The All Different and Global Cardinality Constraints on Set, Multiset and Tuple Variables

Claude-Guy Quimper[1] and Toby Walsh[2]

[1] School of Computer Science, University of Waterloo, Canada
cquimper@math.uwaterloo.ca
[2] NICTA and UNSW, Sydney, Australia
tw@cse.unsw.edu.au

Abstract. We describe how the propagator for the All-Different constraint can be generalized to prune variables whose domains are not just simple finite domains. We show, for example, how it can be used to propagate set variables, multiset variables and variables which represent tuples of values. We also describe how the propagator for the global cardinality constraint (which is a generalization of the All-Different constraint) can be generalized in a similar way. Experiments show that such propagators can be beneficial in practice, especially when the domains are large.

1 Introduction

Constraint programming has restricted itself largely to finding values for variables taken from given finite domains. However, we might want to consider variables whose values have more structure. We might, for instance, want to find a set of values for a variable [12, 13, 14, 15], a multiset of values for a variable [16], an ordered tuple of values for a variable, or a string of values for a variable. There are a number of reasons to want to enrich the type of values taken by a variable. First, we can reduce the space needed to represent possible domain values. For example, we can represent the exponential number of subsets for a set variable with just an upper and lower bound representing possible and definite elements in the set. Second, we can improve the efficiency of constraint propagators for such variables by exploiting the structure in the domain. For example, it might be sufficient to consider each of the possible elements in a set in turn, rather than the exponential number of subsets. Third, we inherit all the usual benefits of data abstraction like ease of debugging and code maintenance.

As an example, consider the round robin sports scheduling problem (prob026 in CSPLib). In this problem, we wish to find a game for each slot in the schedule. Each game is a pair of teams. There are a number of constraints that the schedule needs to satisfy including that all games are different from each other. We therefore would like a propagator which works on an All-Different constraint posted on variables whose values are pairs (binary tuples). In this paper, we consider how to implement such constraints efficiently and effectively. We show how two of the most important constraint propagators, those for the All-Different

B. Hnich et al. (Eds.): CSCLP 2005, LNAI 3978, pp. 1–13, 2006.
© Springer-Verlag Berlin Heidelberg 2006

and the global cardinality constraint (*gcc*) can be extended to deal with variables whose values are sets, multisets or tuples.

2 Propagators for the All-Different Constraint

Propagating the All-Different constraint consists of detecting the values in the variable domains that cannot be part of an assignment satisfying the constraint. To design his propagator, Leconte [18] introduced the concept of *Hall set* based on Hall's work [1].

Definition 1. *A* Hall set *is a set H of values such that the number of variables whose domain is contained in H is equal to the cardinality of H. More formally, H is a Hall set if and only if $|H| = |\{x_i \mid dom(x_i) \subseteq H\}|$.*

Consider the following example.

Example 1. Let $\mathrm{dom}(x_1) = \{3, 4\}$, $\mathrm{dom}(x_2) = \{3, 4\}$, and $\mathrm{dom}(x_3) = \{2, 4, 5\}$ be three variable domains subject to an All-Different constraint. The set $H = \{3, 4\}$ is a Hall set since it contains two elements and the two variable domains $\mathrm{dom}(x_1)$ and $\mathrm{dom}(x_2)$ are contained in H.

In Example 1, variables x_1 and x_2 must be assigned to values 3 and 4, making these two values unavailable for other variables. Therefore, value 4 should be removed from the domain of x_3.

To enforce domain consistency, it is necessary and sufficient to detect every Hall set H and remove its values from the domains that are not fully contained in H. This is exactly what Régin's propagator [4] does using matching theory to detect Hall sets. Leconte [18], Puget [20], López-Ortiz et al. [19] use simpler ways to detect Hall intervals in order to achieve weaker consistencies.

