Technical Report

IRI-TR-17-04

Detection of Permutation Symmetries in Numerical Constraint Satisfaction Problems

Ieva Dauzickaite

March 2017

Institut de Robòtica i Informàtica Industrial

Abstract

This technical report summarizes the work done by Ieva Dauzickaite during her traineeship at IRI from 2016-10-03 to 2017-03-31. The main objectives of the stay at IRI were to study detection of variable symmetries in numerical constraint satisfaction problems and to implement a method that detects such symmetries.

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
`http://www.iri.upc.edu`

Corresponding author:
V. Ruiz de Angulo
tel: +34 93 401 7337
`ruiz@iri.upc.edu`
`http://www.iri.upc.edu/staff/ruiz`

1 Introduction

As problems in robotics involve complicated mechanisms and real world interactions, a significant amount of constraints have to be considered when addressing them. Such constraints may be, for instance, kinematic (i.e., related to the pose of the robot, the mechanical chains defining it, the joint limits, or the contacts with the environment), dynamic (taking into account the forces acting on the robot and the motor effort limits), of collision-avoidance (guaranteeing the safety of the robot itself and of the objects and human operators in the environment), energetic (bounding the energy to be consumed by the robot), or of time (limiting the time to complete a given task). Thus, it comes at no surprise that many tasks in robotics can be modeled as Constraint Satisfaction Problems (CSPs), which are NP-complete problems [1]. Actually, CSPs appear in other domains, including operational research or artificial intelligence. In such fields, the variables in the CSPs are typically discrete modeling problems in transportation, timetable scheduling, map colouring, sudoku, n-queens, propositional satisfiability, or cryptography. Thus, due to its prevalence in many fields, discrete CSPs have been widely studied, but in robotics the variables are typically continuous and, thus, the focus is on numerical CSPs (NCSP) rather than on discrete ones.

In numerical CSPs, the constraints are typically given in the form of equations. Thus, to find a solution fulfilling all the constraints, a system of equations has to be solved. The equation are typically complex and non-linear, e.g. combinations of trigonometric functions, integrals, differential equations etc., and often the number of variables in the problem is considerable. Several methods exist to solve arbitrary systems of equations. Algebraic methods [5] reduce the system to a resultant (e.g., an equation in a single variable), which is solved using standard methods. Continuations approaches [16] trace the solutions from a system with trivial solutions to the desired system of equations. An alternative which has been shown to be competitive with algebraic and continuation approaches is to rely on branch-and-prune [9, 10, 11]. In this approach, the search space is represented as an hyper-box (i.e., a box in the Cartesian space defined by the variables in the problem) that is recursively reduced and split until the solution set is bounded with the desired accuracy (Fig. 1). For large problems, though, it takes a significant amount of time to explore the search space and to isolate the solution set.

One promising approach to alleviate this issue is to exploit symmetries [13, 14], since they are present in many CSPs. For instance, in the problem represented in Fig. 1 there is a clear symmetry about the line $x = y$. By common definition, symmetry is the property of remaining invariant under particular modifications. Various examples under different changes are often observed in distinct branches of science, e.g. reflection symmetry in organisms in biology, rotational symmetry in molecules in chemistry, stability of physical laws describing the motion under translation of objects. In mathematics, the best known are symmetries of graphs of different functions with respect to the origin or some line, e.g. coordinate axis. The given definition of symmetry itself suggests that it is important to identify what the symmetry acts on, i.e. what is changed, and what is left invariant. To speed up branch-and-prune methods it is possible to take advantage of symmetries that permute variables and constraints and keep solutions invariant, i.e. always maps solutions to solutions and, consequently, non-solutions to non-solutions. In the presence of such symmetries, it is only necessary to explore a non-symmetric portion of the search space. Thus, only a subset of all solutions is found. If needed, other solutions can be generated as images of already known solutions by symmetries. For example, if we consider the equation $x + y + z = 3$, where variables x, y and z have the same domain $[0, 3]$, it is possible to consider one of the solutions $(0, 1, 2)$ and generate its symmetrical solutions $(0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)$ only if needed. In the case of Fig. 1, only the solution points with $y \geq x$ need to be isolated since if (x, y) is a solution, (y, x) is also a solution.

However, in order to exploit the symmetries, they must be identified at first. Relying on

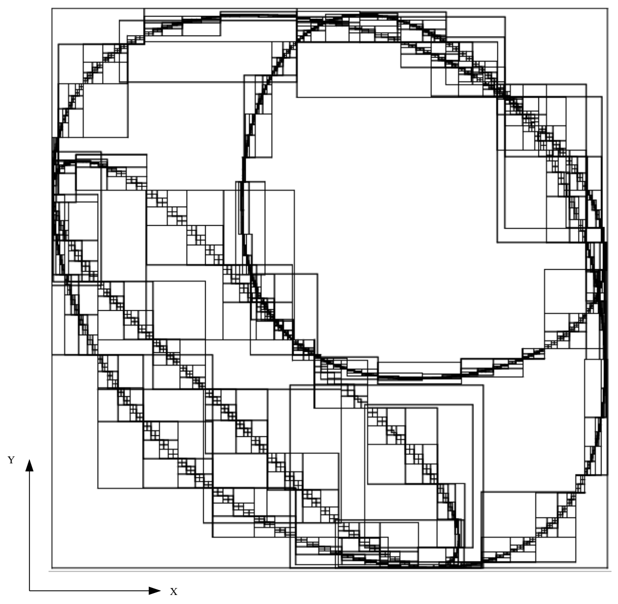


Figure 1: A branch-and-prune procedure progressively isolates the solution set of a system of equations representing the search space with an hyperbox that is reduced, split, and recursively processed until the solution set is isolated at the desired accuracy. In this case, the process converges to the one-dimensional space of valid configurations of a 7R robot.

human insight may lead to finding only a small part of symmetries, as not all of them are obvious or the problem might be too large to comprehend. There has been some attempts to identify symmetries automatically in CSPs with discrete domains. Surprisingly, the literature on symmetry detection on numerical CSPs is scarce. The few existing methods are based on symbolic manipulation and they are prone to missing symmetries. One of the reasons why both automatic symbolic methods and human insight may fail is the complicated expressions that may appear in the equations defining the numerical CSPs constraints. For example, symmetries might be not identified if one of the symmetric constraints is given as an integral of a function that is hard to integrate symbolically. Consequently, we want to design a method that detects all constraint symmetries due to permutations of variables and constraints in numerical CSPs. In this work we present a study of the detection of such symmetries and an implementation of methods that identifies them.

The rest of this paper goes as follows. In section 2 we define principal structures regarding CSPs and symmetries. Previous work on symmetry detection in NCSPs is discussed in section 3. Further, in section 4 we present three methods to identify variable constraint symmetries, and review results of our testing in section 5. Conclusions are proposed in section 6. Lastly, we supply appendix on Group Theory.

2 Problem Formalization

First of all, we give basic definitions.

Definition 1. CSP is a tuple $< X, D, C >$, where

- $X = \{x_1, x_2, \dots, x_n\}$ is a set of variables,
- $D = \{d_1, d_2, \dots, d_n\}$ is a set of domains where d_i specifies possible values for variable $x_i \in X$,

- $C = \{c_1, c_2, \dots, c_m\}$ is a set of constraints. Each constraint can be regarded as a predicate of function on a subset of X.

We call CSP discrete if all domains $d_i \in D$ are discrete. CSP is numerical (NCSP) if its variables are continuous, i.e. domains $d_i \in D$ are subsets of $\mathbb{R}$, usually compact intervals. For NCSPs we will denote D as a box $\mathbf{x}$ defined as the Cartesian product of the variable domains.

A set $\{\langle x_{i_1}, v_{i_1}\rangle, \langle x_{i_2}, v_{i_2}\rangle, \dots, \langle x_{i_r}, v_{i_r}\rangle\}$, where $r < n$, $\{x_{i_1}, x_{i_2}, \dots, x_{i_r}\} \subseteq X$, and $v_{i_j} \in d_{i_j} \in D$, is called a partial assignment. By convention, the variables not included in the partial assignment can take any value in its corresponding domain. If $r = n$, the assignment is complete. Thus, solution to CSP (both discrete and numerical) is a complete assignment such that all the constraints $c_i \in C$ are satisfied. We will denote the solution to NCSP as a point $x = (x_1, x_2, \dots, x_n) := (v_1, v_2, \dots, v_n) \in \mathbf{x}$.

Constraints might be expressed by allowed partial assignments, i.e. extensionally (for discrete CSP), or by a relation between variables, i.e. intensionally. For instance, consider a CSP with variables $X = \{x_1, x_2, x_3\}$ and domains $d_1 = \{1, 2\}$, $d_2 = \{2, 3\}$, $d_3 = \{1, 4\}$. The set $C = \{\{\langle x_1, 2\rangle, \langle x_2, 2\rangle\}, \{\langle x_2, 2\rangle, \langle x_3, 4\rangle\}, \{\langle x_2, 3\rangle, \langle x_3, 4\rangle\}\}$ is the extensional representation of some constraints, while the corresponding intensional representation is $C = \{x_1 = x_2, x_2 < x_3\}$.

A solution symmetry of CSP and NCSP is a bijection s which maps solutions to solutions, i.e. if x is a solution, then $s(x)$ is also a solution. The symmetries that also preserve constraints, i.e. maps C to itself, are identified as constraint symmetries. Moreover, part of solution symmetries can be classified either as variable or value symmetries if they act only on the problem variables or values, respectively. In the case of discrete CSP, s usually acts on variables and values preserving the set of literals, i.e. variable-value pairs denoted as $(x_i = v_i)$, where $x_i \in X$ and $v_i \in d_i$.

In this work we are interested in variable symmetries which are also constraint symmetries. We assume that every $c_i \in C$ is of the form $f_i(x) = 0$ and we have a constraint set $F = \{f_1(x), f_2(x), \dots, f_m(x)\}$. The set of variable constraint symmetries is defined as $G := \{p \in \Sigma_n | Fp = F\} \subseteq Sigma_n$, where $p = \left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ 1^p & 2^p & \cdots & n^p \end{smallmatrix}\right)$, $Fp = \{f_1(x^p), f_2(x^p), \dots, f_m(x^p)\}$, $x^p = (x_{1^p}, x_{2^p}, \dots, x_{n^p})$. Thus, we consider permutation $p \in \Sigma_n$ as a symmetry if and only if there exist a permutation $\sigma \in \Sigma_m$ such that $f_i(x^p) = f_{i^\sigma}(x)\ \forall i \in \mathbb{N}_m$, where $N_m = \{1, 2, \dots, m\}$. For simpler notation $f_i(x^p)$ is further denoted as $f_i p$.

