Vol.2011-AL-134 No.22
2011/3/7

Regular Paper

Around the Constructive Orbit Problem in Distributed Constraint Programming

Xavier Olive[†1] and Hiroshi Nakashima[†1]

This paper presents an efficient method to list equivalence classes (or orbits) of assignations of values to variables, where assignations are symmetrical through a group of variables and values permutations. We also present an algorithm designed to compute efficiently a canonical form of any assignation with respect to those orbits — problem also known as the constructive orbit problem. We illustrate how efficient our algorithm is and how it can significantly improve the resolution of symmetrical problems with DPOP, an efficient class of algorithms solving distributed constraint programming problems.

1. Introduction

This section presents a set of notions and notations required to understand the problem we want to solve and how relevant this problem is to our field of application.

We consider a set of n variables x_i and a set of k values u_j.

Definition 1 Let $\alpha = \left\{x_i \leftarrow u_{\varphi(i)}\right\}_{i\in[1\cdots n]}$, with $\varphi : [1\cdots n] \longrightarrow [1\cdots k]$ be an *assignation* of n variables over k values.

Let $\mathcal{A} = [1\cdots k]^n$ be the set of all possible assignations. We consider a utility function $f : \mathcal{A} \longrightarrow \mathbb{K}$ associating a value to an assignation. We will refer in this paper indifferently to α_i and φ_i which are two representations of the same object. The condensed notation $(1, 2, 2, 3)$ refers to the assignation $\alpha = \{x_1 \leftarrow 1, x_2 \leftarrow 2, x_3 \leftarrow 2, x_4 \leftarrow 3\}$ or $\varphi : \{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 2, 4 \mapsto 3\}$.

Definition 2 Let σ be a bijection from $\mathcal{A}$ to $\mathcal{A}$ so that, for each $\alpha \in \mathcal{A}$, the equality $f(\alpha) = f(\sigma(\alpha))$ stands. σ is called a *symmetry* with respect to f.

Proposition 1 The set of symmetries with respect to f form a group.

Among the group of symmetries G, we distinguish the variable symmetries, which exclusively permutate variables, and value symmetries which exclusively permutate values. We focus here on groups that are generated from a set of variable symmetries and from a set of value symmetries.

Definition 3 We name *orbit* of an element α all the elements in $\mathcal{A}$ that are images of α through an element of G. In other words, $G \cdot \alpha = \{\sigma(\alpha) \,|\, \sigma \in G\}$.

Definition 4 Let $\ll$ denote a lexicographic order operator on the set of assignations:

$$\varphi_1 \ll \varphi_2 \Leftrightarrow \begin{cases} \varphi_1(1) < \varphi_2(1) \\ \exists\, i \in [2\ldots n], \Big(\varphi_1(i) < \varphi_2(i) \ \wedge\ \forall j < i, \varphi_1(j) = \varphi_2(j)\Big) \end{cases}$$

Definition 5 We name *canonical representation* of an orbit $G\cdot\alpha$ the element $\dot{\alpha}$ which is the lowest element in lexicographic order in an orbit $G\cdot\alpha$. We define the function γ such as for all $\beta \in G\cdot\alpha$, we have $\gamma(\beta) = \dot{\alpha}$.

With 3 variables taking values in $\{0, 1, 2\}$, if G is the group of the symmetries swapping all variables and all values with each other, then the orbit of $(1, 1, 1)$ is $\{(0, 0, 0), (1, 1, 1), (2, 2, 2)\}$. The canonical representation of this orbit is $(0, 0, 0)$. We write $\gamma(1, 1, 1) = (0, 0, 0)$.

G separates all 27 assignations of 3 variables taking values in $\{0, 1, 2\}$ into three orbits whose canonical representations are respectively $G\cdot(0, 0, 0)$ for all identical elements, $G \cdot (0, 0, 1)$ for exactly two (among three) identical elements, and $G \cdot (0, 1, 2)$ for all different elements. The purpose of this paper is double. For a given set of variables, a given set of values, and a symmetry group G, we present an efficient way to:

- compute the γ function — referred to as the *constructive orbit problem*;
- list the canonical representations of all different orbits.

We claim that those two methods are of critical importance in the treatment of symmetries in distributed constraint programming.

Related works

Clarke[2], then Donaldson[3] analysed the constructive orbit problem in depth, whilst its general form had been proved NP-hard by Babai[1]. However, they focused on the context of model checking. The theory developped around model checking can be directly applied to a very simplified version of our problem, respectively a problem with no value symmetry. Donaldson suggested that the

†1 Kyoto University

constructive orbit problem — the first item of the problem we want to solve — can be efficiently solved for three categories of symmetry groups: fully symmetric groups, disjoint products and wreath products.

Let us illustrate Donaldson's model with a problem of four variables taking values in $\{1, 2, 3, 4\}$. G is the group of the symmetries permutating all variables *while leaving the values invariant.* The most obvious naive strategy for computing $\gamma(s)$ is to compute $G \cdot s$ and to return the smallest element; in order to enumerate all the orbits, the equivalent strategy would be to compute all the $\gamma(s)$ for every possible s. For example, computing $\gamma(3, 1, 2, 4)$ the naive way would mean listing $(3, 1, 2, 4), (1, 3, 2, 4), (2, 1, 3, 4), \dots$ and pick the smallest element, i.e. $(1, 2, 3, 4)$. As G, in this particular case where all variables are permutable, is a fully symmetric group, Donaldson recommends here to sort values in lexicographic order.