3 Beyond Integer Variables

A propagator designed for integer variables can be applied to any type of variable whose domain can be enumerated. For instance, let the following variables be sets whose domains are expressed by a set of required values and a set of allowed values.

$$\{\} \subseteq S_1, S_2, S_3, S_4 \subseteq \{1, 2\} \text{ and } \{\} \subseteq S_5, S_6 \subseteq \{2, 3\}$$

Variable domains can be expanded as follows:

$$S_1, S_2, S_3, S_4 \in \{\{\}, \{1\}, \{2\}, \{1, 2\}\} \text{ and } S_5, S_6 \in \{\{\}, \{2\}, \{3\}, \{2, 3\}\}$$

And then by enforcing GAC on the All-Different constraint, we obtain

$$S_1, S_2, S_3, S_4 \in \{\{\}, \{1\}, \{2\}, \{1, 2\}\} \text{ and } S_5, S_6 \in \{\{3\}, \{2, 3\}\}$$

We can now convert the domains back to their initial representation.

$$\{\} \subseteq S_1, S_2, S_3, S_4 \subseteq \{1, 2\} \text{ and } \{3\} \subseteq S_5, S_6 \subseteq \{2, 3\}$$

This technique always works but is not tractable in general since variable domains might have exponential size. For instance, the domain of $\{\} \subseteq S_i \subseteq \{1, \ldots, n\}$ contains 2^n elements. The following important lemma allows us to ignore such variables and focus just on those with "small" domains.

Lemma 1. *Let n be the number of variables and let F be a set of variables whose domains are not contained in any Hall set. Let $x_i \notin F$ be a variable whose domain contains more than $n - |F|$ values. Then $dom(x_i)$ is not contained in any Hall set.*

Proof. The largest Hall set can contain the domain of $n - |F|$ variables and therefore has at most $n - |F|$ values. If $|\text{dom}(x_i)| > n - |F|$, then $\text{dom}(x_i)$ cannot be contained in any Hall set. □

Using Lemma 1, we can iterate through the variables and append to a set F those whose domain cannot be contained in a Hall set. A propagator for the All-Different constraint can prune the domains not in F and find all Hall sets. Values in Hall sets can then be removed from the variable domains in F. This technique ensures that domains larger than n do not slow down the propagation. Algorithm 1 exhibits the process for a set of (possibly non-integer) variables X.

Algorithm 1. All-Different propagator for variables with large domains

$F \leftarrow \emptyset$
Sort variables such that $|\text{dom}(x_i)| \geq |\text{dom}(x_{i+1})|$
for $x_i \in X$ **do**
1 ⌊ **if** $|dom(x_i)| > n - |F|$ **then** $F \leftarrow F \cup \{x_i\}$
2 Expand domains of variables in $X - F$.
Find values H belonging to a Hall set and propagate the All-Different constraint on variables $X - F$.
for $x_i \in F$ **do**
⌊ $\text{dom}(x_i) \leftarrow \text{dom}(x_i) - H$;
3 Collapse domains of variables in $X - F$.

To apply our new techniques, three conditions must be satisfied by the representation of the variables:

1. Computing the size of the domain must be tractable (Line 1).
2. Domains must be efficiently enumerable (Line 2).
3. Domains must be efficiently computed from an enumeration of values (Line 3).

The next sections describe how different representations of domains for set, multiset and tuple variables can meet these three conditions.

4 All-Different on Sets

Several representations of domains have been suggested for set variables. We show how their cardinality can be computed and their domain enumerated efficiently. One of the most common representations for a set are the required elements lb and the allowed elements ub, with any set S satisfying $lb \subseteq S \subseteq ub$ belongs to the domain $[12, 14]$. The number of sets in the domain is given by $2^{|ub-lb|}$. We can enumerate all these sets simply by enumerating all subsets of $ub - lb$ and adding them to the elements from lb. A set can be represented as a binary vector where each element is associated to a bit. A bit equals 1 if its corresponding element is in the set and equals 0 if its corresponding element is not in the set. Enumerating all subsets of $ub - lb$ is reduced to the problem of enumerating all binary vectors between 0 and $2^{|ub-lb|}$ exclusively which can be done in $O(2^{|ub-lb|})$ steps, i.e. $O(|\text{dom}(S_i)|)$ steps.