We show that the set G under composition of permutations forms a subgroup of Σ_n.

Proof. Let $p_1, p_2 \in G$. Then $F(p_1 p_2) = (Fp_1)p_2 = Fp_2 = F$, so $p_1 p_2 \in G$.

Let $p \in G$. Then $Fp^{-1} = (Fp)p^{-1} = F(pp^{-1}) = F$, so $p^{-1} \in G$.

Thus, $G \leq \Sigma_n$. □

3 Related Work

There are some studies on the automatic symmetry detection in discrete CSPs. However, due to some significant differences and drawbacks most of the ideas cannot be directly applied to NCSPs.

The first point where transition from discrete CSPs to NCSPs fails is the definition of symmetry. Cohen et al. [3] gives a survey of different symmetry definitions in earlier literature and proposes instead using only two: solution symmetry and constraint symmetry. Solution symmetry is defined generaly as a bijection on the set of literals which preserves the set of solutions to the CSP. Microstructure and its complement are introduced to define constraint symmetry. In a binary case microstructure is a graph with set of literals as the set of vertices and the edges of this graph join every pair of vertices which is allowed by a constraint. In case some constraint includes more than two variables, microstructure is defined as a hypergraph with the same set of vertices and hyperedges connecting combinations of vertices allowed by constraints. Since it is often easier to work with disallowed assignments, authors introduce the complement of microstructure. It is defined as either graph or hypergraph with a set of vertices that is equal to the one of microstructure. The edges or hyperedges of the complement of microstructure connect the vertices which are either disallowed by some constraint or are different assignments for the same variable. Authors define constraint symmetry as an automorphism of a microstructure of the CSP or its complement. Recall that graph automorphism is a permutation of the set of its vertices which preserves its edges, i.e. property of remaining invariant under particular changes holds.

Puget [12] also uses literals in his work on symmetry detection. He defines a variable symmetry as a bijection s that acts on the literal and only permutes the variable, i.e. $(x_i = v)^s = (x_{i^s} = v)$. By analogy, value symmetry is a bijection s that acts on the literal and only permutes the value, i.e. $(x_i = v)^s = (x = v^s)$. The symmetry detection method presented in [12] is based on constructing coloured graphs, whose automorphisms correspond to CSP symmetries. In this work, a coloured graph is defined as a triple (V, E, c), where V is a set of vertices, E is the set of edges, and c is a mapping from V to integers that represent colours. Then its automorphism also preserves colour of each vertex. This way it is possible to classify vertices and restrict the graph symmetries. Puget suggests constructing the graph as follows. First, a vertex for every variable is created. These vertices are grouped by assigning them the same colour. Also, there is a vertex for every constraint. Constraints are classified to a few classes according to their type. Every class of constraints is assigned a unique colour. The rest of the graph construction differs if the constraints are represented extensionally or intensionally.

If the constraints are represented extensionally there is a vertex for every possible value $v_i \in d_i$ for every variable $x_i \in X$. Colouring these vertices differ if we want to detect variable symmetries (this way all value vertices have the same colour) or value symmetries (every class of values for different variable has a different colour). Also, assignment vertices are created for each allowed assignment by every constraint and have the same colour. Edges connect pairs of variables and its possible values, constraints and its allowed assignments, and every assignment and values in that assignment. This way we can detect variable, value and some variable-value symmetries. However, this approach and the one by Cohen [3] rely on literal and it is obvious that literals are of no use for NCSPs as it is impossible to create a literal for every possible value of the variable in continuous domain.

If the constraints are represented intensionally, though, the method by Puget can be applied to NCSPs. In this case, the graph has less vertices than in the extensional case. There are only the variable and constraints vertices as defined above and sometimes a dummy vertex may be required to break a symmetry that only occurs in graph but not in the constraints of CSP. Edges link pairs of variable and constraint vertices. Any automorphism of this graph corresponds to a variable symmetry of CSP. Thus, this method is based on syntactical representation of the problem. As Puget himself states, because of this reason it is prone to missing part of the symmetries and the proposed workaround only copes with a part of this drawback. This is due to the fact that the same constraint can be expressed in different ways and the syntactical equivalence has to be captured encoding the corresponding identities. Not only there can be a list of these rules that is too long to be encoded, but some of the rules may be not even known. For example, a symmetric function $f(x, y) = sin^2(x)sin^2(y)$ with a simple shift $sin(y) = cos(y - \frac{\pi}{2})$ or Pythagorean identity $sin^2(y) = 1 - cos^2(y)$ can be transformed to functions $f(x, y) = sin^2(x)cos^2(y - \frac{\pi}{2})$ and $f(x, y) = sin^2(x)(1 - cos^2(y))$ that do not look symmetric from the first sight. Taking into account the number of possible identities, it seems highly possible that symmetries would fail to be identified. Also, in Puget's approach, some variable symmetries due to associativity of the operator may be lost. In order to avoid such failure it is suggested to expand the expressions and use n-ary version of the operator. However, such approach is impractical in robotics as the expressions are typically too complex.

In conclusion, all the methods to detect variable constraint symmetries in NCSP in the literature use graphs in which the operands are connected directly to the corresponding operator node. The symmetries detected by these graph methods are the automorphisms of the graph, which are simply the transformations of the graph that permute the order of the operators. Thus, the composition of permutation of symmetric operands (and, possibly, associativity when preprocessing the expression) are the only symmetries detected by these approaches.

4 Symmetry Detection with Symmetric Relation Test

In the following sections, three methods to detect variable constraint symmetries for NCSPs not based on the graph of the structure of the symbolic expression of the constraints are presented. They essentially rely on the test of particular symmetric relations between the constraints. Thus, they are able to find a much larger number of symmetry types than graph based approaches discussed in Section 3. The required tests on the symmetric relations can be done symbolically or numerically. Here, we focus only on numerical tests.

The methods are presented with increasing sophistication and decreasing expected computational cost.

4.1 Global Relation Testing

We consider a numerical brute force approach in order to find the set of variable constraint symmetries, G. First, a random point x is generated and $f_i(x)$ is computed for every constraint $f_i \in F$. Then, we compute values of all the constraints at a point x^p for some $p \in \Sigma_n$. Further, it is checked if the list $(f_1(x), f_2(x), \ldots, f_m(x))$ is a permutation of the list $(f_1(x^p), f_2(x^p), \ldots, f_m(x^p))$, up to a numerical threshold. If so, permutation p passes the test. Otherwise, p is dismissed. Note that assuming two function evaluations are the same up to a given numerical threshold may result in false positives, but it works well in practice.

When implementing the method, the presence of numerical errors has to be considered. To say that the list $(f_1(x), f_2(x), \ldots, f_m(x))$ is a permutation of the list $(f_1(x^p), f_2(x^p), \ldots, f_m(x^p))$, vectors of computed values at the points x and x^p are sorted and it is required that absolute difference between corresponding values is smaller than computational error provided by the user. Introduction of such threshold may result in false positives or false negatives. For some functions false negatives might be dismissed with more sophisticated approaches [15, 6]. Moreover, in our implementation the domains for generating x are limited as large input may cause numerical errors which also result in false positives or false negatives.

We perform such check for every permutation $p \in \Sigma_n$. That is, $n!$ tests and for each test we sort a vector of m elements (cost $(m \cdot log(m))$) and we compare the vector (cost m). Thus, the overall computational cost of this method is $n!(m \cdot log(m) + m)$.

4.2 Dividing Problem into Subproblems

Let's consider a function space $Y := \{h | h : \mathbb{R}^n \to \mathbb{R}\}$. We define an equivalence relation on Y: $f \sim h \Leftrightarrow \exists p \in \Sigma_n : \ fp = h$. This relation divides Y into equivalence classes $CL(f) := \{h \in Y | \ \exists p \in \Sigma_n : \ fp = h\}$. Notice that the set of constraints F is a subset of Y. Consequently, the same relation divides F into equivalence subclasses $cl(f) := \{h \in F | \ \exists p \in \Sigma_n : \ fp = h\} \subseteq CL(f)$. With the previous notation $cl(f_i) = \{h \in F | \exists p \in \Sigma_n \wedge \sigma \in \Sigma_m : \ f_i p = f_{i^\sigma} = h\}$. As equivalence classes either are disjoint or coincide, if $g \in G$ then the permutation $\sigma \in \Sigma_m$, such that $f_i g = f_{i^\sigma} \ \forall i \in \mathbb{N}_m$, only permutes elements of the set F within their equivalence subclasses. Thus, we can divide the problem of finding constraint symmetries into subproblems and search for $p \in \Sigma_n$ such that $cl(f_i)p = cl(f_i)$ for every $cl(f_i)$. Every subproblem preserves structure of the original problem, but possibly has less constraints within its scope. In this way, the computational cost of finding the symmetries is reduced.

If there are k different equivalence subclasses, we can organize the constraints in a way that the first k constraints are not pairwise equivalent. This way we have different equivalence subclasses $cl(f_1), cl(f_2), \ldots, cl(f_k),\ k \le m$, and $cl(f_i) = \{f_{i_1}, f_{i_2}, \ldots, f_{i_{l_i}}\}$, $m = \sum\limits_{i=1}^{k} l_i$. Then for every subproblem we are searching for a set of permutations which act on every element of the equivalence subclass and map it to an element of the same subclass, i.e. set $G_i = \{p \in \Sigma_n | \exists \sigma \in \Sigma_{l_i} :\ f_{i_j}p = f_{i_{j^\sigma}}\ \forall j \in \mathbb{N}_{l_i}\}$ is a solution to the subproblem. G_i under composition of permutations is also a subgroup of Σ_n.

Constraint symmetries of the original problem lie in intersection of solutions to the subproblems, i.e. $G = \bigcap\limits_{i=1}^{k} G_i$.

