Symmetry Breaking in Subgraph Pattern Matching

Stéphane Zampelli, Yves Deville, and Pierre Dupont

Université catholique de Louvain,
Department of Computing Science and Engineering,
2, Place Sainte-Barbe
1348 Louvain-la-Neuve (Belgium)
{sz,yde,pdupont}@info.ucl.ac.be

Abstract

Graph pattern matching, a central application in many fields, can be modelled as a CSP. This CSP approach can be competitive with dedicated algorithms. In this paper, we develop symmetry breaking techniques for subgraph matching in order to increase the number of tractable instances of this problem. Specific detection techniques are first developped for the classical variables symmetries and value symmetries. It is also shown how these symmetries can be broken when solving graph matching. We also show how conditional value symmetries can be automatically detected and handled in the search process. Then, the concept of local value symmetries is introduced ; it is shown how these new symmetries can be computed and exploited. Finally, experimental results show that symmetry breaking is an effective way to increase the number of tractable instances of the graph matching problem.

1 Introduction

A symmetry in a Constraint Satisfaction Problem (CSP) is a bijective function that preserves CSP structure and solutions. Symmetries are important because they induce symmetric subtrees in the search tree. If the instance has no solution, failure has to be proved for equivalent subtrees regarding symmetries. If the instance has solutions, many symmetric solutions will have to be enumerated in symmetric subtrees. The detection and breaking of symmetries can thus speed up the solving of a CSP.

Symmetries arise naturally in graphs since a set of bijective function can be viewed as the automorphism group of a graph. However, althought a lot of graph problems have been tackled [1] [2] [25] and a computation domain for graphs has been defined [8], and despite the fact that symmetries and graphs are related, little has been done to investigate the use of symmetry breaking for graph problems in constraint programming.

This paper aims at applying and extending symmetries techniques for subgraph matching. Existing techniques usually handle only initial symmetries and are not able to detect symmetries arising during search, so called conditional symmetries. We will show how to detect and handle those conditional symmetries.

Related Works Handling symmetries to reduce search space has been a subject of research in constraint programming for many years. Crawford and al. [6] showed that computing the set of predicates breaking the symmetries of an instance is NP-hard in general. Different approaches exist for exploiting symmetries. Symmetries can be broken during search either by posting additional constraints (SBDS) [14] [12] or by pruning the tree below a state symmetrical to a previous one (SBDD) [13]. Symmetries can be broken by taking into account the symmetries into the heuristic [18] . The main idea is to select the variable involved in the greatest number of symmetries local to the current state, so that symmetries are broken as soon as possible by the heuristic. Symmetries can be broken by adding constraints to the initial problem at its root node [6] [11]. Symmetries can also be broken by remodelling the problem [26].

More recently, research efforts has been done towards defining, detecting and breaking symmetries. Cohen and al. [4] defined two types of symmetries, solution symmetries and constraint symmetries and proved that the group of constraint symmetries is a subgroup of solution symmetries. Moreover, Gent and al. [10] evaluated several techniques to break conditional symmetries, that is symmetries arising during search. However the detection of conditional symmetries remains a research topic. Symmetries were also shown to produce stronger forms of consistency and more efficient mechanisms for establishing them [9]. Finally, Puget [22] showed how to detect symmetries automatically, and showed that all variable symmetries could be broken with a linear number of constraints for injective problems [21] and all value symmetries can be broken for surjective problems, by adding one variable per value of the problem plus a linear number of binary constraints [20].

Graph pattern matching is a central application in many fields [5]. Many different types of algorithms have been proposed, ranging from general methods to specific algorithms for particular types of graphs. In constraint programming, several authors [15, 24] have shown that graph matching can be formulated as a CSP problem, and argued that constraint programming could be a powerful tool to handle its combinatorial complexity. Within the CSP framework, a model for subgraph monomorphism has been proposed by Rudolf [24] and Valiente et al. [15]. Our modeling [29] is based on these works. Sorlin [27] proposed a filtering algorithm based on paths for graph isomorphism and part of our approach can be seen as a generalization of this filtering. A declarative view of matching has also been proposed in [16]. In [29], we showed that CSP approach is competitive with dedicated algorithms over a graph database representing graphs with various topologies.

Objectives This work aims at developping symmetry breaking techniques for subgraph matching modelled as a CSP in order to increase the number of tractable instances of graph matching. Our first goal is to develop specific detection techniques for the classical variables symmetries and value symmetries, and to break such symmetries when solving the graph matching. Our second goal is to develop more advanced symmetries that can be easily detected for graph matching.

Results

- We show that variable symmetries and value symmetries can be detected by computing the set of automorphisms on the pattern graph and on the target graph.
- We show that conditional value symmetries can be detected by computing the set of automorphisms on various subgraphs of the target graph, called dynamic target graphs. The GE-Tree method can be extended to handle these conditional symmetries.
- We introduce the concept of local value symmetries, that is symmetries on a subproblem. It is shown how such new symmetries can be computed and exploited using standard methods such as GE-Tree.
- Experimental results compare and analyse the enhancement achieved by these symmetries and show that symmetry breaking is an effective way to increase the number of tractable instances of the graph matching problem.

Outline Sections 2 provides the necessary background in graph matching and in symmetry breaking. Section 3 describes a CSP approach for graph matching. Sections 3 and 4 present variable symmetries and value symmetries in graph matching. Conditional value symmetries are handled in Section 6, and Section 7 introduces local value symmetries in graph matching. Finally, Section 8 describes experimental results and Section 9 concludes this paper.

2 Background and Definitions

2.1 Graph matching

Before presenting the basic CSP for subgraph matching, we define the notion of subgraph matching.