In order to exclude from the domain undesired sets, one can also add a cardinality variable [3]. The domain of a set variable is therefore expressed by $\text{dom}(S_i) = \{S \mid lb \subseteq S \subseteq ub, |S| \in \text{dom}(C)\}$ where C is an integer variable. We assume that C is consistent with lb and ub, i.e. $\min(C) >= |lb|$ and $\max(C) <= |ub|$. The size of the domain is given by Equation 1 where $\binom{a}{b}$ is the binomial coefficient.

$$|\text{dom}(S_i)| = \sum_{j \in C} \binom{|ub - lb|}{j - |lb|} \tag{1}$$

The binomial coefficients can efficiently be computed as explained in Chapter 6.1 of [10]. The identity $\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$ can be particularly useful to compute the summation when the domain of C is an interval. The number of steps required to compute $|\text{dom}(S_i)|$ is bounded by $O(|\text{dom}(C)|)$.

Algorithm 2 enumerates all combinations of t elements chosen from elements 0 to $n - 1$. Each element i in a combination is mapped to the i^{th} element in $ub - lb$. By enumerating all t-combinations for $t \in \text{dom}(C)$ to which we add the required elements lb, we enumerate all sets in $|\text{dom}(S_i)|$. Algorithm 2 has a time complexity of $O(t + \binom{n}{t})$. Since we call it for each $t \in \text{dom}(C)$, the total time complexity simplifies to $O(\max(|ub - lb|, |\text{dom}(S_i)|))$.

Sadler and Gervet [7] suggest adding a lexicographic ordering constraint to the domain description. This gives more expressiveness to the domain representation and can eliminate more undesired sets. that We say that $S_1 < S_2$ holds if S_1 comes before S_2 in a lexicographical order. The new domain representation now involves two lexicographic bounds l and u.

$$\text{dom}(S_i) = \{S \mid lb \subseteq S \subseteq ub, |S| = C, l \leq S \leq u\} \tag{2}$$

Knuth [8] represents all subsets of a set using a binomial tree like the one in Figure 1. The empty set is the root of the tree to which we can add elements by branching to a child. One can list all sets in lexicographical order by visiting

Algorithm 2. Enumerate the $\binom{n}{t}$ combinations of t elements between 0 and $n-1$. (Source: Algorithm T, Knuth [8] p.5)

$c_j \leftarrow j-1, \forall j\ 1 \leq j \leq t$
$c_{t+1} \leftarrow n$
$c_{t+2} \leftarrow 0$
repeat
  visit $c_t, c_{t-1}, \ldots, c_1$
  $j \leftarrow 1$
  while $c_j + 1 = c_{j+1}$ **do**
    $c_j \leftarrow j-1$
    $j \leftarrow j+1$
  $c_j \leftarrow c_j + 1$
until $j > t$

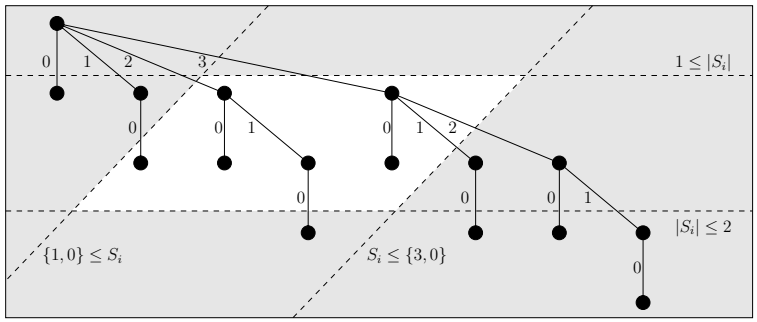

Fig. 1. Binomial tree representing the domain $\emptyset \subseteq S_i \subseteq \{0,1,2,3\}$, $1 \leq |S_i| \leq 2$, and $\{1,0\} \leq S_i \leq \{3,0\}$

the tree from left to right with a depth-first-search (DFS). We clearly see that the lexicographic constraints are orthogonal to the cardinality constraints.

Based on the binomial tree, we compute, level by level, the number of sets that belong to the domain. Notice that sets at level k have cardinality k. A set in the variable domain can be encoded with a binary vector of size $|ub - lb|$ where each bit is associated to a potential element in $ub - lb$. A bit set to one indicates the element belongs to the set while a bit set to zero means that the element does not belong to the set. The number of sets of cardinality k in the domain is equal to the number of binary vectors with k bits set to one and that lexicographically lie between l and u. Let $[u_m, \ldots, u_1]$ be the binary representation of the lexicographic upper bound u. Assuming $\binom{b}{a} = 0$ for all negative values of a, function $C([u_m, \ldots, u_1], k)$ returns the number of binary vectors that are lexicographically smaller than or equal to u and that have k bits set to one.