Let us now consider the same variables with a different symmetry group. G is now permutating the first and second variables on one hand, and the third and fourth variables on the other hand. G is not a fully symmetric group, however it is the disjoint product of $G_{1,2}$ (swapping the 1st and the 2nd variable) and $G_{3,4}$. Donaldson proved then that $\gamma(3, 1, 2, 4) = \gamma_{1,2} \circ \gamma_{3,4}(3, 1, 2, 4)$. In other words, compute separately $\gamma(3, 1)$ and $\gamma(2, 4)$ then join the results: $\gamma(3, 1, 2, 4) = (1, 3, 2, 4)$. Sorting $(3, 1)$ and $(2, 4)$ separately — or listing-then-choosing the lowest element if one multiplicand is not a fully symmetry group — is obviously more efficient than dealing with the whole 4-variable vector.

This approach, albeit reasonable in the case of variable-only symmetries, is not applicable in the case of variable/value symmetries, where the identification of disjoint or wreath product is far from obvious in a context of assignations (comparable to $\mathbb{N} \times \mathbb{N}$), unlike model checking (comparable to $\mathbb{N}$).

We propose here a method able to deal with variable/value symmetries, and also build from scratch a way to list all orbits. We start from an equivalent of the fully symmetric group in this "two-dimension" (variable and value) context (section 2), and extend to the general case where only some variables and some values are permutable (section 4). We then compare performance with the naive method (section 5.1) and show how this method is crucial in distributed constraint programming (section 5.2).

2. Unique set of permutable variables, unique set of permutable values

In this section, we consider a set of interchangeable variables $V = \{x_1, \cdots x_n\}$, and a set of interchangeable values $D = \{u_1, \cdots u_k\}$ with regards to utility function f. We name G the group of symmetries applicable on $\mathcal{A}$.

2.1 γ function

Let (φ, α) be an assignation. In $\alpha = \left\{x_i \leftarrow u_{\varphi(i)}\right\}_{i \in [1 \cdots n]}$, we have at most k different values in $\left\{u_{\varphi(i)}\right\}$. We name $\#(u_j)$ the number of occurences of u_j in α (or the number of occurences of j in the arrival domain of φ).

Lemma 1 The distribution of $\#(u_i)$ is preserved through all $\sigma \in G$.

Proof Permutating variables keeps all $\#(u_i)$ intact and permutating values $u_i \rightleftharpoons u_j$ just permutates $\#(u_i)$ with $\#(u_j)$. □

Proposition 2 $\dot{\alpha} = \left\{u_{\varphi(i)}\right\}$ is a canonical form iff the three following conditions are fulfilled:

$$\begin{cases} \varphi(1) = 1 \\ \varphi(i) = \varphi(i-1) + \{0, 1\} \text{ for } i > 1 \\ i < j \Leftrightarrow \#(u_i) \geq \#(u_j) \end{cases}$$

Proof Consider $\dot{\alpha} = \left\{u_{\varphi(i)}\right\}$ a canonical form. The proof of the first line is direct: if $\varphi(1) \neq 1$, then you can just swap $\varphi(1)$ and 1 and get a symmetric assignation lower in lexicographic order. Similarly, if $\varphi(i) < \varphi(i-1)$, we can swap them and get a lower element in the orbit $G \cdot \dot{\alpha}$; if $\varphi(i) > \varphi(i-1) + 1$, we would swap $\varphi(i)$ with $\varphi(i-1) + 1$ and prove $\dot{\alpha}$ would not be a canonical form. About the third line, let's consider

$$\alpha = (u_1, \dots u_i, u_i, u_i, \dots u_j, u_j, u_j, u_j, \dots)$$

By "moving" upwards the variables assigned to u_j, before the variables assigned to u_i, and by then swapping $u_i \rightleftharpoons u_j$, we get

$$\beta = (u_1, \dots u_i, u_i, u_i, u_i, \dots u_j, u_j, u_j, \dots)$$

We just exhibited β, an element of $G \cdot \alpha$ which lexicographically precedes α.

For the converse, consider now φ_1 being a canonical form, thus satisfying the three conditions and φ_2 another element of φ_1's orbit, also satisfying the same

conditions. Let's prove by recursion that φ_2 is necessarily equal to φ_1. First, we have $\varphi_1(1) = \varphi_2(1)$. Suppose now that $\varphi_1(i) = \varphi_2(i)$ for all $i < j$. Lemma 1 and the third condition yield $\#(u_{\varphi_1(k)}) = \#(u_{\varphi_2(k)})$ for all indices k.
Therefore, $\varphi_1(j) = \varphi_1(j-1) \Leftrightarrow \varphi_2(j) = \varphi_2(j-1)$, and thanks to the second condition, $\varphi_1(j) = \varphi_1(j-1) + 1$ implies that $\varphi_2(j) = \varphi_2(j-1) + 1$.

We conclude that $\varphi_1 = \varphi_2$ are the same canonical form of the considered orbit. □

In order to build the set of permutations to operate on variables and values and get $(\dot{\varphi}, \dot{\alpha})$, we define the bijection $\tau : [1 \cdots n] \longrightarrow [1 \cdots n]$ that will sort variables by the occurence frequency of the values assigned to them. In case of equality, we sort variables in the lexicographical order of assigned values and then that of variables themselves. In other words, for any couple (i, j),

$$\tau(i) < \tau(j) \Leftrightarrow \begin{cases} \#(u_{\varphi\cdot\tau(i)}) \geq \#(u_{\varphi\cdot\tau(j)}) & \text{if } \#(u_{\varphi\cdot\tau(i)}) \neq \#(u_{\varphi\cdot\tau(j)}) \\ \varphi\cdot\tau(i) < \varphi\cdot\tau(j) & \text{if } \begin{cases} \#(u_{\varphi\cdot\tau(i)}) = \#(u_{\varphi\cdot\tau(j)}) \\ \text{and } \varphi\cdot\tau(i) \neq \varphi\cdot\tau(j) \end{cases} \\ i < j & \text{otherwise} \end{cases}$$