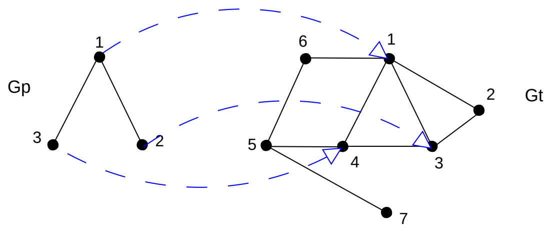

FIG. 1 – Example solution for a monomorphism problem instance.

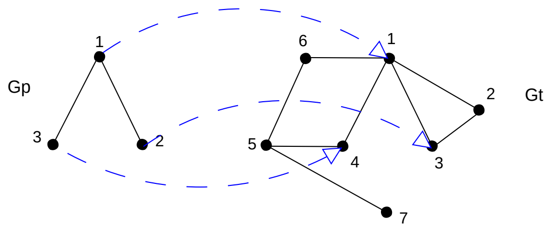

FIG. 2 – Example solution for an isomorphism problem instance.

A **graph** $G = (N, E)$ consists of a **node set** N and an **edge set** $E \subseteq N \times N$, where an edge (u, v) is a pair of nodes. The nodes u and v are the endpoints of the edge (u, v). We consider directed and undirected graphs.

A **subgraph** of a graph $G = (N, E)$ is a graph $S = (N', E')$ where N' is a subset of N and E' is a subset of E.

A **subgraph isomorphism** between a pattern graph $G_p = (N_p, E_p)$ and a target graph $G_t = (N_t, E_t)$ is a total function $f : N_p \to N_t$ respecting two conditions :

1. the function f is injective
2. f is an isomorphism : $(u, v) \in E_p \Leftrightarrow (f(u), f(v)) \in E_t$.

A **subgraph monomorphism** between G_p and G_t is a total function $f : N_p \to N_t$ respecting two conditions :

1. the function f is injective
2. f is a monomorphism : $(u, v) \in E_p \Rightarrow (f(u), f(v)) \in E_t$.

A **subgraph matching** is either a subgraph isomorphism or a subgraph monomorphism.

The **neighborhood function** $V : N \to N$ is defined as $V(i) = \{j \mid (i, j) \in E\}$. We note V_p for the neighborhood function of the pattern graph and V_t for the neighborhood function of the target graph.

In this paper, we focus on symmetries in subgraph monomorphism.

2.2 Symmetries

A CSP instance is a triple $< X, D, C >$ where X is the set of variables, D is the universal domain specifying the possible values for those variables, and C is the set of constraints. In the rest of this document, $n = |N_p|$, $d = |D|$, and $D(x_i)$ is the domain of x_i.

A symmetry over a CSP instance P is a bijection σ mapping solutions to solutions, and hence non solutions to non solutions [22].

Since a symmetry is a bijection where domain and target sets are the same, a symmetry is a permutation. For instance, the permutation $(a_1a_2)(b_1b_2b_3)$ is the bijection $\sigma(a_1) = a_2, \sigma(a_2) = a_1, \sigma(b_1) = b_2, \sigma(b_2) = b_3$, $\sigma(b_3) = b_1$ and $\sigma(c) = c$ otherwise.

A *variable symmetry* is a bijective function $\sigma : X \rightarrow X$ permuting a (non) solution $s = ((x_1, d_1), \ldots, (x_n, d_n))$ to a (non) solution $s' = ((\sigma(x_1), d_1), \ldots, (\sigma(x_n), d_n))$. For instance, the constraint $x + y = 5$ implies the variable symmetry $(x\ y)$.

A *value symmetry* is a bijective function $\sigma : D \rightarrow D$ permuting a (non) solution $s = ((x_1, d_1), \ldots, (x_n, d_n))$ to a (non) solution $s' = ((x_1, \sigma(d_1)), \ldots, (x_n, \sigma(d_n))$. For instance, the constraint x mod $3 = 2$ implies the value symmetry $(8\ 5)$.

A *value and variable symmetry* is a bijective function $\sigma : X \times D \rightarrow X \times D$ permuting a (non) solution $s = ((x_1, d_1), \ldots, (x_n, d_n))$ to a (non) solution $s' = (\sigma(x_1, d_1), \ldots, \sigma(x_n, d_n))$. For instance, consider the CSP $D(x) = [1, 2, 4]$, $D(y) = [1, 3, 4]$, $D(z) = [4]$, $x + y = 5$, $y \leq z$. The set of solutions is $\{(x,1),(y,4),(z,4)\}$, $\{(x,4),(y,1),(z,4)\}$, $((x,2),(y,3),(z,4))\}$. A value and variable symmetry is $((x,1)\ (x,4))$, $((y,4)\ (y,1))$. Note that $(x\ y)$ is not a variable symmetry and $(1\ 4)$ is not a value symmetry.

A *conditional symmetry* of a CSP P is a symmetry holding holds only in a sub-problem $P^{'}$ of P. The conditions of the symmetry are the constraints necessary to generate $P^{'}$ from P [10].

A *group* is a finite or infinite set of elements together with a binary operation (called the group operation) that together satisfy the four fundamental properties of closure, associativity, the identity property, and the inverse property. An *automorphism of a graph* is a graph isomorphism with itself. The sets of automorphisms $Aut(G)$ define a finite permutation group.

Handling symmetries consists in three steps : symmetry detection, breaking the symmetry to reduce search space, and generating the set of all solutions.

2.3 Goal of symmetry breaking

The general goal of symmetry breaking is to find a subset of canonical solutions [19].