$$C([s_m,\ldots,s_1],k) = \sum_{i=1}^{m} s_i \binom{i-1}{k-\sum_{j=i+1}^{m} s_j} + \delta(\boldsymbol{s},k) \tag{3}$$

$$\delta([s_m,\ldots,s_1],k) = \begin{cases} 1 & \text{if } \sum_{i=1}^{m} s_i = k \text{ and } s_0 = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Lemma 2. *Equation 3 is correct.*

Proof. We prove correctness by induction on m. For $m = 1$, Equation 3 holds with both $k = 0$ and $k = 1$. Suppose the equation holds for m, we want to prove it also holds for $m + 1$. We have

$$C([s_{m+1},\ldots,s_1],k) = s_{m+1}\binom{m}{k} + C([s_m,\ldots,s_1],k-s_{m+1}) \tag{5}$$

If $s_{m+1} = 0$, the lexicographic constraint is the same as if we only consider the m first bits. We therefore have $C([s_{m+1},\ldots,s_1],k) = C([s_m,\ldots,s_1],k)$. If $s_{m+1} = 1$, $C(s,k)$ returns $\binom{m}{k}$ which corresponds to the number of vectors with k bits set to 1 and the $(m+1)^{th}$ bit set to zero plus $C([s_m,\ldots,s_1],k-1)$ which corresponds to the number of vectors with k bits set to 1 including the $(m+1)^{th}$ bit. Recursion 5 is therefore correct. Solving this recursion results in Equation 3. □

Let a and b be respectively binary vectors associated to the lexicographical bounds l and u where bits associated to the required elements lb are omitted. We refer by $a-1$ to the binary vector that precedes a in the lexicographic order. The size of the domain is given by the following equation.

$$|\mathrm{dom}(S_i)| = \sum_{k\in C} (C(b,k) - C(a-1,k))$$

Function C can be evaluated in $O(|ub-lb|)$ steps. The size of domain $\mathrm{dom}(S_i)$ therefore requires $O(|ub-lb||C|)$ steps to compute. Enumerating can also proceede level by level without taking into account the required elements lb since they belong to all sets in the domain. The first set on level k can be obtained from the lexicographic lower bound l. If $|l| \neq k$, we have to find the first set l' of cardinality k that is lexicographically greater than l. If $|l| < k$, we simply add to set l the $k-|l|$ smallest elements in $ub-lb-l$. Suppose $|l| > k$ and consider the binary representation of l. Let p be the k^{th} heaviest bit set to 1 in l. We add one to bit p and propagate carries and we set all bits before p to 0. We obtain a bit vector l' representing a set with no more than k elements. If $|l'| < k$, we add the first $k-|l'|$ elements in $ub-lb-l'$ to l' and obtain the first set of cardinality k.

Once the first set at level k has been computed, subsequent sets can be obtained using Algorithm 2. Obtaining the first set of each level costs $O(|\mathrm{dom}(C)|\ |ub-lb|)$ and cumulative calls to Algorithm 2 cost $O(\sum_{i\in\mathrm{dom}(C)} i + |dom(S)|)$. Enumerating the domain therefore requires $O(|\mathrm{dom}(C)||ub-lb|+|\mathrm{dom}(S)|)$ steps.

5 All-Different on Tuples

A tuple t is an ordered sequence of n elements that allows multiple occurrences. Like sets, there are different ways to represent the domain of a tuple. The most common way is simply by associating an integer variable to each of the tuple components. A tuple of size n is therefore represented by n integer variables $x_1, \ldots, x_n$.

To apply an All-Different constraint to a set of tuples, a common solution is to create an integer variable t for each tuple. If each component x_i ranges from 0 to c_i exclusively, we add the following channeling constraint between tuple t and its components.

$$t = ((((x_1 c_2 + x_2)c_3 + x_3)c_4 + x_4)\ldots)c_n + x_n = \sum_{i}^{n} \left(x_i \prod_{j=i+1}^{n} c_j \right)$$

This technique suffers from either inefficient or ineffective channeling between variable t and the components x_i. Most constraint libraries enforce bound consistency on t. A modification to the domain of x_i does not affect t if the bounds of $\mathrm{dom}(x_i)$ remain unchanged. Conversely, even if all tuples encoded in $\mathrm{dom}(t)$ have $x_i \neq v$, value v will most often not be removed from $\mathrm{dom}(x_i)$. On the other hand, enforcing domain consistency typically requires $O(n^k)$ steps where k is the size of the tuple.