Proof. $p \in G \ \Leftrightarrow\ Fp = F \ \Leftrightarrow\ \exists \sigma \in \Sigma_m :\ f_j p = f_{j^\sigma}\ \forall j \in \mathbb{N}_m \ \Leftrightarrow\ p \in G_i\ \forall i \in \mathbb{N}_k \ \Leftrightarrow\ p \in \bigcap\limits_{i=1}^{k} G_i$. □

Next, we describe methods to determine the classes and a method to exploit this information when determining the symmetries of a given problem.

4.2.1 Obtaining Class Information

When dividing constraints into equivalence subclasses we exploit a necessary, but not sufficient condition. Consider a point $c = (c_1, c_1, \ldots, c_1),\ c \in \mathbb{R}^n$. Notice that $c^p = c$ with every $p \in \Sigma_n$. If there exists $\sigma \in \Sigma_m$ such that $f_i(x^p) = f_{i^\sigma}(x)$, $p \in \Sigma_n$, then $f_i(c^p) = f_i(c) = f_{i^\sigma}(c)$. Based on this observation we can divide constraints into possibly smaller sets of functions which have the same value at point c, up to a numerical threshold. These sets are either equivalence subclasses or unions of equivalence subclasses.

The latter case is demonstrated in following example. Functions $f_1(x_1, x_2) = x_1^2 - x_2^2 + 1$ and $f_2(x_1, x_2) = x_1^3 - x_2^3 + 1$ belong to different subclasses. However, $f_1(c) = c_1^2 - c_1^2 + 1 = 1$ and $f_2(c) = c_1^3 - c_1^3 + 1 = 1$. Thus, f_1 and f_2 are assigned to the same equivalence subclass.

In the further implementation of symmetry detection methods it is important to be aware of possibility of having merged equivalence subclasses. Nevertheless, these unions may be separated by applying stronger filter afterwards.

4.2.2 Global Relation Testing with Class Information

We consider a brute force numerical method to find G when constrains are divided into equivalence subclasses $cl(f_i)$ or unions of these sets as well. Assume we have only equivalence subclasses and no unions of them. First, we search for constraint symmetries of the first set $cl(f_1)$, i.e. G_1, as described in section 4.1. Then we proceed with search for $G_1 \cap G_2$. This is achieved by performing the test in 4.1 with constraints in $cl(f_2)$ only for permutations $p \in G_1$. With this test permutations which are in G_1, but not in G_2 are dismissed. In the same manner sets $(G_1 \cap G_2) \cap G_3$, ..., $(G_1 \cap G_2 \cap \cdots \cap G_{k-1}) \cap G_k = \bigcap\limits_{i=1}^{k} G_i = G$ are found. The number of comparisons depends on the number of equivalence subclasses, and cardinalities of every $cl(f_i)$ and G_i.

In case we also have unions of equivalence subclasses $\bigcup\limits_j cl(f_j)$, the same symmetries are found. Notice, $cl(f_i) \cap cl(f_j) = \emptyset$ for any pair of different i, j. Consider $p \in \bigcap\limits_j G_j$. It is true if and only if $p \in G_j\ \forall j$. Thus, $cl(f_j)p = cl(f_j)\ \forall j$. Therefore, $(\bigcup\limits_j cl(f_j))p = \bigcup\limits_j (cl(f_j)p) = \bigcup\limits_j cl(f_j)$ and p passes the test. If $p \notin \bigcap\limits_j G_j$, then $p \notin G_j$ for some j and $cl(f_j)p \neq cl(f_j)$ for the same j. Hence, $(\bigcup\limits_j cl(f_j))p = \bigcup\limits_j (cl(f_j)p) \neq \bigcup\limits_j cl(f_j)$ and p is dismissed.

The cost of the method is $n!(l \cdot log(l) + l) + m \cdot log(m) + m$, where $l = \max\limits_{i \in \{1,2,\ldots,k\}} \{l_i\}$ and where $m \cdot log(m) + m$ is the cost of determining the class information. In the worst case if there is only one equivalence subclass, $l = m$ and the cost is $(n! + 1)(m \cdot log(m) + m)$. Since in many cases $l < m$, this method typically reduces the computational cost of finding the symmetries.

4.3 Local Relation Testing Approach

Methods described in 4.1 and 4.2.2 might be impractical for larger problems due to the $n!$ part of the computational cost. In order to overcome this obstacle we also present a numerical method based on testing relations between pairs of functions. These relations provide all the information needed to find variable constraint symmetries. We exploit the fact that constraints in different equivalence subclasses are not related. Moreover, it is not necessary to test relations for every pair of constraints in the subclass: it is enough to find relations between one representative and every other constraint in the subclass. With these relations constraints can be represented as sets of permutations. Hence, problem might be encoded in a sparse graph which automorphisms are variable constraint symmetries G. Notice, that this graph-based method bypasses the obstacles arising in the graph-based methods relying on the syntactical representation of the constraints discussed in section 3.

Structures of group theory are introduced in pursuit of the relations[1]. Assume the problem is divided into k subproblems. For each subproblem we find a stabilizer of f_i, $i \in \mathbb{N}_k$, i.e. $A_{f_i} := \{p \in \Sigma_n | f_i p = f_i\}$. Stabilizer stores all the symmetric relations between representative of a subclass f_i with itself. It is known that $A_{f_i} \leq \Sigma_n$. We also find a subset of right cosets of A_{f_i} in Σ_n: $A_{f_i}t_{i1},\ A_{f_i}t_{i2}, \ldots,\ A_{f_i}t_{il_i}$, where $t_{ij} \in \Sigma_n :\ f_i t_{ij} = f_{i_j},\ f_{i_j} \in cl(f_i)$. We choose $t_{i1} = id$, hence $A_{f_i} = A_{f_i}t_{i1}$. Every other coset $A_{f_i}t_{ij}$ contains permutations that map f_i to f_{i_j}, i.e. symmetric relations between functions f_i and f_{i_j}. We construct subset of right transversal of A_{f_i} in Σ_n: $RT(A_{f_i}) := \{t_{i1}, t_{i2}, \ldots, t_{il_i}\}$ and sets of cosets of A_{f_i}: $U_i := \{A_{f_i}t_{ij} | t_{ij} \in RT(A_{f_i}),\ f_i t_{ij} = f_{i_j}\}$. Every U_i represents relations between constraints in equivalence subclass $cl(f_i)$.

4.3.1 Generating Relations with Vector Representation

When searching for the stabilizers and their cosets, we introduce another necessary, but not sufficient condition. Due to it, significant number of permutations might be dismissed instantly when searching for relations $p \in \Sigma_n : fp = g$ and the whole Σ_n may not have to be checked. To test the permutations, we define a function which captures the constraint value when only one variable is changed and allows to find possible relations between distinct variables: $\mathcal{F}(f,j)(x) := f(x_1, x_2, \ldots, x_j + \delta, \ldots, x_n)$. Then $\mathcal{F}(f,j)(x^p) = f(x_{1^p}, x_{2^p}, \ldots, x_{j^p} + \delta, \ldots, x_{n^p})$, $p \in \Sigma_n$.

The relation test is performed for functions f and g which are assigned to the same equivalence subclass as described in Section 4.2.1. Thus, we say that there exists a permutation $p \in \Sigma_n$ such that $fp = g$. Therefore $\mathcal{F}(f,j)(x^p) = \mathcal{F}(g,j^p)(x)$ for every $j \in \mathbb{N}_n$. Again, consider the point $c = (c_1, c_1, \ldots, c_1)$, $c \in \mathbb{R}^n$. Then not only $f(c) = g(c)$ but also $\mathcal{F}(f,j)(c^p) = \mathcal{F}(f,j)(c) = \mathcal{F}(g,j^p)(c)\ \forall j \in \mathbb{N}_n$. To exploit this condition we define vectors whose elements represent values

[1]The basic elements of group theory are described in Appendix A.

of constraints when one variable is changed: $u := (\mathcal{F}(f,1)(c), \mathcal{F}(f,2)(c), \ldots, \mathcal{F}(f,n)(c))$ and $v := (\mathcal{F}(g,1)(c), \mathcal{F}(g,2)(c), \ldots, \mathcal{F}(g,n)(c))$. Thus, for every $p \in \Sigma_n$ such that $fp = g$ it is true that $u = v^p$ (u and v coincide when searching for auto-symmetries).

Assume there does not exist such $p \in \Sigma_n$ that the condition $u = v^p$ is met. Thus, there is also no $p \in \Sigma_n : \ fp = g$ and functions f and g are in different equivalence subclasses. This way we easily know that testing with c as described in 4.2.1 failed and some equivalence subclasses were merged.

To construct permutations p satisfying the condition $u = v^p$, we define sets $A_i := \{j | u_j = \alpha, \text{ where } \alpha \text{ is the i-th different value of } u \text{ in ascending order}\}$ and $C_i := \{j | v_j = \alpha, \text{ where } \alpha \text{ is the i-th different value of } v \text{ in ascending order}\}$. Assume that u is a permutation of v. Then the required condition $u = v^p$ is met with permutations $p \in \Sigma_n : \ A_i p = C_i \ \forall i$. We find one such permutation $t \in \Sigma_n$ that $A_i t = C_i \ \forall i$. We also define sets of auto-symmetries of the sets A_i: $P_i := \{p \in \Sigma_n | A_i p = A_i\}$. Thus, condition $u = v^p$ is satisfied with permutations $\mathcal{P} = \{p \in \Sigma_n | A_i p = C_i \ \forall i\} = (\bigcup_i P_i)t$.

As condition $u = v^p$ is necessary, but not sufficient, we have to filter permutations in $\mathcal{P}$ with a stronger filter. We adapt the relation test in 4.1 for two functions and only test the permutations in $\mathcal{P}$.

4.3.2 Representing Constraints with Permutations

It is important to notice that for every subproblem both sets U_i and $cl(f_i)$ have the same amount of elements and if all the constraints are different, so are the cosets.