Without loss of generality, we may apply an arbitrary order upon variables and values. Let $\leq_{lex}$ be an ordering upon vectors representing the solutions. Given G the symmetry group and Sol the set of solutions, the subset $BSol$ of canonical solutions is defined as :

$$BSol = \{s \in Sol \mid s \leq_{lex} \sigma(s)\ \ \forall\, \sigma \in G\}.$$

The solutions Sol can be generated by applying the elements of G to $BSol$:

$$Sol = \{\sigma(s) \mid \sigma \in G \wedge s \in Bsol\}.$$

3 CSP approach for graph matching

The CSP model of graph matching should represent a total function $f : N_p \rightarrow N_t$. This total function can be modeled with $X = x_1, ..., x_n$ with x_i representing the ith node of G_p and $D = N_t$. Thus the set of variables is the set of pattern nodes and their initial domain is the set of target nodes.

The injection constraint can be stated by using `alldiff`$(x_1, ..., x_n)$.

Conditions on the function for monomorphism have to be translated into constraints.

The monomorphism constraint states that if an edge exists between two pattern nodes, then an edge must exist between their corresponding images :

$$\forall\ (i, j) \in E_p : (f(i), f(j)) \in E_t\ .$$

For each $(i, j) \in E_p$, the corresponding basic monomorphism constraint is defined as :

$$MC(x_i, x_j) \equiv (x_i, x_j) \in E_t\ .$$

A global constraint $MC(x_1, ..., x_n)$ can be formulated, instead of having one constraint MC per node pair :

$$MC(x_1, ..., x_n) = \bigwedge_{(i,j)\in E_p} MC(x_i, x_j)\ .$$

Moreover, a redundant constraint pruning the search space has been proposed in [15]. This constraint reduces the search time for difficult instances. This redundant constraint is a local Alldiff constraint [23] upon the neighborhood of a node, by noting that the number of candidates available in the union of x_i neighbors domain could not be less than the actual number of x_i neighbors in the pattern graph :

$$LA(x_i) \equiv |\cup_{j\in V_p(i)} D(x_j) \cap V_t(x_i)| \geq |V_p(i)|\ .$$

An algorithmic global constraint $LA(x_1, ..., x_n)$ can be formulated :

$$LA(x_1, ..., x_n) \equiv \bigwedge_i LA(x_i)\ .$$

For the monomorphism problem, the following constraints of the corresponding CSP are :

`alldiff`$(x_1, ...x_n)$, `MC`$(x_1, ..., x_n)$ and `LA`$(x_1, ..., x_n)$.

Implementation, comparison with dedicated algorithms, and extension to subgraph isomorphism can be found in [29]. Extension of this framework for approximate matching using graph and function domain computation has been introduced in [7].

4 Variable Symmetries

4.1 Detection

This section shows that, in graph matching, variable symmetries are the automorphisms of the pattern graph and do not depend on the target graph.

It has been shown that the set of variable symmetries of the CSP is the automorphism group of a *symbolic graph* [22] . The automorphism group of this symbolic graph is the set of symmetries of the constraint. The final symbolic graph is obtained by merging nodes playing the same role in the different symbolic graphs. The automorphism group can be computed by using tools such as NAUTY [17]. Those tools output a set of generators of the group useful for breaking symmetries.

We will apply those ideas to the pattern graph, representing the symbolic graph of the constraint network of the CSP. The pattern G_p is transformed into a symbolic graph $S(G_p)$ where $Aut(S(G_p))$ is the set of variable symmetries of the CSP.

Definition 1 *A CSP P modeling a subgraph monomorphism instance (G_p, G_t) can be transformed into the following symbolic graph $S(P)$:*

1. *Each variable x_i is a distinct node labelled i*
2. *If there exists a constraint $MC(x_i, x_j)$, then there exists an arc between i and j in the symbolic graph*
3. *The constraint alldiff, as suggested in [22], is transformed into a node typed with label 'a'; an arc (a, x_i) is added to the symbolic graph*

Because LA constraints are redundant, they do not modify the set of solutions, hence they do not modify the set of variable symmetries of P. The constraint LA can be safely omitted in the symbolic graph.

If we do not consider the extra node and arcs introduced by the alldiff constraint, then the symbolic graph $S(P)$ and G_p are isomorphic by construction.

Given the labeling of nodes representing constraints, an automorphism in $S(P)$ maps the alldiff node to itself and the nodes corresponding to the variables to another node corresponding to the variables. Each automorphism in $Aut(G_p)$ will thus be a restriction of an automorphism in $Aut(S(P))$, and an element in $Aut(S(P))$ will be an extension of an element in $Aut(G_p)$. Hence the two following theorems.

Theorem 1 *Given a subgraph monomorphism instance (G_p, G_t) and its associated CSP P :*

- $\forall\, \sigma \in Aut(G_p)\ \exists\, \sigma' \in Aut(S(P))$:
  $\forall\, n \in N_p : \sigma(n) = \sigma'(n)$
- $\forall\, \sigma' \in Aut(S(P))\ \exists\, \sigma \in Aut(G_p)$:
  $\forall\, n \in N_p : \sigma(n) = \sigma'(n)$

Theorem 2 *Given a subgraph monomorphism instance (G_p, G_t) and its associated CSP P, the set of variable symmetries of P is the set of bijective functions $Aut(S(P))$ restricted to N_p, which is equal to $Aut(G_p)$.*

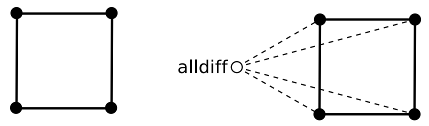

FIG. 3 – Example of symbolic graph for a square pattern.

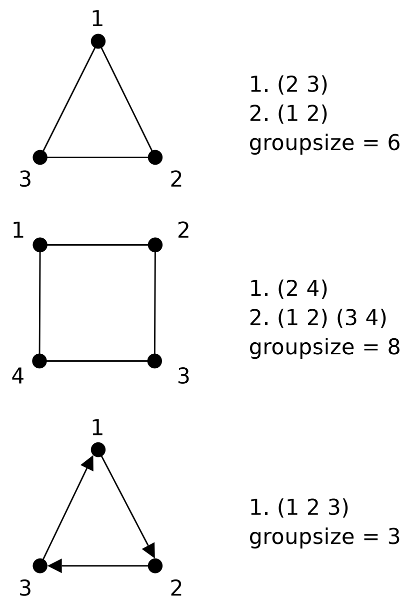

FIG. 4 – Example of pattern graphs and their generators.