Then, we define the bijection $\upsilon : [1 \cdots k] \longrightarrow [1 \cdots k]$ in order to permute values so that the most used value becomes u_1, the second most used value becomes u_2, etc. Strictly speaking,

$$\text{for any couple } (i,j),\ \upsilon(i) < \upsilon(j) \Leftrightarrow \begin{cases} \#(u_i) < \#(u_j) & \text{if } \#(u_i) \neq \#(u_j) \\ i < j & \text{otherwise} \end{cases}$$

Proposition 3 $\dot{\alpha} = \{x_i \leftarrow u_{\upsilon\cdot\varphi\cdot\tau(i)}\}$, $(\dot{\varphi} = \upsilon \circ \varphi \circ \tau,\ \dot{\alpha})$ is by construction the lowest element in lexicographical order in the orbit of (φ, α), i.e., $\dot{\alpha} = \gamma(\alpha)$.

For example, let $V = \{x_1, x_2, x_3\}$ and $D = \{1, 2, 3, 4, 5\}$. We want to compute $\gamma(4, 3, 4)$, i.e., find υ and τ for $\varphi : \{1 \mapsto 4,\ 2 \mapsto 3,\ 3 \mapsto 4\}$. We start counting $\#(1) = 0$, $\#(2) = 0$, $\#(3) = 1$, $\#(4) = 2$, and $\#(5) = 0$. We then define τ to swap $x_2 \leftrightharpoons x_3$, then υ to reorder values as follows:

$$\upsilon \circ \varphi \circ \tau : \begin{Bmatrix} 1 & \longmapsto & 3 \\ 2 & \longmapsto & 4 \\ 3 & \longmapsto & 2 \\ 4 & \longmapsto & 1 \\ 5 & \longmapsto & 5 \end{Bmatrix} \circ \begin{Bmatrix} 1 & \longmapsto & 4 \\ 2 & \longmapsto & 3 \\ 3 & \longmapsto & 4 \end{Bmatrix} \circ \begin{Bmatrix} 1 & \longmapsto & 1 \\ 2 & \longmapsto & 3 \\ 3 & \longmapsto & 2 \end{Bmatrix} = \begin{Bmatrix} 1 & \longmapsto & 1 \\ 2 & \longmapsto & 1 \\ 3 & \longmapsto & 2 \end{Bmatrix}$$

We found $\boxed{\gamma(4, 3, 4) = (1, 1, 2).}$

2.2 List of orbits

Lemma 2 The number of orbits in G is equal to the number of different decompositions of integer n into a sum of at most k positive integers.

Proof We associate to each decomposition $k_1 + k_2 + \cdots k_{m \leq k} = n$ (with k_i in descending order) the following assignation

$$\alpha = \left\{ \underbrace{u_1, u_1, \cdots u_1}_{k_1}, \underbrace{u_2, \cdots u_2}_{k_2}, \cdots \underbrace{u_m, \cdots u_m}_{k_{m \leq k}} \right\} \tag{1}$$

As any assignation which does not match that pattern is not a canonical form according to Proposition 2, we cannot have more orbits than partitions. Moreover, as both variable and value permutations globally preserve the distribution of $\#(u_i)$, we cannot have less orbits than partitions. □

Corollary 1 Listing the orbits of assignations of n permutable variables over k permutable values is equivalent to listing all partitions of n elements into subsets of at most k elements.

We describe here in detail a way to list all partitions of n elements into subsets of at most k elements. The parameter m being the biggest element in the subset is to be initialized to n.

The listing on the beginning of next page describes a way to list all partitions of n elements into subsets of at most k elements. For example, if we want to list all canonical forms of $V = \{x_1, x_2, x_3\}$ and $D = \{1, 2, 3, 4, 5\}$, we start computing `partition`$(3, 5)$, then build the corresponding assignation as in Equation 1.

```python
# n is the sum of all elements in a subset
# m is the biggest element in a subset
# k is the max number of elements in a subset

def partition(n,m,k):
    result = []
    for i in range(n-m,n-(n-1)/k): # ensures the first element is the biggest
        if (i==0):
            result.append([n])
        else:
            for e in partition(i,min(n-i,i,m),(k-1)):
                result.append([n-i]+e) # the + symbol stands for concatenation
    return result
```

$3 = 3 \Rightarrow (1,1,1)$
$3 = 2+1 \Rightarrow (1,1,2)$
$3 = 1+1+1 \Rightarrow (1,2,3)$

3. Unique set of permutable variables, set of permutable values' sets

We consider here a set of interchangeable variables $V = \{x_1, \cdots x_n\}$ and a set of sets of interchangeable values $D = \{D_1, \cdots D_m\}$. We name $d(u)$ the indice of the domain containing value u. Without loss of generality[⋆1], we assume that $d(u) < d(v) \Rightarrow u < v$. We name G the group of symmetries applicable on $\mathcal{A}$.

3.1 γ function

Let (φ, α) be an assignation of variables.

We first reorder variables in order to group together permutable values; we build the bijection $\psi : [1 \cdots n] \longrightarrow [1 \cdots n]$ so that

$\psi(i) < \psi(j) \Leftrightarrow d\left(u_{\varphi\cdot\psi(i)}\right) < d\left(u_{\varphi\cdot\psi(j)}\right) \vee \left(d(u_{\varphi\cdot\psi(i)}) = d(u_{\varphi\cdot\psi(j)}) \wedge i < j\right)$

Then, we deal with each set of permutable values separately.