Proof. Assume two cosets are the same: $A_{f_i} t_{ia} = A_{f_i} t_{ib}$. Then $t_{ia} t_{ib}^{-1} \in A_{f_i} \Rightarrow f_i = f_i(t_{ia} t_{ib}^{-1}) = (f_i t_{ia}) t_{ib}^{-1} = f_{i_a} t_{ib}^{-1} \Rightarrow f_i t_{ib} = f_{i_a} \Rightarrow f_{i_b} = f_{i_a}$. □

Thus, there exist a bijection between the equivalence subclass $cl(f_i)$ and set of cosets U_i: $f_{i_j} \leftrightarrow A_{f_i} t_{ij}$. That is, every constraint f_{i_j} is represented by a set of permutations. Recall that when solving the subproblem we are searching for permutations of variables which keep the subclass of constraints $cl(f_i)$ invariant, i.e. map the constraints from $cl(f_i)$ to constraints in the same subclass $cl(f_i)$. When constraints are represented with sets of permutations, the solution to the subproblem is a set of permutations which map sets representing constraints to the sets representing constraints in the same equivalence subclass: $G_i = \{t \in \Sigma_n | U_i t = U_i\}$.

Proof. Let $p \in \Sigma_n$. Then

$$\begin{aligned}
U_i p = U_i &\Leftrightarrow \exists \sigma \in \Sigma_{l_i} : \ (A_{f_i} t_{ij}) p = A_{f_i}(t_{ij} p) = A_{f_i} t_{ij^\sigma} \quad \forall j \in \mathbb{N}_{l_i} \\
&\Leftrightarrow t_{ij} p \in A_{f_i} t_{ij^\sigma} \quad \forall j \in \mathbb{N}_{l_i} \\
&\Leftrightarrow t_{ij} p t_{ij^\sigma}^{-1} \in A_{f_i} \quad \forall j \in \mathbb{N}_{l_i} \\
&\Leftrightarrow f_i(t_{ij} p t_{ij^\sigma}^{-1}) = f_i \quad \forall j \in \mathbb{N}_{l_i} \\
&\Leftrightarrow f_i(t_{ij} p) = f_i t_{ij^\sigma} \quad \forall j \in \mathbb{N}_{l_i} \\
&\Leftrightarrow (f_i t_{ij}) p = f_i t_{ij^\sigma} \quad \forall j \in \mathbb{N}_{l_i} \\
&\Leftrightarrow \exists \sigma \in \Sigma_{l_i} : \ f_{ij} p = f_{i_{j^\sigma}} \quad \forall j \in \mathbb{N}_{l_i} \\
&\Leftrightarrow p \in G_i.
\end{aligned}$$

□

Thus, we are searching for permutations which preserve the cosets in every U_i, i.e. $G = \bigcap_{i=1}^{k} G_i = \{p \in \Sigma_n | U_i p = U_i \ \forall i \in \mathbb{N}_k\}$.

Notice that in case some equivalence subclass $cl(f_i)$ has only one element, that is f_i itself, and its stabilizer A_{f_i} contains only id element, i.e. $U_i = \{A_{f_i}\} = \{\{id\}\}$, the solution to the original problem is $G = \{id\}$.

4.3.3 Coloured Graph Construction

We approach the search for G as follows. If $|A_{f_i}| = q_i$ and there are l_i elements in $cl(f_i)$, then all the constraints in F are represented by $q = \sum_{i=1}^{k} q_i l_i$ permutations. The multiset of all these permutations: $P = \{p_1^{11}, p_2^{11}, \dots, p_{q_1}^{11}, p_1^{12}, \dots, p_{q_1}^{1l_1}, p_1^{21}, \dots, p_{q_k-1}^{kl_k}, p_{q_k}^{kl_k}\}$, where $p_a^{ij} \in A_{f_i} t_{ij},\ a \in \mathbb{N}_{q_i}$.

We construct a $q \times n$ matrix D with each row representing a permutation from P:

$$D = \begin{pmatrix} p_1^{11} \\ p_2^{11} \\ \vdots \\ p_{q_k}^{kl_k} \end{pmatrix} = \begin{pmatrix} 1^{p_1^{11}} & 2^{p_1^{11}} & \dots & n^{p_1^{11}} \\ 1^{p_2^{11}} & 2^{p_2^{11}} & \dots & n^{p_2^{11}} \\ \vdots & \vdots & \ddots & \vdots \\ 1^{p_{q_k}^{kl_k}} & 2^{p_{q_k}^{kl_k}} & \dots & n^{p_{q_k}^{kl_k}} \end{pmatrix}$$

When it is irrelevant to which coset the permutation belongs, we will denote p_r^{ab} as p_d, where d is the number of the row representing p_r^{ab} in the matrix D.

Then we construct a coloured graph $\mathcal{G} = (W, E, c)$, where W is a set of nodes, E - set of edges and c is a function which assigns colour to every node $w \in W$. $\mathcal{G}$ is constructed in a way that automorphisms of $\mathcal{G}$ correspond to constraint symmetries of our problem. To this intent a node is created for:

- every possible value of the matrix D elements, i.e. every element of the set $\mathbb{N}_n$ (value nodes $z_i,\ i \in \mathbb{N}_n$),
- every element of the matrix D (matrix nodes $m_{ij},\ i \in \mathbb{N}_q,\ j \in \mathbb{N}_n$. When it is convenient we will make more explicit the subproblem, coset, number of permutation within the coset and component of the permutation that corresponds to m_{ij} by denoting it $p_r^{ab}(j)$),
- every coset $A_{f_i} t_{ij}$ of the problem, element of U_j for every $j \in \{1, 2, \dots, k\}$ (coset nodes $b_{ij},\ i \in \mathbb{N}_k, j \in \mathbb{N}_{l_k}$).

Is convenient to group the nodes with the same color. We denote as Z the set of value nodes, M the set of matrix nodes, and B_i the set of coset nodes corresponding to the cosets involved in a subproblem U_i, $\{b_{ij}\}_{j \in \mathbb{N}_{l_i}}$.

For further graph construction, we create edges for every pair of:

- matrix nodes m_{ij} and m_{ij+1}, $i \in \mathbb{N}_q,\ j \in \mathbb{N}_{n-1}$,
- matrix node m_{ij} and value node $z_{j^{p_i}}$,
- matrix node $m_{i1} = p_r^{ad}(1)$ and coset node b_{ad}, for any $r \in \mathbb{N}_{q_a}$.

Finally, a different and unique colour is assigned to each of the subsets Z, M and B_i's. That is:

- matrix nodes have the same unique colour,
- value nodes have the same unique colour,
- coset nodes in B_i have the same unique colour.

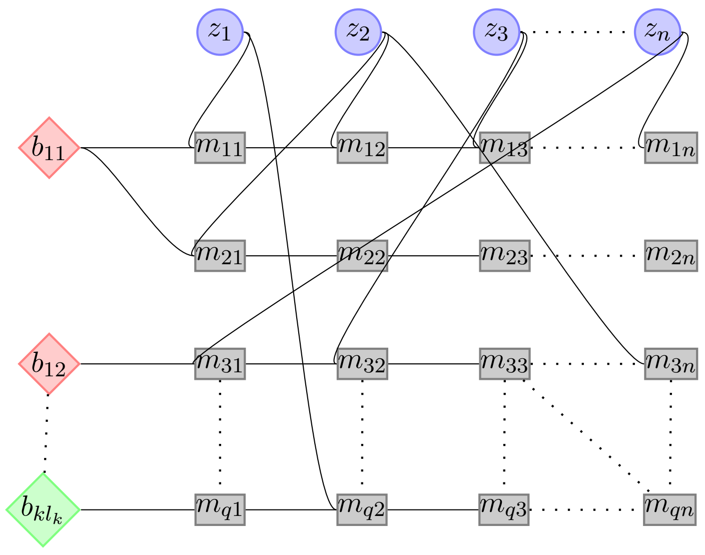


Figure 2: Coloured graph scheme.

We define a function $\phi : M \cup Z \to \mathbb{N}_n$, where $\phi(z_i) = i$ and, for $m_{ij} = p_a^{rl}(j)$, $\phi(m_{ij}) = \phi(p_a^{rl}(j)) = j^{p_i} = j^{p_a^{rl}}$. Note that matrix node m_{ij} is linked by en edge to value node z_r if and only if $\phi(m_{ij}) = \phi(z_r)$.

Moreover, the following proposition shows that there is a bijection between the automorphisms of $\mathcal{G}$ and the group of constraint symmetries of the problem and, therefore, G can be obtained by computing the automorphisms of $\mathcal{G}$.

Proposition 1. There exist a bijection from the group of automorphisms of the graph $\mathcal{G}$ to the group of symmetries of the problem given by:

$$Aut(\mathcal{G}) \longrightarrow G \tag{1}$$

$$\pi \longrightarrow t(\pi) = \begin{pmatrix} 1 & 2 & \cdots & n \\ \phi(\pi(z_1)) & \phi(\pi(z_2)) & \cdots & \phi(\pi(z_n)) \end{pmatrix} \tag{2}$$

Proof. $\underline{t(\pi) \text{ is a permutation of } \mathbb{N}_n}$

Let π be an arbitrary automorphism of $\mathcal{G}$. In the following, we abbreviate the function $t(\pi)$ for this fixed π as t. First, we check that t is indeed a permutation. All nodes in Z have the same colour, different of the colours of other nodes. Therefore, the restriction of the automorphism π to Z is a permutation of Z. Thus, the mapping $t(i) = t(\phi(z_i)) = \phi(\pi(z_i))$ is a permutation of $\mathbb{N}_n$.