Theorem 2 says that only $Aut(G_p)$ has to be computed in order to get all variable symmetries.

Figure 3 shows a pattern transformed into its symbolic graph.

Figure 4 gives some example of variable symmetries for different patterns. For each pattern graph, the list of generators and the size of the whole group are given. The undirected triangle graph has two generators $(2\ 3)$ and $(1\ 2)$ and $3!$ automorphisms (with e being the identity function) :

1. $\sigma_1 = \begin{pmatrix} 2 & 3 \end{pmatrix}$
2. $\sigma_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}$
3. $\sigma_1 \cdot \sigma_2 = \sigma_1\sigma_2 = \begin{pmatrix} 1 & 3 & 2 \end{pmatrix}$
4. $\sigma_2 \cdot \sigma_1 = \sigma_2\sigma_1 = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$
5. $\sigma_1 \cdot \sigma_2\sigma_1 = \sigma_1\sigma_2\sigma_1 = e$
6. $\sigma_2 \cdot \sigma_1\sigma_2 = \sigma_2\sigma_1\sigma_2 = \begin{pmatrix} 1 & 3 \end{pmatrix}$

The automorphism group of the square undirected graph, known as D_4, has two generators and 8 automorphisms. The directed triangle has one generator and 3 automorphisms : $\{\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 3 & 2 \end{pmatrix}, e\}$.

4.2 Breaking

Two techniques were selected to break variable symmetries. The first technique is an approximation and consist in breaking only the generators of symmetry group [6]. Those generators are obtained by using a tool such as NAUTY, that outputs the generator of the symmetry group. For each variable symmetry σ, an ordering constraint is posted to keep only canonical solutions. Since $s \leq \sigma s \Leftrightarrow ((x_1, v_1), \cdots, (x_n, v_n)) \leq ((\sigma(x_1), v_1), \cdots, (\sigma(x_n), v_n))$, a constraint $x_1 \leq \sigma(x_1)$ is posted to respect the lexicographic ordering.

The second technique breaks all variable symmetries of a injective problem by using a SchreierSims algorithm, provided that the generators of the variable symmetry group are known [22]. In an injective problem such as subgraph matching, Puget showed the number of constraints to be posted is linear with the number of variables. The Schreier-Sims algorithm is an efficient method of computing a base and strong generating set of a permutation group. It takes generators as input and runs in $O(n^2 log^3|G| + t.n.log|G|)$ where G is the group, t the number of generators and n the size of the of group of all permutations containing G. The strong generating set output is precisely the information needed to post the non redundant breaking symmetry constraints.

These two techniques will be compared in the experimental results section.

5 Value Symmetries

5.1 Detection

In graph matching, value symmetries are automorphisms of the target graph and do not depend on the pattern graph.

Theorem 3 *Given a subgraph monomorphism instance (G_p, G_t) and its associated CSP P, each $\sigma \in Aut(G_t)$ is a value symmetry of P.*

Proof Suppose $Sol = (v_1, \cdots, v_n)$ is a solution. Consider the subgraph $G = (N, E)$ of G_t, where $N = \{v_1, \cdots, v_n\}$ and $E = \{(i, j) \mid (\sigma^{-1}(i), \sigma^{-1}(j)) \in E_p\}$. This means there exists a monomorphic function $f^{'}$ matching G_p to σG. Hence $((x_1, \sigma(v_1)), \cdots, (x_n, \sigma(v_n)))$ is a solution.

■

All value symmetries of P are not in $Aut(G_t)$. Consider Figure 5. There exists two value symmetric solutions : $\{(x_1, 1), (x_2, 2), (x_3, 3), (x_4, 4)\}$ and $\{(x_1, 2), (x_2, 1), (x_3, 4), (x_4, 3)\}$ althought $Aut(G_t) = \emptyset$.

Figure 6 gives an example of a value symmetry on the target graph. There is only one generator for this graph : $(1\ 2)$. Suppose the pattern graph is a path of length 2 : $x_1 \rightarrow x_2 \rightarrow x_3$. Suppose $(1, 3, 2)$ is a solution. Then $(2, 3, 1)$ is also a solution. Suppose $(1, 3, 4)$ is a solution. Then $(2, 3, 4)$ is also a solution.

5.2 Breaking

Breaking inital value symmetries can be done by using GE-Tree technique [3]. The idea is to modify the

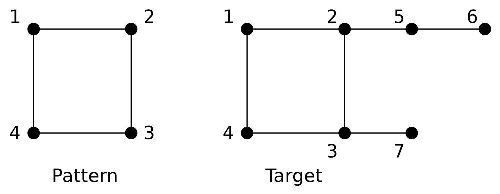

FIG. 5 – Example of matching where the set of value symmetries is not empty and $Aut(G_t) = \emptyset$.

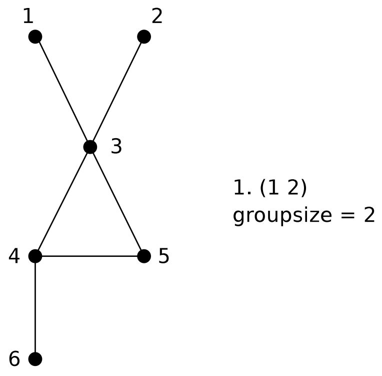

FIG. 6 – Example of value symmetry on the target graph.

distribution by avoiding symmetrical value assignment. Suppose a state S is reached, where $x_1, \cdots, x_k$ are assigned to $v_1, \cdots, v_k$ respectively, and $x_{k+1}, \cdots, x_n$ are not assigned yet. The variable x_{k+1} should not be assigned to two symmetrical values, since two symmetric subtrees would be searched. For each value $v_i \in D(v_{k+1})$ that is symmetric to a value $v_j \in D(v_{k+1})$, only one state S_1 should be generated with the new constraint $x_{k+1} = v_i$; no new state S_2 with $x_i = v_j$ should be generated.