For example, let $V = \{x_1, x_2, x_3, x_4, x_5\}$ and $D = \{\{1,2\},\{3,4\},\{5\}\}$. We want to compute $\gamma(3,2,2,4,3)$. We first permutate variables (by $\psi : \{1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 2, 4 \mapsto 4, 5 \mapsto 5\}$) in order to isolate $\{1,2\}$ and $\{3,4\}$: $\gamma(3,2,2,4,3) = \gamma(2,2,3,4,3)$, before solving separately the two problems:

- $V_1 = \{x_1, x_2\}, D_1 = \{1,2\}, \boxed{\gamma_1(2,2) = (1,1)}$
- $V_2 = \{x_3, x_4, x_5\}, D_2 = \{3,4\}, \boxed{\gamma_2(3,4,3) = (3,3,4)}$

and concatenate them into:

$\gamma(3,2,2,4,3) = \gamma(2,2,3,4,3) = \gamma_1(2,2) * \gamma_2(3,4,3) = (1,1) * (3,3,4) = \boxed{(1,1,3,3,4)}$

3.2 List of orbits

We naturally extend Lemma 1 into the following:

Lemma 3 For each assignation $\alpha \in \mathcal{A}$, $\#(D_i) = \sum_{u_{i,j} \in D_i} \#(u_{i,j})$, is invariant through any permutation in G.

Corollary 2 We can list all orbits for $V \times \{D_i\}$ by enumerating all decompositions of n into sums of $\#(D_i)$.

For instance, for $V = \{x_1, x_2, x_3, x_4, x_5\}$ and $D = \{\{1,2\},\{3,4\},\{5\}\}$, we first decompose 5 into sums of no more than 3 elements.

$$\begin{aligned}
5 = 5 &\Rightarrow \{\#(D_1) = 5\}, \{\#(D_2) = 5\}, \{\#(D_3) = 5\} \\
5 = 4+1 &\Rightarrow \{\#(D_1) = 4, \#(D_2) = 1\}, \{\#(D_1) = 4, \#(D_3) = 1\} \\
&\quad \{\#(D_2) = 4, \#(D_3) = 1\}, \{\#(D_1) = 1, \#(D_2) = 4\} \\
&\quad \{\#(D_1) = 1, \#(D_3) = 4\}, \{\#(D_2) = 1, \#(D_3) = 4\} \\
5 = 3+2 &\Rightarrow \{\#(D_1) = 3, \#(D_2) = 2\}, \{\#(D_1) = 3, \#(D_3) = 2\} \\
&\quad \{\#(D_2) = 3, \#(D_3) = 2\}, \{\#(D_1) = 2, \#(D_2) = 3\} \\
&\quad \{\#(D_1) = 2, \#(D_3) = 3\}, \{\#(D_2) = 2, \#(D_3) = 3\} \\
5 = 3+1+1 &\Rightarrow \{\#(D_1) = 3, \#(D_2) = 1, \#(D_3) = 1\} \\
&\quad \{\#(D_1) = 1, \#(D_2) = 3, \#(D_3) = 1\} \\
&\quad \{\#(D_1) = 1, \#(D_2) = 1, \#(D_3) = 3\} \\
5 = 2+2+1 &\Rightarrow \{\#(D_1) = 2, \#(D_2) = 2, \#(D_3) = 1\} \\
&\quad \{\#(D_1) = 2, \#(D_2) = 1, \#(D_3) = 2\} \\
&\quad \{\#(D_1) = 1, \#(D_2) = 2, \#(D_3) = 2\}
\end{aligned}$$

Then for each $\{\#(D_i) = k_i\}$, we split V into $\{x_1, \ldots x_{k_1}\}, \{x_{k_1+1}, \ldots x_{k_2}\}, \ldots$ and refer to the case of unique set of permutable values for listing the orbits of $\{x_{k_{i-1}+1}, \ldots x_{k_i}\}$ variables taking values on D_i.

For instance, for $\{\#(D_1) = 2, \#(D_2) = 3\}$, we list separately the orbits of

⋆1 We could redefine the lexicographical order so that $(u) \ll (v)$ iff $d(u) < d(v) \vee (d(u) = d(v) \wedge u < v)$

$\{x_1, x_2\}$ taking values on $\{1, 2\}$ (resp. $(1, 1)$ and $(1, 2)$) and of $\{x_3, x_4, x_5\}$ taking values on $\{3, 4\}$ (resp. $(3, 3, 3)$ and $(3, 3, 4)$) before concatenating them.

Therefore, the orbits issued of $\{\#(D_1) = 2, \#(D_2) = 3\}$ are

$$\boxed{(1,1,3,3,3), (1,1,3,3,4), (1,2,3,3,3), (1,2,3,3,4)}$$

4. Set of permutable variables' sets, set of permutable values' sets

In this section, we consider a set of sets of interchangeable variables $V = \{V_1, \cdots V_n\}$ and a set of sets of interchangeable values $D = \{D_1, \cdots D_m\}$. We assume that all $V_i = \left\{x_{i_1}, \cdots x_{i_{l_i}}\right\}$ and all $D_i = \left\{u_{i_1}, \cdots u_{i_{k_i}}\right\}$ are ordered so that $i < j \Rightarrow i_1 < j_1$. We name G the group of symmetries applicable on $\mathcal{A}$.

4.1 γ function

Let (φ, α) be an assignation of variables. We want to find the first element $\dot{\alpha}$ in lexicographic order in the orbit of α. We first decompose φ according to the $\{V_i\}$ in $\{\varphi_1, \ldots \varphi_n\}$.