$\underline{t \in G}$

First, note that for $p_l = p_r^{ij}$,

$$p_l = (\phi(m_{l1}), \phi(m_{l2}), \ldots, \phi(m_{ln})) \tag{3}$$

and

$$p_r^{ij} = (\phi(p_r^{ij}(1)), \phi(p_r^{ij}(2)), \ldots, \phi(\phi(p_r^{ij}(n))) \tag{4}$$

Then, we analize what $p_i t$ means in the terms of the graph automorphism:

$$p_i t = \begin{pmatrix} 1 & 2 & \cdots & n \\ \phi(m_{i1}) & \phi(m_{i2}) & \cdots & \phi(m_{in}) \end{pmatrix} \begin{pmatrix} 1 & 2 & \cdots & n \\ \phi(\pi(z_1)) & \phi(\pi(z_2)) & \cdots & \phi(\pi(z_n)) \end{pmatrix} =$$
$$= \begin{pmatrix} 1 & 2 & \cdots & n \\ \phi(\pi(z_{\phi(m_{i1})})) & \phi(\pi(z_{\phi(m_{i2})})) & \cdots & \phi(\pi(z_{\phi(m_{in})})) \end{pmatrix}. \tag{5}$$

We know that $(m_{ij}, z_r) \in E \Leftrightarrow \ \phi(m_{ij}) = \phi(z_r) = r \Leftrightarrow \phi(m_{ij}) = \phi(z_{\phi(m_{ij})})$. As π is an automorphism, if $(m_{ij}, z_r) \in E \Rightarrow (\pi(m_{ij}), \pi(z_r)) \in E$. Then $\phi(\pi(m_{ij})) = \phi(\pi(z_r))$ and also $\phi(\pi(m_{ij})) = \phi(\pi(z_{\phi(m_{ij})}))$.

Thus, we can conclude:

$$p_i t = \left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ \phi(\pi(m_{i1})) & \phi(\pi(m_{i2})) & \cdots & \phi(\pi(m_{in})) \end{smallmatrix} \right). \tag{6}$$

The same result using the extended notation for p_i and for the matrix nodes m_{ij} is

$$p_r^{ij} t = \left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ \phi(\pi(p_r^{ij}(1))) & \phi(\pi(p_r^{ij}(2))) & \cdots & \phi(\pi(p_r^{ij}(n)))) \end{smallmatrix} \right). \tag{7}$$

Now we show that application of π to M produces a permutation of the rows of M.

Remember that nodes M can only map to M nodes and, therefore, $\pi(M)$ is a permutation of M. Hence, to prove that $\pi(M)$ is a permutation of the rows of M we have only to show that π maps a row of M to a row of M. Take the first node of an arbitrary row i of M, matrix node $m_{i1} = p_z^{rl}(1)$ is linked to coset node b_{rl}. $\pi(b_{rl})$ must be a coset node belonging to B_r. Then assume $\pi(m_{i1}) = m_{ja}$ and $a \neq 1$. Thus, $(b_{rl}, m_{i1}) \in E$, but $(\pi(b_{rl}), \pi(m_{i1})) \notin E$ as a coset node can only be linked to the first element of a row of M. Then π is not an automorphism and there is a contradiction.

Thus, $\pi(m_{i1}) = m_{j1}$. Assume $\pi(m_{i2}) \neq m_{j2}$. Then $(m_{i1}, m_{i2}) \in E$, but $(\pi(m_{i1}), \pi(m_{i2})) \notin E$. Contradiction.

It is obvious that the same holds for all $m_{ir},\ r \in \{3, 4, \ldots, n\}$. Thus, $(\pi(m_{i1}), \pi(m_{i2}), \ldots, \pi(m_{in})) = (m_{j1}, m_{j2}, \ldots, m_{jn})$ and π permutes the rows of M. Or, in other words,

$$\exists\ \gamma \in \Sigma_q,\ (\pi(m_{i1}), \pi(m_{i2}), \ldots, \pi(m_{in})) = (m_{i^\gamma 1}, m_{i^\gamma 2}, \ldots, m_{i^\gamma n}).$$

To belong to G, t must permute the cosets of every U_i, i.e., $U_i t = U_i\ \forall i \in \mathbb{N}_k$ must hold or, what is the same, $\exists\ \sigma_i \in \Sigma_{l_i},\ A_{f_i} t_{ij}\ t = A_{f_i} t_{ij^{\sigma_i}}\ t\ \forall\ i \in \mathbb{N}_k,\ j \in \mathbb{N}_{l_i}$. Still in more detail, t must satisfy

$$\exists\ \sigma_i,\ \{p_a^{ij}\}_{a \in \mathbb{N}_{q_i}} t = \{p_a^{ij^{\sigma_i}}\}_{a \in \mathbb{N}_{q_i}}\ \forall\ i \in \mathbb{N}_k,\ j \in \mathbb{N}_{l_i} \Leftrightarrow \exists\ \sigma_i,\ \nu_{ij},\ p_a^{ij} t = p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}\ \forall\ a, i, j.$$

To this aim, first remember that because of color restrictions, π is a permutation of each B_i. This implies that for every i, a permutation $\sigma_i \in \Sigma_{l_i}$ exists such that $\pi(b_{ij}) = b_{ij^{\sigma_i}} \forall\ j$. Then, note that for the first node of an arbitrary row of M, $p_z^{ij}(1) = p_d^{ij^{\sigma_i}}(1)$ for some d because, if not, $(p_z^{ij}(1), b_{ij}) \in E$ and $(\pi(p_z^{ij}(1)), \pi(b_{ij})) \notin E$, contradicting π is an automorphism. Therefore, the set $\pi(\{p_a^{ij}(1)\}_{a \in \mathbb{N}_{q_i}})$ must be included in $\{p_a^{ij^{\sigma_i}}(1)\}_{a \in \mathbb{N}_{q_i}}$. But, since π is a bijection, a restriction of π to any subset must be also a bijection and its image under π must have the same cardinality. Thus π is a bijection of $\{p_a^{ij}(1)\}_{a \in \mathbb{N}_{q_i}}$ on $\{p_a^{ij^{\sigma_i}}(1)\}_{a \in \mathbb{N}_{q_i}}$, which means that for the given i, j there exists ν_{ij} such that $\pi(p_a^{ij}(1)) = p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(1)\ \forall\ a, i, j$. Moreover, since π maps rows of M to rows of M, what applies to the first element of each row applies also to the complete rows:

$$\exists\ \sigma_i,\ \nu_{ij},\ (\pi(p_a^{ij}(1)), \ldots, \pi(p_a^{ij}(n))) = (p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(1), \ldots, p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(n))\ \forall\ a, i, j. \tag{8}$$

Applying ϕ to all the elements of the tuple,

$$\exists\ \sigma_i,\ \nu_{ij},\ (\phi(\pi(p_a^{ij}(1))) \ldots, \phi(\pi(p_a^{ij}(n)))) = (\phi(p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(1)), \ldots, \phi(p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(n)))\ \forall\ a, i, j,$$

and using (7),

$$\exists\ \sigma_i,\ \nu_{ij},\ (\phi(\pi(p_a^{ij}(1))), \ldots, \phi(\pi(p_a^{ij}(n)))) = (\phi(p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(1)), \ldots, \phi(p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(n)))\ \forall\ a, i, j.$$

Finally, applying (7) to the left hand of the equality and (4) to right hand we get the desired result, $\exists\ \sigma_i,\ \nu_{ij},\ p_a^{ij} t = p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}\ \forall\ a, i, j$.

ϕ is a bijective function

It is clear that $t(\pi)$ is a unique permutation and, therefore, ϕ is a function. Let's prove that ϕ is injective, i.e., that $\pi_1 \neq \pi_2 \ \Rightarrow\ t(\pi_1) \neq t(\pi_2)$.

If $\pi_1 \neq \pi_2$ then either a value node or a matrix or a coset node is mapped differently by π_1 and π_2. If a value node is mapped differently, then obviously

$$\left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ \phi(\pi_1(z_1)) & \phi(\pi_1(z_2)) & \cdots & \phi(\pi_1(z_n)) \end{smallmatrix}\right) \neq \left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ \phi(\pi_2(z_1)) & \phi(\pi_2(z_2)) & \cdots & \phi(\pi_2(z_n)) \end{smallmatrix}\right),$$

and the proof is finished. Now assume that a matrix node $p_a^{ij}(r)$ is not mapped to the same node by π_1 and π_2. Remember that a row is always mapped to a row as a block by an automorphism (i.e., a node and its image are always in the same row). This means that the complete row $p_a^{ij}(1), \ldots, p_a^{ij}(n)$ is mapped to different rows by π_1 and π_2:

$$(\pi_1(p_a^{ij}(1)), \ldots, \pi_1(p_a^{ij}(n)) = (p_l^{iv}(1), \ldots, p_l^{iv}(n))$$

and

$$(\pi_2(p_a^{ij}(1)), \ldots, \pi_2(p_a^{ij}(n)) = (p_d^{ib}(1), \ldots, p_d^{ib}(n))$$

for some v, l, a, d, where $v \neq b$ or $l \neq d$.

If, contradicting the hypothesis, $t(\pi_1) = t(\pi_2)$ then on the one hand we have

$$(\phi(\pi_1(p_a^{ij}(1))), \ldots, \phi(\pi_1(p_a^{ij}(n))) = p_a^{ij}\ t(\pi_1) = p_a^{ij}\ t(\pi_2) = (\phi(\pi_2(p_a^{ij}(1))), \ldots, \phi(\pi_2(p_a^{ij}(n))).$$

We know that $p_l^{iv} \neq p_d^{ib}$ because in the case $v \neq b$ then p_l^{iv} and p_d^{ib} are two permutations pertaining to two different cosets of A_{f_i}, which have not intersection. If $v = b$, then $l \neq d$ for sure, which implies that p_l^{iv} and p_d^{ib} are two different permutations of the same coset. Thus, on the other hand we have

$$(\phi(\pi_1(p_a^{ij}(1))), \ldots, \phi(\pi_1(p_a^{ij}(n))) = (\phi(p_l^{iv}(1)), \ldots, \phi(p_l^{iv}(n))) = p_l^{iv} \neq$$
$$\neq p_d^{ib} = (\phi(p_d^{ib}(1)), \ldots, \phi(p_d^{ib}(n))) = (\phi(\pi_2(p_a^{ij}(1))), \ldots, \phi(\pi_2(p_a^{ij}(n))).$$

This is a contradiction which. Thus, if a matrix node is mapped differently by π_1 and π_2 then $t(\pi_1) \neq t(\pi_2)$.

Let's analyse now the case of π_1 and π_2 mapping a coset node b_{ij} differently. As each coset node is linked to a different set of M nodes, the set of M nodes linked to b_{ij} must be mapped by π_1 and π_2 to different sets of M nodes. Therefore, this case implies the previous one (a M node mapped differently by π_1 and π_2) for which, also in this case if $\pi_1 \neq \pi_2 \ \Rightarrow\ t(\pi_1) \neq t(\pi_2)$ and we can conclude that ϕ is injective.