To address this issue, one can define a tuple variable whose domain is defined by the domains of its components.

$$\mathrm{dom}(t) = \mathrm{dom}(x_1) \times \ldots \times \mathrm{dom}(x_n)$$

The size of such a domain is given by the following equation which can be computed in $O(n)$ steps.

$$|\mathrm{dom}(t)| = \prod_{i=1}^{n} |\mathrm{dom}(x_i)|$$

The domain of a tuple variable can be enumerated using Algorithm 3. Assuming the domain of all component variables have the same size, Algorithm 3 runs in $O(|dom(t)|)$ which is optimal.

As Sadler and Gervet [7] did for sets, we can add lexicographical bounds to tuples in order to better express the values the domain contains. Let l and u be these lexicographical bounds.

$$\mathrm{dom}(t) = \{t \mid t[i] \in \mathrm{dom}(x_i), l \leq t \leq u\}$$

Let $idx(v, x)$ be the number of values smaller than v in the domain of the integer variable x. More formally, $idx(v, x) = |\{w \in \mathrm{dom}(x) \mid w < v\}|$. Assuming $idx(v, x)$ has a running time complexity of $O(\log(|\mathrm{dom}(x)|))$, the size of

Algorithm 3. Enumerate tuples of size n in lexicographical order. (Source: Algorithm T, Knuth [8] p.2).

Initialize first tuple: $a_j \leftarrow \min(\text{dom}(x_j))$, $\forall j\ 1 \leq j \leq n$
repeat
  visit $(a_1, a_2, \ldots, a_n)$
  $j \leftarrow n$
  while $j > 0$ ***and*** $a_j = \max(dom(x_j))$ **do**
    $a_j \leftarrow \min(\text{dom}(x_j))$
    $j \leftarrow j - 1$
  $a_j \leftarrow \min(\{a \in \text{dom}(x_j) \mid a > a_j\})$
until $j = 0$

the domain can be evaluated in $O(n + \log(|\text{dom}(t)|))$ steps using the following equation.

$$|\text{dom}(t)| = 1 + \sum_{i=1}^{n} \left((idx(u[i], x_i) - idx(l[i], x_i)) \prod_{j=i+1}^{n} |\text{dom}(x_i)| \right)$$

We enumerate the domain of tuple variables with lexicographical bounds similarly as tuple variables without lexicographical bounds. We simply initialize Algorithm 3 with tuple l and stop enumerating when tuple u is reached. In average case analysis, this operation is performed in $O(|\text{dom}(t)|)$ steps.

6 All-Different on Multi-sets

Unlike sets, multi-sets allow multiple occurrences of the same element. We use $occ(v, S)$ to denote the number of occurrences of element v in multi-set S. An element v belongs to a multi-set A if and only if its number of occurrences $occ(v, A)$ is greater than 0. We say that set A is included in set B ($A \subseteq B$) if for all element v we have $occ(v, A) \leq occ(v, B)$. The domain representation of multi-sets is generally similar to the one for standard sets. We have a multi-set of essential elements lb and a multi-set of allowed elements ub. Equation 6 gives the domain of a multi-set and Equation 7 shows how to compute its size in $O(|ub|)$ steps.

$$\text{dom}(S_i) = \{S \mid lb \subseteq S \subseteq ub\} \tag{6}$$

$$|\text{dom}(S_i)| = \prod_{v \in ub} (occ(v, ub) - occ(v, lb) + 1) \tag{7}$$

Multisets can be represented by a vector where each component represents the number of occurrences of an element in the multi-set. Of course, for the multi-set to be in the domain, this number of occurrences must lie between $occ(v, lb)$ and $occ(v, ub)$. Therefore a multi-set variable is equivalent to a tuple variable where the domain of each component is given by the interval $[occ(v, lb), occ(v, ub)]$.

Enumerating the values in the domain is done as seen in Section 5. The same approach can be used to introduce lexicographical bounds to multi-sets.