A convenient way to compute those symetrical values is to compute a base and a strong generating set by the SchreierSims. Algorithm SchreierSims outputs the subgroups of $Aut(G_t)$ G_i $(1 \leq i \leq d)$ such that $\forall\, \sigma \in G_i : \sigma(j) = j\ \forall\, j \in [1, i]$ (called the pointwize stabilizators of G). Moreover SchreierSims outputs the set of images of i that let $0, \cdots, i$ invariant : $U_{i+1} = (i+1)^{G_{i+1}}$. Those sets U_i are interesting because they give the set of symmetrical values of i given that the values $1, ..., i$ are not subject to any permutation (mapped to themselves).

In order to use those U_i, the values are assigned in an increasing order, so that the hypothesis that $1, \cdots, i$ is not subject to any permutation is ensured. Suppose a state S is reached, $x_1, \cdots, x_k$ are assigned to $v_1, \cdots, v_k$ respectively, with $v_1 \leq \cdots \leq v_k$ and $v_i \leq v_j\ \forall\, i \in [1, k]\ \forall\, j \in [k+1, d]$. The variables $x_{k+1}, \cdots, x_n$ are not assigned yet. The next value $v_{k+1} \in D(x_j)$ is selected in the increasing ordering and is assigned to x_j. We create two new states S_1 and S_2. The constraint $x_{k+1} = v_{k+1}$ is posted in S_1 and the constraints $x_{k+1} \neq v_{k+1}$ and

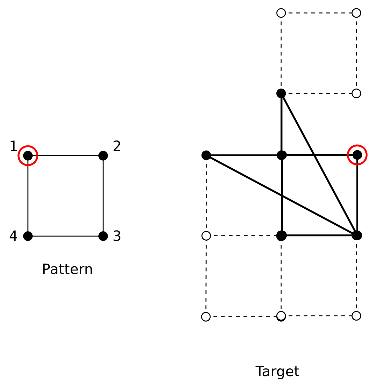

FIG. 7 – Example of dynamic target subgraph.

$x_{k+1} \neq k^{'} \; \forall \; k^{'} \in U_k$. The value symmetries in the state S_2 have been deleted for x_{k+1}.

6 Conditional Value Symmetries

In subgraph monomorphism, the relations between values are explicitly represented in the target graph. This allows the detection of conditional values symmetries.

6.1 Detection

During the search, the target graph looses a node a whenever $a \notin \cup_{i \in N_p} D(x_i)$. This is interesting because the relation between the values are known dynamically.

The set of values $\cup_{i \in N_p} D(x_i)$ denotes the nodes of subgraph of G_t in which a solution is searched. For a given state S, such a subgraph can be, for a given state S, computed efficiently. We first define this subgraph of G_t.

Definition 2 *Let S be a state in the search where $x_1, \cdots, x_k$ are assigned, and $x_{k+1}, \cdots, x_n$ are not assigned. The* ***dynamic target graph*** *$G_t^* = (N_t^*, E_t^*)$ is a subgraph of G_t such that :*

- $N_t^* = \cup_{i \in [1, \cdots, n]} D(x_i)$
- $E_t^* = \{(a, b) \in E_t \mid a \in N_t^* \wedge b \in N_t^*\}$

Figure 7 shows an example of dynamic target graph. In this figure, the circled nodes are assigned together. The blank nodes are the nodes excluded from $\cup_{i \in [1, \cdots, n]} D(x_i)$, and the black nodes are the nodes included in $\cup_{i \in [1, \cdots, n]} D(x_i)$. The plain edges are the selected edges for the dynamic target subgraph.

Each automorphism of G_t^* is a conditional value symmetry for the state S.

Theorem 4 *Given a subgraph monomorphism intance (G_p, G_t), its associated CSP P, and a state S in the search, each $\sigma \in Aut(G_t^*)$ is a conditional value symmetry of P. Moreover, the conditions of σ are $x_1 = v_1, \cdots, x_k = v_k$.*

Proof Suppose $Sol = (v_1, \cdots, v_k)$ is a partial solution. Consider the subgraph G_t^*. The state S can be considered as a new CSP $P^{'}$ of an instance (G_p, G_t^*) with additional constraints $x_1 = v_1, \cdots, x_k = v_k$. By Theorem 3, the thesis follows.

■

The size of G_t^* is an important issue, as we will dynamically compute symmetry information with it. The following theorem shows that, because of the MC constraints, the longest path in G_p has the same length than the longest path in G_t whenever at least a variable is assigned.

Definition 3 *Let $G = (N, E)$ be a graph. Then $maxd(G)$ denotes the size of the longest simple path between two nodes $a, b \in N$.*

Theorem 5 *Given a subgraph monomorphism intance (G_p, G_t), its associated CSP P, and a state S in the search, if $\exists \; i \in N_p \mid |D(x_i)| = 1$, then $maxd(G_p) = maxd(G_t^*)$.*

This is a nice result for complexity issues, when $maxd(G_p)$ is small. In Figure 7, $maxd(G_p)$=2 and only nodes at shortest distance 2 from node 1 in the target graph are included in G_t^*.

The dynamic target graph G can be computed dynamically. In [7], we showed how graph matching can be modelled and implemented in CP(Graph), an extension of CP with graph domain variables. In this setting, a graph domain variable T is used for target graph, with initial domain $[\emptyset, \cdots, G_t]$. When a solution is found, T is instantiated to the matched subgraph of G_t. Hence, during the search, the dynamic target graph G_t^* will be the upper bound of variable T and can be obtained in $O(1)$.