Finally we show that ϕ is also surjective by proving

$$s \in G \Rightarrow \exists\ \pi \in Aut(\mathcal{G}),\ t(\pi) = s$$

The automorphism π satisfying the condition is: $\pi(z_i) = z_{i^s}$, $\pi(p_a^{ij}(r)) = p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(r)$ and $\pi(b_{ij}) = b_{ij^{\sigma_1}}$, where σ_i and ν_{ij} are the permutations satisfying (8). First, we check that $t(\pi) = s$. As $\phi(z_{i^s}) = i^s$, we have

$$t(\pi) = \left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ \phi(\pi(z_1)) & \phi(\pi(z_2)) & \cdots & \phi(\pi(z_n)) \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ 1^s & 2^s & \cdots & n^s \end{smallmatrix}\right) = s$$

Second, we prove that $\pi \in Aut(\mathcal{G})$. By the definition of π, it is evident that color constraints hold (a node of Z is mapped to a node of Z, a node of M to a node of M and a node of B_i is mapped to a node of B_i). It only remains to verify that edge constraints are also satisfied, case by case:

- Edges $(p_a^{ij}(r), (p_a^{ij}(r+1)), r \in \mathbb{N}_{n-1}$. We have $(\pi(p_a^{ij}(r)), \pi(p_a^{ij}(r+1))) = (p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(r), p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(r+1)) \in E$.

- Edges $(p_a^{ij}(r), z_{{}_r p_a^{ij}})\ \forall\ i, j, a, r$. Remember that, sin a M node is linked to a Z node if and only if they are mapped to the same value by ϕ, this set is also $(p_a^{ij}(r), z_{\phi(p_a^{ij}(r))})\ \forall\ i, j, a, r$. On the one hand we have

  $$p_a^{ij}\ s = p_{a^{\nu_{ij}}}^{ij^{\sigma_i}} = (\phi(p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(1), \ldots, \phi(p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(n))) = (\phi(\pi(p_a^{ij}(1))), \ldots, \phi(\pi(p_a^{ij}(n))).$$

  On the other hand, $p_a^{ij}\ s = p_a^{ij}\ t(\pi)$, which is the same expression as (5):

  $$p_a^{ij}\ s = (\phi(\pi(z_{\phi(p_a^{ij}(1))})), \ldots, \phi(\pi(z_{\phi(p_a^{ij}(n))}))).$$

  Thus, $\phi(\pi(p_a^{ij}(r))) = \phi(\pi(z_{\phi(p_a^{ij}(r))}))$, which implies that $(\pi(p_a^{ij}(r)), \pi(z_{\phi(p_a^{ij}(r)})) \in E$.

- Edges $(p_a^{ij}(1), b_{ij})$. We have $(\pi(p_a^{ij}(1)), \pi(b_{ij})) = (b_{ij^{\sigma_1}}, p_{a^{\nu_{ij}}}^{ij^{\sigma_i}}(r))$ which is also an edge because the subscripts of the coset node coincide with the superscripts of the matrix node.

□

5 Results

Methods Global Relation Testing (Global) presented in section 4.1, Global Relation Testing with Class Information (G classes) in 4.2.2 and Local Relation Testing (Local) in 4.3 were implemented with Matlab. In the Local approach Nauty [7] is used to search for automorphisms of $\mathcal{G}$. In case stopping condition (there is an equivalence class with only one element and only *id* as its stabilizer) is met, Local returns answer immediately and does not call Nauty. Note, Local returns generators of the symmetry group while Global and G classes find full groups.

Methods were tested with 40 problems chosen from [2, 4, 8]. We selected 20 problems with trivial symmetry group (non-symmetric problems) and 20 problems with symmetry group containing not only *id* element (symmetric problems). Number of variables ranges from 2 to 10 and there is one problem with 20 variables. In the latter case, the problem could only be tested with Local approach as generating full permutation group of order 20! for Global and G classes requires excessive amount of memory and time. When testing the relations, domains for random point x are restricted to intersection of the suggested domain and $[-10, 10]^n$. The chosen allowed computational error is 10^{-6}. Given set of parameters produces no false negatives or false positives for tested problems.

Tables with characteristics of the problems and time needed to find constraint symmetry groups with the methods are provided. The time for Local approach includes only time required for principal operations, i.e. searching for relations, and graph automorphisms (Nauty cpu time). In the tables n denotes number of variables in the problem, $|G|$ - size of the group of constraint symmetries, Subclasses - number of equivalence subclasses, and Relations is a sum of permutations representing constraints, i.e. number q defined in section 4.3.3. Note that $|G|$=1 for non-symmetric problems, thus it is excluded. Solving times are rounded to the fourth decimal place. There are separate tables for symmetric (Table 1) and non-symmetric (Table 2) problems. We also present scatter plots concerning solving time and number of variables (Fig. 3), and solving time and number of relations (Fig. 4). In the plots, Graph approach is represented with principal operations time as well, since other parts of the approach heavily on the implementation.

We also provide tables (Table 3 for symmetric and Table 4 for non-symmetric problems) with running time of different parts of Local implementation. Column Relations represents time needed to find stabilizers and cosets, Input - generating input file for Nauty, Automorphisms - Nauty cpu time, Output - parsing Nauty's output file, and all these times are summed in Total.

Notice that for symmetric problems with small number of variables ($n < 6$) and relations Local approach takes longer than Global or Global (classes). Although Global might show the best time when tackling these small problems, it is the worst method for all the others. Problems with larger number of variables and medium (from 120 to 75600) number of relations are fastest solved with Local. However, G classes shows the best time for Cyclic, which is problem with largest number of variables and relations. Due to the size of the graph Nauty itself takes more time to find its automorphisms than full execution of Global and G classes.

For non-symmetric cases with small number of variables ($n < 4$) Global and G classes perform better than Local approach as well. Again, Global is competitive only for small problems ($n < 4$). For all the rest problems, Local approach performs best of all the methods. Time differs significantly for the largest problems. Notice that due to stopping condition even problems with large number of variables or relations (More20 and Eco9 respectively) are solved in less than 0.1 seconds.

Problem	n	$\|G\|$	Subclasses	Relations	Time		
					Global	G classes	Local
Boon	6	4	3	120	0.141 3	0.081 2	0.013 5
Bronstein	3	2	3	10	0.001 1	0.001 8	0.009 2
Brown	8	5040	2	75600	7.002 1	6.557 1	2.943 0
Caprasse	4	2	2	4	0.003 3	0.003 4	0.015 8
Cbms2	3	3	1	3	0.001 1	0.001 5	0.010 8
Comp.soft[1]	2	2	1	2	0.000 5	0.001 0	0.010 2
Conform1	3	6	1	6	0.001 1	0.001 4	0.010 0
Cyclic	9	18	9	1088748	71.242 0	55.599 0	93.849 5
Cyclo	3	6	1	6	0.001 2	0.001 7	0.010 7
Extended Freudenstein	8	24	2	5760	6.982 2	4.297 3	0.177 1
Extended Powel	8	384	2	11520	6.972 6	4.346 9	0.516 5
Extended Wood	8	2	4	3360	6.968 1	2.986 6	0.069 7
Lorentz	4	4	1	4	0.003 3	0.003 8	0.012 7
Noon	5	120	1	120	0.015 8	0.016 3	0.017 6
Rabmo	9	4	9	1052	70.711 9	18.728 1	0.046 0
Reimer	5	12	5	60	0.016 4	0.010 7	0.015 6
Sparse	5	120	1	120	0.016 0	0.020 7	0.028 2
Virasoro	8	8	3	128	7.136 5	6.997 1	0.463 8
Vrahatis	9	9	1	45360	68.748 2	67.759 2	2.217 2
Wright	5	120	1	120	0.015 9	0.016 5	0.013 6

Table 1: Symmetric problems. 1 - full name: comp.soft-sys.math.maple-14706.

Problem	n	Subclasses	Relations	Time		
				Global	G classes	Local
Apollonius	8	8	21660	8.338 2	3.144 2	0.432 8
Branin System	3	3	4	0.001 2	0.001 9	0.023 9^a
Chemistry	5	5	8	0.016 0	0.008 2	0.006 3^a
Combustion Chemistry	4	3	6	0.003 3	0.003 4	0.006 9^a
Countercurrent Reactors	8	6	576	6.957 4	2.062 3	0.084 4
Eco9	8	8	40513	7.037 9	2.239 7	0.020 1^a
Eiger-Sikorski-Stenger	8	2	6480	6.984 1	2.368 6	0.123 2
Eqlin	3	3	4	0.002 0	0.005 7	0.070 5^a
Geneig	6	6	125	0.103 2	0.045 8	0.014 5^a
Himmelblau	2	2	2	0.000 6	0.000 8	0.002 7^a
I5	10	10	7200	795.204 9	187.960 0	5.586 7
Katsura6	7	7	1452	0.800 9	0.260 2	0.011 3^a
Kin1 Modified	6	6	30	0.106 6	0.040 6	0.019 2
Kincox	4	3	12	0.003 3	0.002 9	0.008 0
Kinema	9	8	3804	69.562 7	28.197 6	0.086 9
More20	20	20	20	$*$	$*$	0.080 1^a
Nauheim	8	8	1812	6.946 8	2.096 5	0.037 4
Robot Kinematics	8	5	6540	6.925 9	2.055 8	0.092 1
Stenger	2	2	2	0.000 6	0.000 8	0.001 9^a
Yamamura1	6	6	720	0.104 7	0.046 9	0.019 8