While we treat the V_i separately, we cannot escape the fact that a permutation of the values has an impact on all the V_i together. If we want to proceed recursively, once we find a canonical form on V_i, the value permutations we would apply on V_{i+1} to find a canonical form also have an impact on V_i. As a consequence, we need to find a way to characterize value permutations on V_{i+1} that will have no impact on this V_i.

That idea brought up the following definition of the ∇_α operator.

Definition 6 We define the ∇_α operator to decompose each set of symmetric values in $D = \{D_1, \ldots D_m\}$ according to the number of occurrence of each value in an assignation α:

$$\nabla_\alpha \cdot D = \left\{ \left\{ x \in D_i \mid \#(x) = j \right\}_{\substack{i \in [1..m] \\ j \in \mathbb{N}}} \right\} \tag{2}$$

For example, if $D = \{\{1, 2, 3, 4\}\}$ and $\alpha = (1, 1, 2, 2, 3)$, then we have the following — because $\#(1) = \#(2) = 2$, $\#(3) = 1$ and $\#(4) = 0$:

$$\nabla_\alpha \cdot D = \{\{1, 2\}, \{3\}, \{4\}\}$$

This means that if we swap $1 \rightleftharpoons 2$, we can swap $x_1 \rightleftharpoons x_3$ and $x_2 \rightleftharpoons x_4$ and get α back. However, we cannot swap $1 \rightleftharpoons 3$ because there is no variable permutation to get α back.

Proposition 4 We can find the global Γ function by recursivity:

- we first compute γ^D on the assignation restricted on the first set of permutable variables $\dot{\varphi}_1$,
- we build a ∇_1 operator with respect to that $\dot{\varphi}_1$,
- we prepend $\dot{\varphi}_1$ to the result of Γ on the assignation restricted to the remaining variables, on which we apply υ_1 and τ_1 found in the first step.

In other words,

$$\Gamma^D\Big(\varphi_1 * \cdots * \varphi_n\Big) = \underbrace{\gamma^D(\varphi_1)}_{\upsilon_1 \circ \varphi_1 \circ \tau_1} * \Gamma^{\nabla_1 \cdot D}\Big(\upsilon_1 \circ (\varphi_2 * \cdots * \varphi_n) \circ \tau_1\Big) \tag{3}$$

We will show at first that $\Gamma^D(\varphi)$ is an element of the orbit of φ by exhibiting a sequence of variable and value permutation leading to it; then show that no element in the orbit of φ can be smaller in lexicographic order than $\Gamma^D(\varphi)$. For that purpose, we need the following lemma and its corollary on ∇_φ operator.

Lemma 4 Let φ be an assignation of all permutable variables over a set of values. For any permutation of two values $\upsilon_{a \rightleftharpoons b}$, there exists a permutation of variables ψ restoring φ iff a and b have the same number of occurences. In other words,

$$\forall \upsilon_{a,b}\ \exists \tau_\upsilon \ s.t.\ \ \upsilon_{a,b} \circ \varphi \circ \psi = \varphi \Leftrightarrow \#(a) = \#(b)$$

Proof

Indeed, provided that a and b have the same number of occurrences, we can construct τ_υ by swapping the first variable that takes value a with the first variable that takes value b, the second variable that takes value a with the second variable that takes value b, and so forth. Conversely, if $\#(a) > \#(b)$, i.e. we have more variables assigned to value a than to value b, we cannot swap them by pair. □

Since any permutation can be decomposed as a sequence of pair permutations, and since ∇_φ decomposes values' sets according to the number of occurences of values in φ, we can state the following,

Corollary 3 Let φ be an assignation of all permutable variables over a set of values' sets D. For any permutation of values υ, there exists a permutation of variables ψ such as $\varphi = \upsilon \circ \varphi \circ \psi$ iff $\upsilon \in U_\varphi$ where U_φ is the group of value

permutation inferred by $\nabla_\varphi \cdot D$.

Proof of Proposition 4

Let's consider two sets of permutable variables, a set of values' sets D, and a simple assignation $\varphi = \varphi_1 * \varphi_2$. We can write:

$$\begin{aligned}\Gamma^D\Big(\varphi_1 * \varphi_2\Big) &= \underbrace{\gamma^D(\varphi_1)}_{\upsilon_1 \circ \varphi_1 \circ \tau_1} * \Gamma^{\nabla_1 \cdot D}\Big(\upsilon_1 \circ \varphi_2 \circ \tau_1\Big)\\ &= \upsilon_1 \circ \varphi_1 \circ \tau_1 * \upsilon_2\upsilon_1 \circ \varphi_2 \circ \tau_1\tau_2\end{aligned}$$

and since $\upsilon_2 \in U_1$, according to Corollary 3, there exists a ψ_1 such as

$$\begin{aligned}&= \upsilon_2\upsilon_1 \circ \varphi_1 \circ \tau_1\psi_1 * \upsilon_2\upsilon_1 \circ \varphi_2 \circ \tau_1\tau_2\\ &= \upsilon_2\upsilon_1 \circ \Big(\varphi_1 \circ \tau_1\psi_1 * \varphi_2 \circ \tau_1\tau_2\Big)\end{aligned}$$

and since ψ_1 (resp. τ_2) acts only on φ_1's (resp. φ_2) scope

$$= \upsilon_2\upsilon_1 \circ (\varphi_1 * \varphi_2) \circ \tau_1\psi_1\tau_2$$

We exhibited a set of variables and values permutations leading from φ to $\Gamma(\varphi)$; henceforth $\Gamma(\varphi)$ is in the orbit of φ. Let's now consider $\dot\varphi \ll \Gamma(\varphi)$ an element of the orbit of φ and state a few fundamental facts:

(1) $\gamma^D(\varphi_1)$ is the lowest element the orbit of φ_1, namely $U \circ \varphi_i \circ T$, where U and T are groups of value and variable permutations inferred from D and V respectively;

(2) similarly, $\gamma^{\nabla_1 \cdot D}(\upsilon_1 \circ \varphi_2 \circ \tau_1)$ is the lowest element in the orbit $U_1 \circ (\upsilon_1 \circ \varphi_2 \circ \tau_1) \circ T$; or in $U_1 \circ (\upsilon_1 \circ \varphi_2) \circ T$ because τ_1 does not act on φ_2;

(3) $\dot\varphi$ should have the form

$$\dot\varphi = \upsilon \circ \varphi \circ \tau = \upsilon \circ (\varphi_1 * \varphi_2) \circ \tau = (\upsilon \circ \varphi_1 \circ \tau) * (\upsilon \circ \varphi_2 \circ \tau)$$

where $\upsilon \in U$ and $\tau \in T$, and thus

$$\upsilon \circ \varphi_1 \circ \tau = \gamma^D(\varphi_1) = \upsilon_1 \circ \varphi_1 \circ \tau_1 = \upsilon_2\upsilon_1 \circ \varphi_1 \circ \tau_1\psi_1$$

must hold to make $\dot\varphi$ lexicographically smaller than $\Gamma(\varphi)$.

Then, since $\dot\varphi \neq \Gamma(\varphi)$, we have $\upsilon \neq \upsilon_2\upsilon_1 \vee \tau \neq \tau_1\psi_1\tau_2$.

(1) If $\upsilon \neq \upsilon_2\upsilon_1$, then we name $\dot\upsilon = \upsilon\upsilon_1^{-1} \neq \upsilon_2$:

(a) if $\dot\upsilon \notin U_1$, then there exists some $\dot\psi \in T$ such that $\dot\varphi_1 = \upsilon \circ \varphi_1 \circ \tau = \dot\upsilon\upsilon_1 \circ \varphi_1 \circ \tau_1\dot\psi$, but it cannot make $\dot\varphi_1 = \upsilon_1 \circ \varphi_1 \circ \tau_1 = \gamma^D(\varphi_1)$ because of the only-if part of Corollary 3. Therefore $\dot\varphi_1 \neq \gamma^D(\varphi_1)$ and thus $\dot\varphi_1 \gg \gamma^D(\varphi_1)$.

(b) If $\dot\upsilon \in U_1$, $\dot\varphi_1$ can be equal to $\gamma^D(\varphi_1)$, but in that case, $\dot\varphi_2 = \dot\upsilon\upsilon_1 \circ \varphi_2 \circ \tau \gg \upsilon_2\upsilon_1 \circ \varphi_2 \circ \tau_1\tau_2 = \gamma^{\nabla_1 \cdot D}(\varphi_2)$ because both of them are in the orbit $U_1 \circ (\upsilon_1 \circ \varphi_2) \circ T$, $\dot\upsilon \neq \upsilon_2$ and $\gamma^{\nabla_1 \cdot D}(\varphi_2)$ is the lowest element in the orbit.

(2) Only $\upsilon = \upsilon_2\upsilon_1$ is hence valid, leading to $\tau = \dot\tau_1\dot\tau_2$ with $\dot\tau_1 \neq \tau_1\psi_1 \vee \dot\tau_2 \neq \tau_2$:

(a) $\dot\tau_1 \neq \tau_1\psi_1$ is impossible, because $\dot\varphi_1 = \upsilon_2\upsilon_1 \circ \varphi_1 \circ \dot\tau_1$ is in $U \circ \varphi_1 \circ T$ where $\gamma^D(\varphi_1)$ is the lowest element and thus $\dot\varphi_1 \gg \gamma^D(\varphi_1)$

(b) therefore, $\dot\tau_2 \neq \tau_2$, leads us to $\dot\varphi_2 = \upsilon_2\upsilon_1 \circ \varphi_2 \circ \tau_1\dot\tau_2 \gg \gamma^{\nabla_1 \cdot D}(\varphi_2)$ because $\dot\varphi_2 \in U_1 \circ \varphi_2 \circ T$.

$\Gamma^D(\varphi)$ is then necessarily the lowest in lexicographic order in the orbit of φ.

The proof with $\varphi_1 * \cdots * \varphi_n$ can then be naturally induced from the $\varphi_1 * \varphi_2$ case.

□

For example, let $V = \{\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{x_6\}\}$ and $D = \{\{1, 2, 3, 4, 5\}\}$. We want to compute $\gamma(4, 5, 3, 5, 5, 4)$. We start decomposing φ:

$$\varphi = \begin{cases} \varphi_1 &= \{1 \mapsto 4, 2 \mapsto 5, 3 \mapsto 3\} \\ \varphi_2 &= \{4 \mapsto 5, 5 \mapsto 5\} \\ \varphi_3 &= \{6 \mapsto 4\} \end{cases}$$

4.1.1 Computing $\dot\varphi_1$

We consider the resolution of $\gamma^D \cdot \varphi_1$ with $V_1 = \{x_1, x_2, x_3\}$ and $D = \{\{1, 2, 3, 4, 5\}\}$ according to previous sections.

$$\upsilon_1 \circ \varphi_1 \circ \tau_1 : \begin{cases} 1 \longmapsto 4 \\ 2 \longmapsto 5 \\ 3 \longmapsto 1 \\ 4 \longmapsto 2 \\ 5 \longmapsto 3 \end{cases} \circ \begin{cases} 1 \longmapsto 4 \\ 2 \longmapsto 5 \\ 3 \longmapsto 3 \end{cases} \circ \begin{cases} 1 \longmapsto 3 \\ 2 \longmapsto 1 \\ 3 \longmapsto 2 \end{cases} = \begin{cases} 1 \longmapsto 1 \\ 2 \longmapsto 2 \\ 3 \longmapsto 3 \end{cases}$$

We have $\dot\alpha_1 = (1, 2, 3)$. We also compute $\nabla_1 \cdot D = \{\{1, 2, 3\}, \{4, 5\}\}$.