7 Indexing Domain Values

Propagators for the All-Different constraint, like the one proposed by Régin [4], need to store information about some values appearing in the variable domains. When values are integers, the simplest implementation is to create a table T where information related to value v is stored in entry $T[v]$. Algorithm 1 ensures that the propagator is called over a maximum of n variables each having no more than n (possibly distinct) values in their domain. We therefore have a maximum of n^2 values to consider. When these n^2 values come from a significantly greater set of values, table T becomes sparse. In some cases, it might not even be realistic to consider such a solution. To allow direct memory access when accessing the information of a value, we need to map the n^2 values to an index in the interval $[1, n^2]$.

We suggest to build an indexing tree able to index sets, multi-sets, tuples, or any other sequential data structure. Each node is associated to a sequence. The root of the tree is the empty sequence ($\emptyset$). We append an element to the current sequence by branching to a child of the current node. There are at most n^2 nodes corresponding to a value in a variable domain. These nodes are labeled with integers from 1 to n^2. Figure 2 shows the indexing tree based on the domain of 5 set variables.

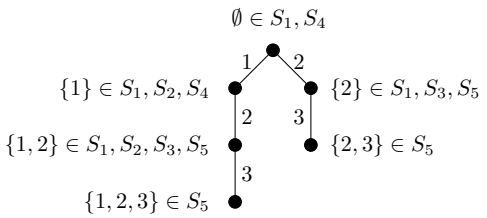

Fig. 2. Indexing tree representing the following domains: $\emptyset \subseteq S_1 \subseteq \{1,2\}, \{1\} \subseteq S_2 \subseteq \{1,2\}, \{2\} \subseteq S_3 \subseteq \{1,2\}, \emptyset \subseteq S_4 \subseteq \{1\}, \{2\} \subseteq S_5 \subseteq \{1,2,3\}$

This simple data structure allows to index and retrieve in $O(l)$ steps the number associated to a sequence of length l.

8 Global Cardinality Constraint

The global cardinality constraint (*gcc*) is a generalization of the All-Different constraint. A value v must be assigned to at least $\lfloor v \rfloor$ variables and at most $\lceil v \rceil$ variables. Traditionally, the *lower capacity* $\lfloor v \rfloor$ and the *upper capacity* $\lceil v \rceil$ are

given by look-up tables. When working with large domains, these look-up tables could require too much memory. We therefore assume that the lower and upper capacity of each value is returned by a function. For instance, the constant functions $\lfloor v \rfloor = 0$ and $\lceil v \rceil = 1$ define the All-Different constraint. In order to be feasible, the following restrictions apply: $\sum_v \lfloor v \rfloor \leq n$ and $\sum_v \lceil v \rceil \geq n$. For efficiency reasons, we assume that the values L whose lower capacity is positive are known, i.e. $L = \{v \mid \lfloor v \rfloor > 0\}$ is known.

Based on the concept of upper capacity, we give a new definition to a Hall set.

Hall set [9]. A Hall set H is a set of values such that there are $\sum_{v \in H} \lceil v \rceil$ variables whose domains are contained in H; i.e., H is a Hall set iff $|\{x_i \mid \text{dom}(x_i) \subseteq H\}| = \sum_{v \in H} \lceil v \rceil$.

Under *gcc*, Lemma 1 becomes the following lemma.

Lemma 3. *Let F be a set of variables whose domains are not contained in any Hall set and assume $\lceil v \rceil \geq k$ holds for all value v. If $x_i \notin F$ is a variable whose domain contains more than $\lfloor \frac{n-|F|}{k} \rfloor$ values, then dom(x_i) is not contained in any Hall set.*

Proof. The largest Hall set can contain the domain of $n - |F|$ variables and therefore has at most $\lfloor \frac{n-|F|}{k} \rfloor$ values. If $|\text{dom}(x_i)| > \lfloor \frac{n-|F|}{k} \rfloor$, then $\text{dom}(x_i)$ cannot be contained to any Hall set. □

Following [9], the *gcc* can be divided into two constraints: the lower bound constraint is only concerned with the lower capacities ($\lfloor v \rfloor$) and the upper bound constraint is only concerned with the upper capacities ($\lceil v \rceil$).