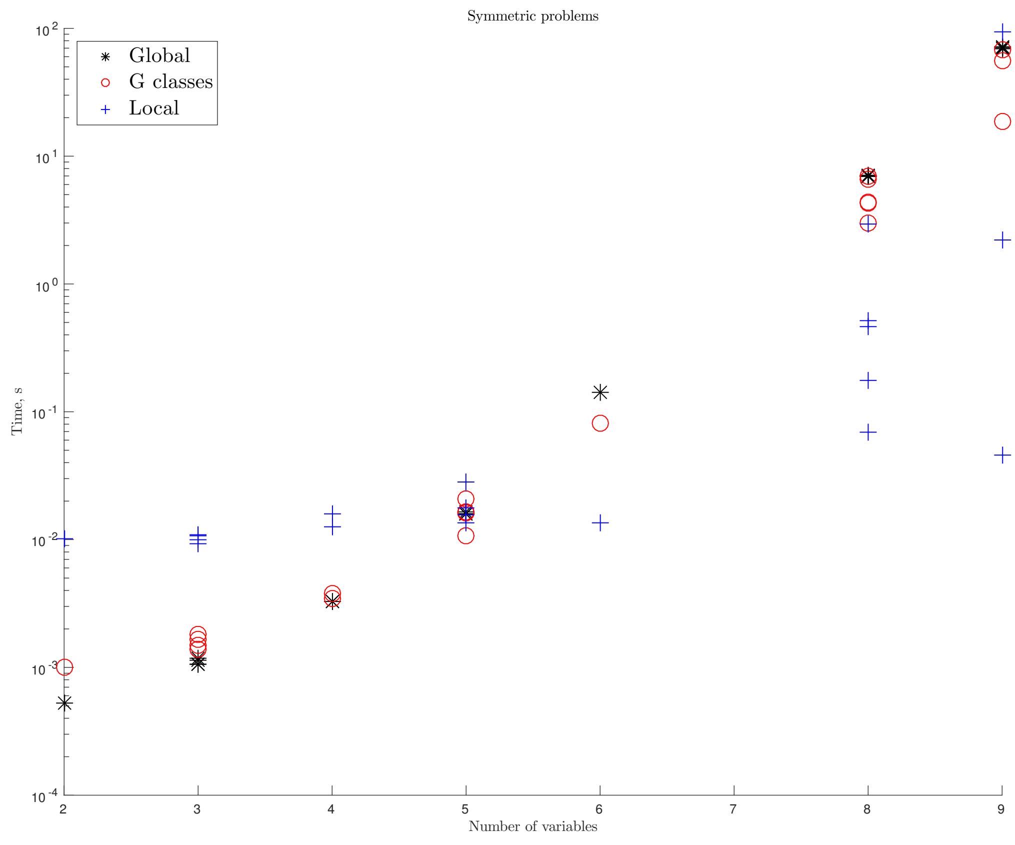
Table 2: Non-symmetric problems. a - stopping condition was met, $*$ - could not be solved.

Problem	Relations	Input	Automorphisms	Output	Total
Boon	0.009 0	0.020 7	0.004 5	0.507 0	0.541 2
Bronstein	0.004 6	0.001 9	0.004 7	0.004 6	0.015 7
Brown	0.379 2	15.422 8	2.563 7	1 394.696 9	1 413.062 7
Caprasse	0.011 0	0.002 0	0.004 8	0.003 7	0.021 6
Cbms2	0.006 2	0.001 6	0.004 7	0.004 8	0.017 3
Comp.soft[1]	0.005 4	0.001 2	0.004 8	0.002 0	0.013 4
Conform1	0.005 2	0.001 9	0.004 8	0.006 2	0.018 1
Cyclic	19.226 2	253.383 3	74.623 3	*	*
Cyclo	0.005 7	0.002 2	0.005 0	0.007 8	0.020 7
Extended Freudenstein	0.065 7	1.285 6	0.111 3	13.107 8	14.570 4
Extended Powel	0.102 7	2.638 3	0.413 8	92.415 1	95.569 8
Extended Wood	0.032 8	0.733 5	0.036 9	14.866 9	15.670 0
Lorentz	0.007 7	0.001 8	0.005 0	0.006 1	0.020 6
Noon	0.011 7	0.017 7	0.005 9	0.225 2	0.260 5
Rabmo	0.029 0	0.257 3	0.017 0	4.141 6	4.444 9
Reimer	0.010 5	0.009 4	0.005 0	0.087 0	0.112 0
Sparse	0.022 6	0.022 4	0.005 7	0.237 3	0.287 9
Virasoro	0.457 5	0.034 6	0.006 3	0.413 1	0.911 5
Vrahatis	0.236 6	11.398 7	1.980 6	129.923 0	143.538 9
Wright	0.007 8	0.021 7	0.005 8	0.413 8	0.449 1

Table 3: Time needed for different parts of Local implementation for symmetric problems. 1 - full name: comp.soft-sys.math.maple-14706, $*$ - file could not be parsed.

Problem	Relations	Input	Automorphisms	Output	Total
Apollonius	0.114 3	4.355 7	0.318 4	0.000 9	4.789 3
Branin System	0.023 9	0.000 8	0	0	0.024 7
Chemistry	0.006 3	0.000 4	0	0	0.006 7
Combustion Chemistry	0.006 9	0.000 3	0	0	0.007 1
Countercurrent Reactors	0.031 4	0.127 0	0.053 0	0.003 0	0.214 4
Eco9	0.020 1	0.000 4	0	0	0.020 5
Eiger-Sikorski-Stenger	0.051 8	1.310 2	0.071 4	0.000 8	1.434 2
Eqlin	0.070 5	0.013 3	0	0	0.083 8
Geneig	0.014 5	0.000 4	0	0	0.014 9
Himmelblau	0.002 7	0.000 2	0	0	0.002 9
I5	4.877 5	6.085 9	0.709 2	0.000 9	11.673 5
Katsura6	0.011 3	0.000 4	0	0	0.011 7
Kin1 Modified	0.015 5	0.005 7	0.003 7	0.001 3	0.026 3
Kincox	0.004 6	0.002 1	0.003 4	0.000 9	0.011 0
Kinema	0.033 1	0.857 3	0.053 8	0.001 0	0.945 2
More20	0.080 1	0.002 7	0	0	0.082 9
Nauheim	0.020 8	0.367 1	0.016 6	0.001 0	0.405 4
Robot Kinematics	0.043 0	1.327 2	0.049 2	0.001 1	1.420 4
Stenger	0.001 9	0.000 2	0	0	0.002 1
Yamamura1	0.012 8	0.111 0	0.007 0	0.000 9	0.131 7

Table 4: Time needed for different parts of Local implementation for non-symmetric problems.



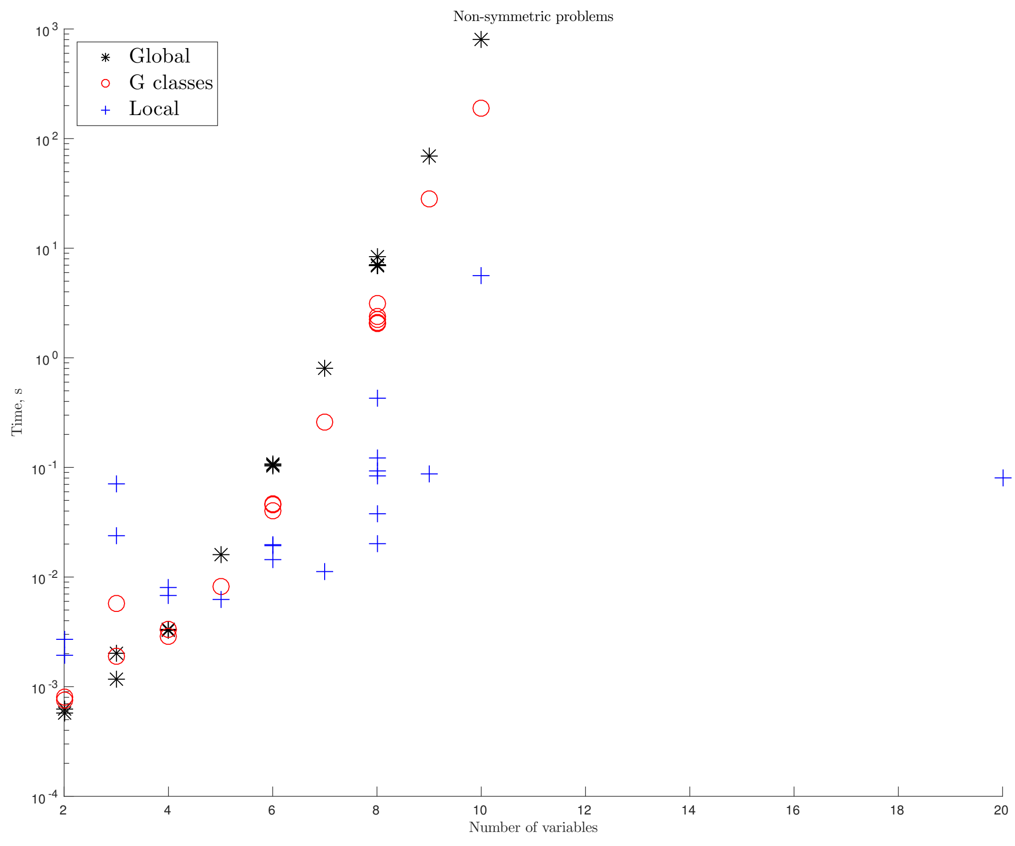
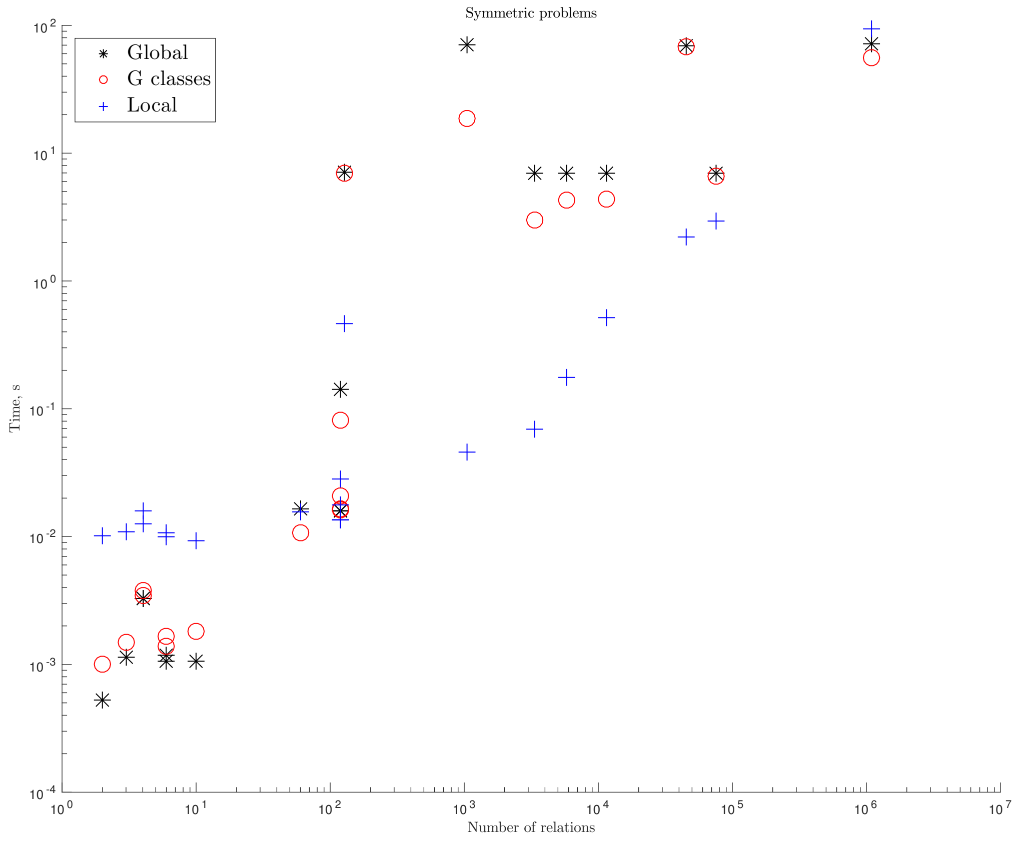