6.2 Breaking

In this subsection, we show how to modifiy GE-Tree method to handle conditional value symmetries. Before distribution, the following actions are triggered :

1. Get G_t^*.
2. The NAUTY and SchreierSims algorithms are called. This returns the new $U_i^{'}$ sets.
3. The main problem is how to adapt the variable and value selection such that conditional value symmetries are broken. In GE-Tree, from a given state S, two branches are created :

   (a) a new state S_1 with a constraint $x_k = v_k$

   (b) a new state S_2 with constraints :

      i. $x_k \neq v_k$

      ii. $x_k \neq v_j \; \forall \; j \in U_{k-1}$.

   To handle conditional value symmetries, we slightly modify this schema. From a given state S, two branches are created :

   (a) a new state S_1 with a constraint $x_k = v_k$

(b) a new state S_2 with constraints :
   i. $x_k \neq v_k$
   ii. $x_k \neq v_j \ \forall\, j \in U_{k-1} \cup U'_{k-1}$

An issue is how to handle structure U. In Gecode system (http ://www.gecode.org), in which the actual implementation is made, the states are copied and trailing is not needed. Thus the structure U must not be updated because of backtraking. A single global copy is kept during the whole search process.

In a state S where conditional values symmetries are discovered, structure U is copied into a new structure U'' and merged with U'. This structure U'' shall be used for all states S' having S in its predecessors.

Of course, some heuristics should be added to choose the states where a new conditional value symmetry should be computed.

7 Local Value Symmetries

In this section, we introduce the concept of local value symmetries, that is value symmetries on a subproblem. Such symmetries will be detected and exploited during the search.

7.1 Detection

We first introduce partial dynamic graph concept. Those graphs are associated to a state in the search and corespond to the unsolved part of the problem. This can be viewed as a new local problem to the current state.

Definition 4 *Let S be a state in the search whose variables $x_1, \cdots, x_k$ are assigned to $v_1, \cdots, v_k$ respectively, and $x_{k+1}, \cdots, x_n$ are not assigned yet.*
*The **partial dynamic pattern graph** $G_p^- = (N_p^-, E_p^-)$ is a subgraph of G_p such that :*
- $N_p^- = \{i \in [k+1, d]\}$
- $E_p^- = \{(i,j) \in E_p \mid a \in N_p^- \wedge b \in N_p^-\}$

*The **partial dynamic target graph** $G_t^- = (N_t^-, E_t^-)$ is a subgraph of G_t such that :*
- $N_t^- = \cup_{i\in[k+1,d]} D(x_i)$
- $E_t^- = \{(a,b) \in E_t \mid a \in N_t^- \wedge b \in N_t^-\}$

When forward checking (FC) is used during the search, in any state in the search tree, every constraint involving *one* uninstantiated variable is arc consistent. In other words, every value in the domain of an uninstantiated variable is consistent with the partial solution. This FC property on a binary CSP ensures that one can focus on the uninstantiated variables and their associated constraints without loosing or creating solutions to the initial problem. Such a property also holds when the search achieves stronger consistency in the search tree (Partial Look Ahead, Maintaining Arc Concistency, ...).

Theorem 6 *Let (G_p, G_t) be a subgraph monomorphism intance, P its associated CSP, and S a state in of P during the search, where the assigned variables are $x_1, \cdots, x_k$ with values $v_1, \cdots, v_k$. Let P' be a new CSP of a subgraph monomorphism instance (G_p^-, G_t^-) with additional constraints $x'_{k+1} = D(x_1), \cdots, x'_n = D(x_n)$. Then :*

1. *Each $\sigma \in Aut(G_t^-)$ is a value symmetry of P'.*
2. *Assuming we have the FC property, we have*
   $((x_1, v_1), \cdots, (x_n, v_n)) \in Sol(S)$
   iff
   $((x_{k+1}, v_{k+1}), \cdots, (x_n, v_n)) \in Sol(P')$.

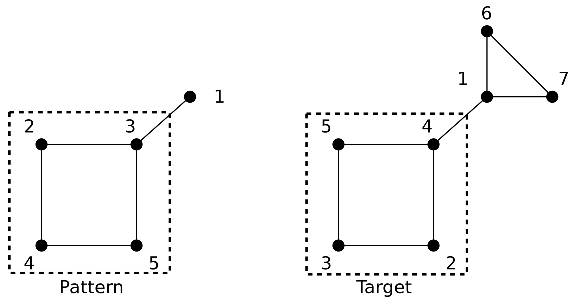

FIG. 8 – Example of conditional local value symmetry. The dashed squares show the new subgraph monomorphism instance for CSP P'.

The theorem states that value symmetries of the local CSP P' can be obtained by computing $Aut(G_t^+)$, and that these symmetries can be exploited without loosing or adding solutions to the initial matching problem.

It is important to notice that the value symmetries of P' are *not* conditional symmetries of P. It is not possible to add constraints to P to generate P'. As the CSP P' is a local CSP associated to a state S, these value symmetries are called *local value symmetries*.

The computation of G_t^- can be easily performed thanks to graph variables. If T is the target graph variable over initial domain $[\emptyset, \cdots, G_t]$, then during the computation G_t^- is $lub(T) \setminus glb(T)$.