4.1.2 Computing $\dot\varphi_2$

We consider the resolution of $\gamma^{\nabla_1 \cdot D} \cdot \gamma^D \cdot \varphi_2 = \gamma^{\nabla_1 \cdot D}(3, 3)$ with $V_2 = \{x_4, x_5\}$ and $\nabla_1 \cdot D = \{\{1, 2, 3\}, \{4, 5\}\}$

$$\upsilon_2 \circ \left(\gamma^D \cdot \varphi_2\right) \circ \tau_2 : \left\{ \begin{array}{ccc} 1 & \longmapsto & 2 \\ 2 & \longmapsto & 3 \\ 3 & \longmapsto & 1 \end{array} \right. \circ \left\{ \begin{array}{ccc} 4 & \longmapsto & 3 \\ 5 & \longmapsto & 3 \end{array} \right. \circ \left\{ \begin{array}{ccc} 4 & \longmapsto & 4 \\ 5 & \longmapsto & 5 \end{array} \right. = \left\{ \begin{array}{ccc} 4 & \longmapsto & 1 \\ 5 & \longmapsto & 1 \end{array} \right.$$

We have $\dot{\alpha}_2 = (1, 1)$. We then compute $\nabla_2 \cdot \nabla_1 \cdot D = \{\{1\}, \{2, 3\}, \{4, 5\}\}$.

4.1.3 Computing $\dot{\varphi}_3$

We consider the resolution of $\gamma^{\nabla_2 \cdot \nabla_1 \cdot D} \cdot \gamma^{\nabla_1 \cdot D} \cdot \gamma^D \cdot \varphi_3 = \gamma^{\nabla_2 \cdot \nabla_1 \cdot D}(3)$ with $V_3 = \{x_6\}$ and $\nabla_2 \cdot \nabla_1 \cdot D = \{\{1\}, \{2, 3\}, \{4, 5\}\}$.

$$\upsilon_3 \circ \left(\gamma^{\nabla_1 \cdot D} \cdot \gamma^D \cdot \varphi_3\right) \circ \tau_3 : \left\{ \begin{array}{ccc} 2 & \longmapsto & 3 \\ 3 & \longmapsto & 2 \end{array} \right. \circ \left\{ \begin{array}{ccc} 6 & \longmapsto & 3 \end{array} \right. \circ \left\{ \begin{array}{ccc} 6 & \longmapsto & 6 \end{array} \right. = \left\{ \begin{array}{ccc} 6 & \longmapsto & 2 \end{array} \right.$$

We have $\dot{\alpha}_3 = (2)$. The solution is $\boxed{\gamma(4, 5, 3, 5, 5, 4) = (1, 2, 3, 1, 1, 2)}$.

4.2 List of orbits

Similarly, we can list orbits by considering all the V_i in order and by dividing the set of sets of permutable values with the ∇ operator. We recursively build a tree as deep as n, number of V_i.

For example, see Figure 1 for an example with $V = \{\{x_1, x_2\}, \{x_3\}, \{x_4\}\}$ and $D = \{\{1, 2, 3\}\}$. The figure represents a tree where each leaf is a possible orbit for the symmetry group.

5. Experimental results

5.1 Orbits' listing problem

We tried here to compare the performance of our algorithm to the naive approach. We generated randomly 200 problems consisting of sets of permutable variables' sets and sets of permutable values' sets and listed all the canonical forms for each orbit.

We traced in Figure 2 an average of the execution time for both our method and the first naive method in function of the size of the problem, measured here as the sum of the number of variables and of the number of values. We used a Java 6 Sun environment on a Core2Duo based Linux with 2GB RAM.

Our method offers an outstanding speedup compared to the naive method, standing between 25 and 300,000 for large problems.