Figure 3: Results for symmetric and non-symmetric cases in respect to number of variables in the problem.



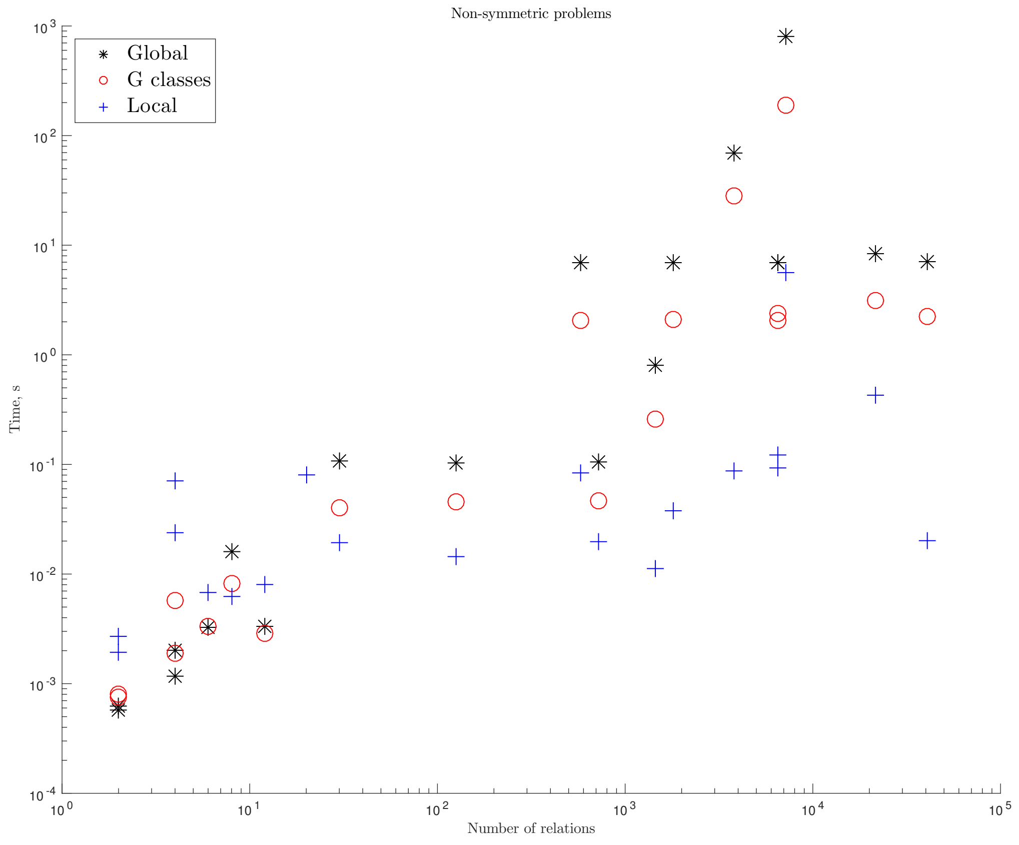


Figure 4: Results for symmetric and non-symmetric cases in respect to number of relations in the problem.

6 Conclusions

Presented algorithms can be used to detect variable constraint symmetries or conclude that the symmetry group is trivial in variety of problems. We have shown that for larger ones which possibly could benefit more when breaking the symmetries Local approach performs better than other methods given the number of relations is relatively low.

Total time of Local approach could be improved in the future if input for Nauty is generated faster with the same representation of constraints. Moreover, since the number of auto-symmetries for large problems can grow large, it might be also useful to search for a less extensive representation.

Appendix A: Group Theory

Definition 1. A set G together with an operation $* : G \times G \to G$ is called a group if it satisfies the following conditions:

1. (Associativity) $\forall a, b, c \in G : (a * b) * c = a * (b * c)$;
2. (Existence of identity element) $\exists e \in G : a * e = e * a = a \ \forall a \in G$;
3. (Existence of inverse element) $\forall a \in G \ \exists a^{-1} \in G \ : a * a^{-1} = a^{-1} * a = e$.

We abbreviate the notation of the group $(G, *)$ to G.

Definition 2. A non-empty set $H \subseteq G$ together with operation $* : G \times G \to G$ is called a subgroup of a group G if $\forall h_1, h_2 \in H : \ h_1 * h_2^{-1} \in H$.
We use notation $H \leq G$ to denote that H is a subgroup of G.

Definition 3. A set $X \subseteq G$ is called a generating set of a group $(G, *)$ if every element of G can be expressed as a finite product of elements from X and their inverses under operation $*$, i.e. $\forall g \in G : \ g = x_1^{\alpha_1} * x_2^{\alpha_2} * \cdots * x_n^{\alpha_n}$, where $x_i \in X, \ \alpha_i \in \{-1, 1\}, \ n < \infty$.
We will denote $G = \langle X \rangle$. Elements $x_i \in X$ are called generators of the group G.

Definition 4. A set $Hg = \{h * g \mid h \in H\}$, where $g \in G$, is called a right coset of a subgroup H in a group G.

Definition 5. A set T is called a right transversal of a subgroup H, $H \leq G$, if T contains exactly one element from each right coset of H in G.

Definition 6. A group is called a symmetric group on a set Ω if it is a group of all permutations of Ω under composition of mappings. We will denote this group as $Sym(\Omega)$. In case $\Omega = \{1, 2, \ldots, n\}$, we will denote the $Sym(\Omega)$ as Σ_n.

Definition 7. Action of a group G on a set Ω is a homomorphism $\phi : G \to Sym(\Omega)$.

Definition 8. A set $G_\beta = \{g \in G \mid \beta^g = \beta\}$, where G is a group acting on Ω, $\beta \in \Omega$, is called stabilizer of element β in group G.

References

[1] K. Apt, “Principles of constraint programming,” Cambridge University Press, 2003.

[2] COCONUT European Project, The COCONUT benchmark, `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`, 2003.

[3] D. A. Cohen, P. Jeavons, C. Jefferson, K. E. Petrie, B. M. Smith, “Symmetry Definitions for Constraint Satisfaction Problems,” in *van Beek P. (eds) Principles and Practice of Constraint Programming - CP 2005*, CP2005, Lecture Notes in Computer Science, vol 3709, pp. 17-31, Springer, 2005.

[4] COPRIN, The COPRIN examples page, `http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/`, 2011.

[5] D. Cox, J. Little, and D. O’Shea, “An Introduction to Computational Algebraic Geometry and Commutative Algebra,” 2nd ed. Springer, 1997.

[6] G. H. Gonnet, “Determining equivalence of expressions in random polynomial time,” in *STOC ’84 Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984.

[7] B. D. McKay and A. Piperno, “Practical Graph Isomorphism, II,” Journal of Symbolic Computation, vol. 60, pp. 94-112, 2014.

[8] PHCPack, The database of polynomial systems, `http://homepages.math.uic.edu/~jan/demo.html`, 2003.

[9] J. M. Porta, F. Thomas, L. Ros, C, Torras, “A branch-and-prune algorithm for solving systems of distance constraints,” IEEE International Conference on Robotics and Automation, pp.342-348, 2003.

[10] J.M. Porta, L. Ros, F. Thomas, C. Torras, “A branch-and-prune solver for distance constraints,” IEEE Transactions on Robotics, vol. 21, no. 2, pp. 176-187, 2005.

[11] J.M. Porta, L. Ros, O. Bohigas, M. Manubens, C. Rosales, L. Jaillet, “The CUIK Suite: Analyzing the Motion Closed-Chain Multibody Systems,” IEEE Robotics & Automation Magazine vo. 21, no. 3, pp. 105-114, 2014.

[12] J.-F. Puget, “Automatic Detection of Variable and Value Symmetries,” in *van Beek P. (eds) Principles and Practice of Constraint Programming - CP 2005*, CP2005, Lecture Notes in Computer Science, vol. 3709, pp. 475-489, 2005.

[13] V. Ruiz de Angulo and C. Torras. “Exploiting single-cycle symmetries in continuous constraint problems,”. Journal of Artificial Intelligence Research, vol. 34, pp. 499-520, 2009.

[14] G. Alexandre, J. Cristophe, V. Ruiz de Angulo and C. Torras: “Variable symmetry breaking in numerical constraint problems.” Artificial Intelligence, vol 229, pp. 105-125, 2015.

[15] J. T. Schwartz, “Fast Probabilistic Algorithms for Verification of Polynomial Identities,” Journal of the ACM (JACM), vol. 27, no. 4, pp. 701-717, 1980.

[16] A. J. Sommese and C. W. Wampler, “The Numerical Solution of Systems of Polynomials Arising in Engineering and Science,” World Scientific, 2005.

Acknowledgements

This work was supervised by Vicente Ruiz de Angulo and Josep M. Porta.