Consider the subgraph monomorphism instance (G_p, G_t) in Figure 8. Nodes of the pattern graph are the variables of the corresponding CSP, i.e. node i of G_p corresponds to variable x_i. Suppose that x_1 has been assigned to value 1. Because of $\mathrm{MC}(x_1, x_3)$, $D(x_3) = \{4,6,7\}$. Moreover, because of $\mathrm{alldiff}(x_1, \cdots, x_n)$, value 1 is deleted from all domains $D(x_i)$ $(i \neq 1)$. The new CSP P' consistd of the subgraph of $G_p^- = (\{2,3,4,5\}, \{(2,3), (3,2), (3,5), (5,3), (4,5), (5,4), (2,4), (4,2)\})$ and $G_t^- = (\{2,3,4,5\}, \{(2,3), (3,2), (3,5), (5,3), (4,5), (5,4), (2,4), (4,2)\})$. The domains of the variables of P' are : $D(x_3) = \{4,6,7\} = \{4\}$, $D(x_2) = \{2,5,6,7\} = \{2,5\}$, $D(x_5) = \{2,5,6,7\} = \{2,5\}$, $D(x_3) = \{3,4,6,7\} = \{3,4\}$. The automorphisms of G_t^- are D_4. For the state S, $Sol(S) = \{(1,5,4,3,2), (1,2,4,3,5)\}$ and $BSol(S) = \{(1,2,4,3,5)\}$. For the subproblem P', $Sol(P') = \{(5,4,3,2), (2,4,3,5)\}$ and $BSol(P') = \{(2,4,3,5)\}$. The partial assignment $(x_1, 1)$ in state S together with the solutions of P' equals $Sol(S)$.

7.2 Breaking

Breaking local value symmetries is equivalent to breaking value symmetries on the subproblem P'. Puget's method and the dynamic GE-Tree method can thus be applied to the local CSP P'

8 Experimental results

The CSP model for subgraph monomorphism has been implemented in Gecode (http ://www.gecode.org), using CP(Graph) and CP(Map) [8] [7] . CP(Graph) provides graph domain variables and CP(Map) povides function domain variables. All the software was implemented in C++. The standard implementation of NAUTY algorithm was used. We also implemented SchreierSims algorithm. The computation of the constraints for breaking injective problems was implemented, and GE-Tree method was also incorporated.

We have evaluated variable symmetry detection and breaking, value symmetry detection and breaking, and variable and value symmetry breaking.

The data graphs used to generate instances are from the GraphBase database containing different topologies and has been used in [15]. There is a directed and an undirected set of graphs. We took the first 30 graphs and the first 50 graphs from GraphBase. The directed set contains graphs ranging from 10 nodes to 462 nodes. The undirected set contains graphs ranging from 10 nodes to 138 nodes. Using those graphs, there are 405 instances for directed graphs and 1225 instances for undirected graphs. All runs were performed on a dual Intel(R) Xeon(TM) CPU 2.66GHz with 2 Go of RAM.

A main concern is how much time it takes to preprocess the graphs. NAUTY processed each undirected graph in less than 0.02 second. For directed graphs, each graph was processed in less than 0.01 second except one of them which terminate in 0.8 second and 4 of them which did not terminate in five minutes. Note that we did not tune NAUTY. For the SchreierSims algorithm, each directed graph was processed in less than one second except for 3 of them which terminate in 0.5 second, 1 of them in 1.5 seconds, and 1 of them in 3.1 seconds. All undirected graphs were processed in less than one second, except two of them, with 4 seconds and 8 seconds.

In our tests, we look for all solutions. A run is solved if it finishes under 5 minutes, unsolved otherwise. We applied the basic CSP model, the model where breaking variable symmetries with generators (Gen.) are posted, and finally the full variable symmetry (FVS) that breaks all variable symmetries. Results are shown in Table 1 and 2. In those runs, the preprocessing time has not been considered. The total time column shows the total time needed for the solved instances. The mean time column shows the mean time for the solved instances.

Thanks to variable symmetry breaking constraints more instances are solved, either for the directed graphs or for the undirected graphs. Moreover, the time for

TAB. 1 – Comparison over GraphBase undirected graphs.

All solutions 5 min.				
	solved	unsol	total time	mean time
CSP	58%	42%	70 min.	5.95 sec.
Gen.	60,5%	39,5%	172 min.	13.95 sec.
FVS	61.8%	38.2%	101 min.	8 sec.

TAB. 2 – Comparison over GraphBase directed graphs.

All solutions 5 min.				
	solved	unsol	total time	mean time
CSP	67%	33%	21 min.	4.31 sec.
Gen.	74%	26%	47 min.	8.87 sec.
FVS	74%	26%	40 min.	7.64 sec.

solved instances was increased because of the variable symmetry breaking constraints. Regarding the mean time, the full variable symmetry breaking constraint has a clear advantage. This mean time increase is an astonishing behaviour that should be investigated.

Value symmetry breaking was evaluated on the set of directed graphs. Table 3 shows that only one percent was gained. This may be due to the fact that there are less symmetries in directed graph than in undirected graphs.

For variable and value symmetries, a total of 233 undirected random instances were treated. We evaluated variable and values symmetries separately and then together in Table 4. This table shows that, as expected, value symmetries and variable symmetries each increase the number of solved instances. Notice here that value symmetry breaking with GE-Tree leads to new solved instances and better performance, reducing mean time on solved instances. Full variable symmetry technique makes new instances solved, but does not significatively reduce mean time on solved instances. Moreover, the combination of value symmetry breaking and variable symmetry breaking do not combine the power of the two techniques. In fact the GE-Tree upper bound of the number of the solved solutions is not increased by using full variable symmetry technique, and its mean time is even increased.

From these experiments, we conclude that although variable and value symmetry gives better performances and make new instances solved, they are not sufficient to make a significative higher percentage of instances solved. This calls for conditional and local symmetry detection and breaking.

TAB. 3 – Comparison over GraphBase directed graphs for value symmetries.

All solutions 5 min.				
	solved	unsol	total time	mean time
GE-Tree	68%	32%	21 min.	4.39 sec.

Tab. 4 – Comparison over GraphBase undirected graphs for variable and value symmetries.

