satisfaction problems with two disjoint Uses constraints (class D).

Acknowledgements

Hebrard and Walsh are supported by the National ICT Australia, which is funded through the Australian Government's *Backing Australias Ability* initiative, in part through the Australian Research Council. Hnich received support from Science Foundation Ireland (Grant 00/PI.1/C075).

References

1. Régin, J.: A filtering algorithm for constraints of difference in CSPs. In: Proceedings AAAI'94, Seattle WA (1994) 362–367
2. Beldiceanu, N., Contejean, E.: Introducing global constraints in chip. Mathl. Comput. Modelling **20** (1994) 97–123
3. Régin, J.: Generalized arc consistency for global cardinality constraint. In: Proceedings AAAI'96, Portland OR (1996) 209–215

4. Beldiceanu, N.: Pruning for the minimum constraint family and for the number of distinct values constraint family. In: Proceedings CP'01, Paphos, Cyprus (2001) 211–224
5. Beldiceanu, N., Katriel, I., Thiel, S.: Filtering algorithms for the *same* and *usedby* constraints. In: MPI Technical Report MPI-I-2004-1-001. (2004)
6. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: The Range and Roots constraints: Specifying counting and occurrence problems. In: Proceedings IJCAI'05, Edinburgh, Scotland (2005) 60–65
7. Beldiceanu, N., Carlsson, M., Rampon, J.: Global constraint catalog. Technical Report T2005:08, Swedish Institute of Computer Science, Kista, Sweden (2005)
8. Pachet, F., Roy, P.: Automatic generation of music programs. In: Proceedings CP'99, Alexandria VA (1999) 331–345
9. Bessiere, C., Hebrard, E., Hnich, B., Walsh, T.: The complexity of global constraints. In: Proceedings AAAI'04, San Jose CA (2004) 112–117 to appear.
10. Quimper, C., López-Ortiz, A., van Beek, P., Golynski, A.: Improved algorithms for the global cardinality constraint. In: Proceedings CP'04, Toronto, Canada (2004) 542–556
11. Ahuja, R., Magnanti, T., Orlin, J.: Network flows. Prentice Hall, Upper Saddle River NJ (1993)
12. Schrijver, A.: Combinatorial Optimization - Polyhedra and Efficiency. Springer-Verlag, Berlin (2003)
13. Cheng, B., Choi, K., Lee, J., Wu, J.: Increasing constraint propagation by redundant modeling: an experience report. Constraints **4** (1999) 167–192
14. ILOG: Reference and User Manual. ILOG Solver 5.3, ILOG S.A. (2002)
15. Prosser, P.: An empirical study of phase transition in binary constraint satisfaction problems. Artificial Intelligence **81** (1996) 81–109