The upper bound constraint is similar to the All-Different constraint. Up to $\lceil v \rceil$ variables can be assigned to a value v instead of only 1 with the All-Different constraint. Lemma 3 suggests to modify Line 1 of Algorithm 1 by testing if $|\text{dom}(x_i)| > \frac{|X|-|F|}{k}$ before inserting variable x_i in set F.

The lower bound constraint can easily be handled when variable domains are large. Consider the set L of values whose lower capacity is positive, i.e. $L = \{v \mid \lfloor v \rfloor > 0\}$. In order for the lower bound constraint to be satisfiable over n variables, the cardinality of L must be bounded by n. The values not in L can be assigned to a variable only if all values v in L have been assigned to at least $\lfloor v \rfloor$ variables. Since all values not in L are symmetric, we can replace them by a single value p such that $\lfloor p \rfloor = 0$. We now obtain a problem where each variable domain is bounded by $n + 1$ values. We can apply a propagator for the lower bound constraint on this new problem. Notice that if the lower bound constraint propagator removes p from a variable domain, it implies by symmetry that all values not in L should be removed from this variable domain.

9 Experiments

To test the efficiency and effectiveness of these generalizations to the propagator for the All-Different constraint, we ran a number of experiments on a well

known problem from design theory. A Latin square is an $n \times n$ table where cells can be colored with n different colors. We use integers between 1 and n to identify the n colors. A Graeco-Latin square is m Latin squares $A_1, \ldots, A_m$ such that the tuples $\langle A_1[i,j], \ldots, A_m[i,j] \rangle$ are all distinct. The following tables represent a Graeco-Latin square for $n = 4$ and $m = 2$.

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

3	4	1	2
1	2	3	4
2	1	4	3
4	3	2	1

We encode the problem using one tuple variable per cell. There is an All-Different constraint on each row and each column. We add a redundant 0/1-Cardinality-Matrix constraint on each value as suggested by Régin [11]. We use two different encodings for tuples: one is the tuple encoding where each component is an integer variable, the other is the factored representation. We enforce bounds consistency on the channeling constraints between the cell variables and the factored tuple variables. As suggested in [11], our heuristic chooses the variable with the smallest domain and we break ties on the variable that has the most bounded variables on its row and column. We use the same implementation of the All-Different propagator for both tuple encodings.

Table 1 and Figure 3 clearly show that when tuples gets longer, our technique outperforms the factored representation of tuples. This is mainly due to space requirements since the factored representation of tuples requires more memory than the cache can contain.

Table 1. Time to solve a Graeco-Latin square using factored and tuple variables

n \ m	3		4		5		6	
	factored	tuple	factored	tuple	factored	tuple	factored	tuple
8	0.48	0.23	0.57	0.35	4.51	0.40	56.48	1.08
9	0.33	0.49	0.31	0.85	1.77	0.94	23.09	2.39
10	0.58	0.91	0.56	1.57	3.44	1.78	52.30	4.36
11	1.05	1.62	1.04	2.97	7.33	3.23	124.95	7.69
12	1.76	2.80	1.79	5.59	13.70	6.04	263.28	13.61
13	2.86	4.69	2.85	9.00	23.96	9.74	493.04	22.80
14	4.37	7.03	4.17	14.34	38.95	15.19		33.79
15	6.88	10.62	6.56	22.18	69.89	23.63		50.23
16	10.11	15.41	9.54	32.52	110.08	34.55		73.60
17	14.21	21.48	13.82	45.35	174.18	47.89		102.98
18	20.41	30.55	19.13	64.87	255.76	68.46		146.21
19	28.28	42.12	25.01	91.45	364.58	95.99		204.45
20	38.31	56.10	34.35	122.30	540.06	136.43		274.29