All solutions 5 min.				
	solved	unsol	total time	mean time
CSP	53,6%	46,3 %	31 min.	20.1 sec.
GE-Tree	55,3%	45,7 %	6 min.	3.21 sec.
FVS	54,9 %	45,1%	31 min.	19 sec.
GE-Tree and FVS	55,3 %	44,7%	26 min.	8.68 sec.

9 Conclusion

In this paper, we presented techniques for symmetry breaking in graph matching. Specific detection techniques were first developped for the classical variables symmetries and value symmetries. We show that variable symmetries and value symmetries can be detected by computing the set of automorphisms on the pattern graph and on the target graph. We also showed that conditional value symmetries can be detected by computing the set of automorphisms on various subgraphs of the target graph, called dynamic target graphs. The GE-Tree method has been extended to handle these conditional symmetries. We introduced the concept of local value symmetries, that is symmetries on a subproblem. It was shown how such new symmetries can be computed and exploited using standard methods such GE-Tree. Experimental results analysed the enhancement achieved by variables symmetries and value symmetries. It showed that symmetry breaking is an effective way to increase the number of tractable instances of the graph matching problem.

Future work includes more experiments on conditional symmetries and local value symmetries, and the development of heuristics for the integration of these symmetries on suitable search states. An interesting research direction is the automatic detection of symmetries in graph domain variable. Finally, an open issue is the ability to handle local variable symmetries.

Références

[1] N. Beldiceanu, P. Flener, and X. Lorca. The tree constraint. In *Proceedings of CP-AI-OR'05*, volume LNCS 3524. Springer-Verlag, 2005.

[2] H. Cambazard and E. Bourreau. Conception d'une constrainte globale de chemin. In *10e Journ. nat. sur la résolution de problèmes NP-complets (JNPC'04)*, pages 107–121, 2004.

[3] Ronay-Dougal C.M., I.P. Gent, Kelsey T., and Linton S. Tractable symmetry breaking in using restricted search trees. *ECAI'04*, 2004.

[4] David Cohen, Peter Jeavons, Christopher Jefferson, Karen E.Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. In van Beek [28], pages 17–31.

[5] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3) :265–298, 2004.

[6] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry breaking predicates for search problem. In *Proceedings of KR'96*, 1996.

[7] Yves Deville, Grégoire Dooms, Stéphane Zampelli, and Pierre Dupont. Cp(graph+map) for approximate graph matching. *1st International Workshop on Constraint Programming Beyond Finite Integer Domains, CP2005*, 2005.

[8] Grégoire Dooms, Yves Deville, and Pierre Dupont. Cp(graph) : Introducing a graph computation domain in constraint programming. *Principles and Pratice of Constraint Programming*, 2005.

[9] Ian .P. Gent, Tom Kelsey, Steve Linton, and Colva Roney-Dougal. Symmetry and consistency. In van Beek [28], pages 271–285.

[10] Ian .P. Gent, Tom Kelsey, Steve A. Linton, Iain McDonald, Ian Miguel, and Barbara M. Smith. Conditional symmetry breaking. In van Beek [28], pages 256–270.

[11] I.P. Gent. A symmetry breaking constraint for indistinguishable values. In *Proceedings of CP'01, SymCon'01 Workshop*, 2001.

[12] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints : symmetry breaking during search. In *Proceedings of CP'02*, pages 415–430, 2002.

[13] I.P. Gent, W. Harvey, and T. Kelsey. Generic sbdd using computational group theory. In *Proceedings of CP'03*, pages 333–346, 2003.

[14] I.P. Gent and B.M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings of CP'01*, pages 599–603, 2001.

[15] Javier Larrosa and Gabriel Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical. Structures in Comp. Sci.*, 12(4) :403–422, 2002.

[16] Nikos Mamoulis and Kostas Stergiou. Constraint satisfaction in semi-structured data graphs. In Mark Wallace, editor, *CP2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 393–407. Springer, 2004.

[17] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30 :45–87, 1981.

[18] P. Meseguer and C. Torras. Exploiting symmetries within the constraint satisfaction search. *Artificial intelligence*, 129(1-2) :133–163, 2001.

[19] Jean-Francois Puget. Symmetry breaking using stabilizers. In Francesca Rossi, editor, *Proceedings of CP'03*, volume 2833 of *Lecture Notes in Computer Science*, pages 585–599. Springer, 2003.

[20] Jean-Francois Puget. Breaking all values symmetries in surjection problems. In *Proceedings of CP'05*, pages 490–504, 2005.

[21] Jean-Francois Puget. Elimination des symétries dans les problèmes injectifs. In *Proceedings des Journées Francophones de la Programmation par Contraintes*, 2005.

[22] Jean-François Puget. Automatic detection of variable and value symmetries. In van Beek [28], pages 477–489.

[23] J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, volume 1, pages 362–367. Amer. Assoc. Artificial Intelligence, 1994.

[24] Michael Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *TAGT*, volume 1764 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 1998.

[25] M. Sellman. Cost-based filtering for shorter path constraints. In *Proc. of the 9th International Conference on Principles and Pratice of Constraint Programming (CP).*, volume LNCS 2833, pages 694–708. Springer-Verlag, 2003.

[26] B. Smith. Reducing symmetry in a combinatorial design problem. *Proc. CP-AI-OR'01, 3rd Int. Workshop on Integration of AI and OR Techniques in CP*, 2001.

[27] Sébastien Sorlin and Christine Solnon. A global constraint for graph isomorphism problems. In Jean-Charles Régin and Michel Rueher, editors, *CPAIOR*, volume 3011 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2004.

[28] Peter van Beek, editor. *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, Augustus 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*. Springer, 2005.

[29] Stéphane Zampelli, Yves Deville, and Pierre Dupont. Approximate constrained subgraph matching. *Principles and Pratice of Constraint Programming*, 2005.