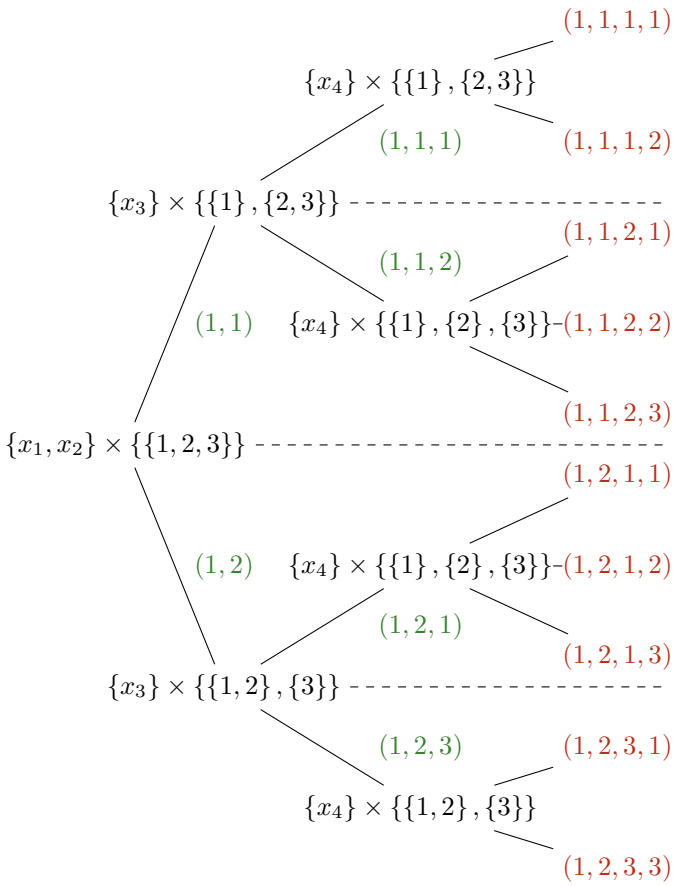

Fig. 1 List of orbits for $V = \{\{x_1, x_2\}, \{x_3\}, \{x_4\}\}$ and $D = \{\{1, 2, 3\}\}$.

5.2 Application to distributed constraint programming

Distributed constraint programming is a paradigm where a number of agents own variables linked together by constraints. Agents are separate computing units, with private and public data: private information is not to be communicated with other agents, and public information is shared by two or more agents. The challenge is to get a consistent solution, valid for all agents.

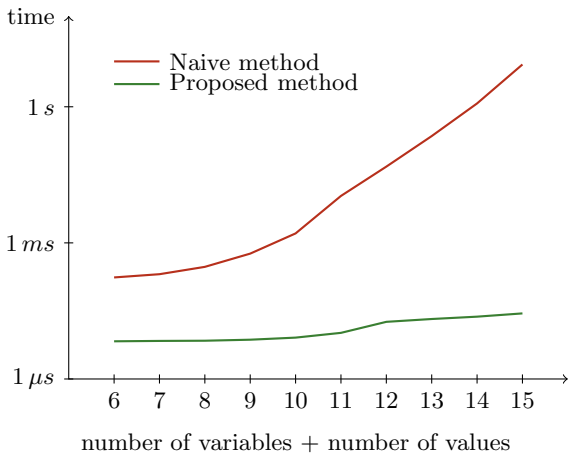

Fig. 2 Execution time for the proposed algorithm: the speedup gets as big 300,000 for large problems.

Symmetry breaking being an efficient technique to improve constraint programming[4], the authors studied several ways to exploit those symmetries in a distributed context[7),8)]: 9) focuses on methods to improve the DPOP algorithm[11)], a distributed version of the bucket elimination method. This method is based on the idea that a constraint to be propagated is represented extensively, i.e., as an evaluation function of the set of assignations. An hypercube is an object listing the variables, their domains and the evaluation function of every single assignation. As two symmetrical assignations have the same constraint evaluation, we proposed a sparse version of this hypercube to keep only one assignation per symmetry orbit and cut the constraint representation size. On the other hand, 10) is another method based on symmetry breaking by dominance detection[4)] but in a distributed environment, using SynchBB[6)] instead of Branch & Bound.

6. Conclusion

We presented in this paper a very efficient method to solve an extension of the constructive orbit problem, showing a speedup ranging from 25 to 300,000. We extended Donaldson's work[3)] focusing on model checking (and transposable to variable symmetries), to the context of constraint programming with both variable and value symmetries. We also exhibited promising results in distributed constraint programming, where the detection/exploitation of symmetry is a challenging issue considering the distribution of data. The efficient way to list orbits and compute γ lets us reduce the data volume dramatically resulting in a speedup going up to two-fold even in an environment where communication remains rather cheap.

References

1) Babai, L. and Luks, E.: Canonical labeling of graphs, *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, ACM, pp.171–183 (1983).
2) Clarke, E., Emerson, E.A., Jha, S. and Sistla, A.P.: Symmetry reductions in model checking, *Computer Aided Verification*, Springer, pp.147–158 (1998).
3) Donaldson, A.F. and Miller, A.: On the constructive orbit problem, *Annals of mathematics and artificial intelligence*, Vol.57, pp.1–35 (2009).
4) Gent, I., Petrie, K. and Puget, J.-F.: Symmetry in Constraint Programming, *Handbook of constraint programming* (Rossi, F., van Beek, P. and Walsh, T., eds.), Elsevier Science Ltd, chapter10, pp.329–376 (2006).
5) Hirayama, K., Yeoh, W. and Zivan, R.(eds.): *Proceedings of the 11th International Workshop on Distributed Constraint Programming* (2009).
6) Hirayama, K. and Yokoo, M.: Distributed Partial Constraint Satisfaction Problem, *Principles and Practice of Constraint Programming*, pp.222–236 (1997).
7) Olive, X. and Nakashima, H.: Breaking Symmetries in Distributed Constraint Programming Problems, Hirayama et al.[5)], pp.165–169.
8) Olive, X. and Nakashima, H.: SymDPOP: Adapting DPOP to exploit partial symmetries, *Proceedings of the 12th International Workshop on Distributed Constraint Programming* (van der Hoek, W., Kaminka, G.A., Lespérance, Y., Luck, M. and Sen, S., eds.), pp.38–52 (2010).
9) Olive, X. and Nakashima, H.: Efficient Representation of Constraints and Propagation of Variable–Value Symmetries in Distributed Constraint Reasoning, *Journal of Information Processing*, Vol.19 (2011).
10) Olive, X. and Nakashima, H.: Symmetry Breaking by Dominance Detection in Distributed Environments, *IPSJ SIG Notes*, Vol.2011-AL-134 (2011).
11) Petcu, A. and Faltings, B.: A Scalable Method for Multiagent Constraint Optimization, *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, Vol.19, p.266 (2005).