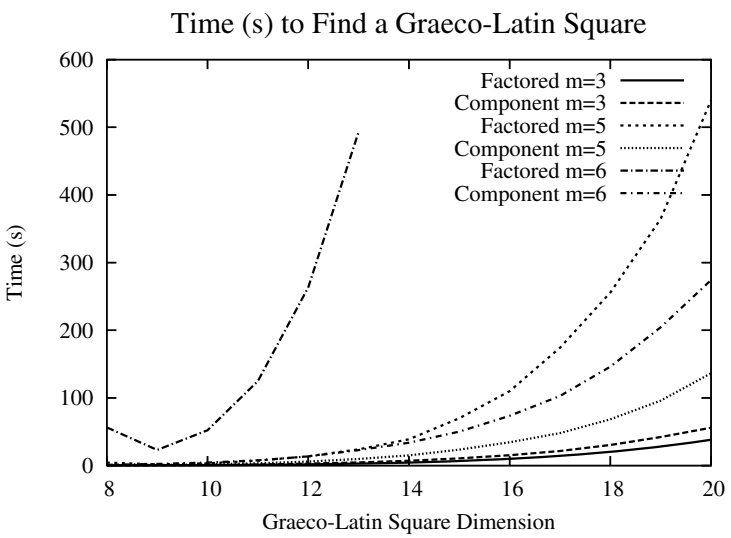

Fig. 3. Time in seconds to solve a Graeco-Latin square with m different square sizes. The data is extracted from Table 1. We see that for $m \geq 5$, the component encoding offers a better performance than the factored encoding.

10 Conclusions

We have described how Régin's propagator for the All-Different constraint can be generalized to prune variables whose domains are not just simple finite domains. In particular, we described how it can be used to propagate set variables, multiset variables and variables which represent tuples of values. We also described how the propagator for the global cardinality constraint can be generalized in a similar way. Experiments showed that such propagators can be beneficial in practice, especially when the domains are large. Many other global constraints still remain to be generalized to deal with other variable types than simple integer domains.

References

1. P. Hall, On representatives of subsets. *Journal of the London Mathematical Society*, pages 26–30, 1935.
2. J. Hopcroft and R. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, volume 2 pages 225–231, 1973.
3. ILOG S. A., *ILOG Solver 4.2 user's manual*. 1998.

4. J.-C. Régin, A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 362–367, Seattle, 1994.
5. J.-C. Régin, Generalized arc consistency for global cardinality constraint. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 209–215, Portland, Oregon, 1996.
6. K. Stergiou and T. Walsh, The difference all-difference makes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 414–419, Stockholm, 1999.
7. A. Sadler and C. Gervet, Hybrid Set Domains to Strengthen Constraint Propagation and Reduce Symmetries. In *In Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 604–618, Toronto, Canada, 2004.
8. D. Knuth, *Generating All Tuples and Permutations*. Addison-Wesley Professional, 144 pages, 2005.
9. A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek, A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 245–250, Acapulco, Mexico, 2003.
10. W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing, Second Edition,* Cambridge University Press, 1992.
11. J.-C. Régin and C. P. Gomes, The Cardinality Matrix Constraint. In *In Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 572–587, Toronto, Canada, 2004.
12. C. Gervet, Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *Constraints Journal,* 1(3) pages 191–244, 1997.
13. C. Gervet, *Set Intervals in Constraint Logic Programming: Definition and Implementation of a Language.* PhD thesis, Université de Franche-Comté, France, September 1995. European thesis, in English.
14. J.-F. Puget, Finite set intervals. In *Proceedings of Workshop on Set Constraints*, held at CP'96, 1996.
15. T. Müller and M. Müller, Finite set constraints in Oz. In François Bry,Burkhard Freitag, and Dietmar Seipel, editors, *13. Workshop Logische Programmierung*, pages 104–115, Technische Universität München, pages 17–19 September 1997.
16. T. Walsh, Consistency and Propagation with Multiset Constraints: A Formal Viewpoint. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, Kinsale, Ireland, 2003.
17. I.P. Gent and T. Walsh, CSPLib: a benchmark library for constraints. *Technical report APES-09-1999*, 1999.
18. M. Leconte, A bounds-based reduction scheme for constraints of difference. In *Proceedings of the Constraint-96 International Workshop on Constraint-Based Reasoning,* pages 19–28, 1996.
19. A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek, A fast and simple algorithm for bounds consistency of the alldifferent constraint. in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)* pages 245–250, 2003.
20. J.-F. Puget, A Fast Algorithm for the Bound Consistency of Alldiff Constraints. In *Proceedings of the 15th National Conference on Artificiel Intelligence (AAAI-98)* and the *10th Conference on Innovation Applications of Artificial Intelligence (IAAI-98)*", pages 359–366, 1998.