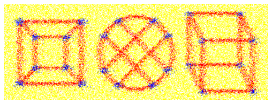

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 21, no. 2, pp. 195–218 (2017)
DOI: 10.7155/jgaa.00412

Symmetry Breaking Constraints for the Minimum Deficiency Problem

Sivan Altinakar[1] *Gilles Caporossi*[2] *Alain Hertz*[1]

[1]Department of Mathematics and Industrial Engineering
École Polytechnique - Gerad, Montréal, Canada
[2]Department of Decision Sciences
HEC - Gerad, Montréal, Canada

Abstract

An edge-coloring of a graph $G = (V, E)$ is a function c that assigns an integer $c(e)$ (called color) in $\{0, 1, 2, \dots\}$ to every edge $e \in E$ so that adjacent edges receive different colors. An edge-coloring is compact if the colors of the edges incident to every vertex form a set of consecutive integers. The minimum deficiency problem is to determine the minimum number of pendant edges that must be added to a graph such that the resulting graph admits a compact edge-coloring.

Because of symmetries, an instance of the minimum deficiency problem can have many equivalent optimal solutions. We present a way to generate a set of symmetry breaking constraints, called GAMBLLE constraints, that can be added to a constraint programming model. The GAMBLLE constraints are inspired by the Lex-Leader ones, based on automorphisms of graphs, and act on families of permutable variables. We analyze their impact on the reduction of the number of optimal solutions as well as on the speed-up of the constraint programming model.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
August 2016	December 2016	January 2017	January 2017	January 2017

Published: January 2017

Article type: Regular paper

Communicated by: G. Liotta

E-mail addresses: sivan.altinakar@gerad.ca (Sivan Altinakar) gilles.caporossi@gerad.ca (Gilles Caporossi) alain.hertz@gerad.ca (Alain Hertz)

1 Introduction

An *edge-coloring* of a graph $G = (V, E)$ is a function $c : E \to \{0, 1, 2, \dots\}$ that assigns a color $c(e)$ to every edge $e \in E$ such that $c(e) \neq c(e')$ whenever e and e' share a common endvertex. Let E_v denote the set of edges incident to vertex $v \in V$. An edge-coloring of $G = (V, E)$ is *compact* if $\{c(e) : e \in E_v\}$ is a set of consecutive non-negative integers for all vertices $v \in V$.

The problem of determining a compact k-edge-coloring (if any) of a graph was introduced by Asratian and Kamalian [3]. Determining whether or not a given graph admits a compact edge-coloring is known to be an $\mathcal{NP}$-complete problem [23], even for bipartite graphs. Given an edge-coloring c of a graph G and a vertex v, the *deficiency of* c *at* v, denoted $d_v(G, c)$, is the minimum number of integers that must be added to $\{c(e) : e \in E_v\}$ to form a set of consecutive integers. The *deficiency* of c is then defined as the sum $d(G, c) = \sum_{v \in V} d_v(G, c)$. Hence, c is compact if and only if $d(G, c) = 0$. The *deficiency of a graph* G, denoted $d(G)$, is the minimum deficiency $d(G, c)$ over all edge-colorings c of G. This concept, which was introduced by Giaro et al. [10], provides a measure of how close G is to be compactly colorable. Indeed, $d(G)$ is the minimum number of pendant edges that must be added to G such that the resulting graph is compactly colorable. The *Minimum Deficiency Problem* is to determine $d(G)$. It is an $\mathcal{NP}$-hard problem studied in [1, 2, 4, 8, 9, 10, 11, 12, 13, 20, 22].

As observed in [1], the problem of determining the deficiency of a small graph is surprisingly hard. The main difficulty is not to generate an optimal solution, but rather to prove its optimality. This is mainly due to the existence of many equivalent optimal solutions. The objective of this paper is to introduce symmetry-breaking constraints, in order to eliminate as many redundant solutions as possible.

Multiple authors have each identified and defined in different ways various types of symmetries in their respective research contexts. This paper adopts the terminology of [5, 7], which consists of a very general classification into *solution* and *problem* (or *constraint*) symmetries. Such permutations of the set of $(variable, value)$ pairs respectively preserve the solutions or the constraints of the problem. Moreover, the group of constraint symmetries is a subgroup of the group of solution symmetries. It is also worth noting that identifying solution symmetries usually requires finding all the solutions first, whereas constraint symmetries can be derived from the structure and expression of the problem. Finally, both of these types of symmetries allow two special cases, *variable* and *value* symmetries, which only permute variables or values, respectively.

The *Lex-Leader Method* proposed by Crawford, Ginsberg, Luks and Roy [6], and later improved in [5, 14, 19], will be at the basis of the research presented here. It adds constraints so as to allow only one member of each equivalence class. Such a method can produce a huge number of constraints and sometimes adding them to a model can be counterproductive. We propose to generate only a subset of these constraints, called GAMBLLE constraints, and analyze their impact on the reduction of the number of optimal solutions.

In [1], the authors have compared the performances of four models for the

minimum deficiency problem. It clearly appears that constrained programming models are significantly better than integer programming ones. The constrained programming model defined in [1] is described in Section 2. Graph automorphisms play an important role in creating symmetries for the minimum deficiency problem. This is illustrated in Section 3, and two methods to identify some or all of these automorphisms are proposed in Section 4. We then define GAMBLLE constraints in Section 5. These are added to the constraint programming model. Computational experiments are reported in Section 6 where we compare eight different ways of adding symmetry breaking constraints for the minimum deficiency problem.

2 Model

The symmetries encountered when solving a problem are clearly dependant on the model used to solve it. Following the previous experimentations in [1], a constraint programming model will be used to solve the minimum deficiency problem. It appears to be much faster than integer programming models and provides a simple correspondence to the graph formulation.

Consider a graph $G = (V, E)$, with vertex set V and edge set E. As mentioned in the introduction, E_v is the set of edges incident to vertex v. We denote by $\deg_v = |E_v|$ the degree of vertex v, and by $\Delta = \max_{v \in V} \deg_v$ the maximum degree of G. Also, C represents a set of colors $\{0, 1, \ldots, K-1\}$ where $K \geq \Delta$, $c_e \in C$ is the color assigned to edge e, and $\underline{c}_v$ and $\overline{c}_v$ respectively denote the minimal and maximal color assigned to an edge incident to vertex v. For the domain of the last two kind of variables, rather than using the entire set C, it is possible to shrink it a little, simply by taking into account the degree of their associated vertex. Finally, d_v is the deficiency at vertex v and $\sum_{v \in V} d_v$ is the deficiency of the coloring. We will use the following constrained programming model proposed in [1]:

$$\min \quad \sum_{v \in V} d_v \tag{1}$$

$$\text{s.t.} \quad \underset{e \in E_v}{\text{allDifferent}} \; c_e \qquad \forall v \in V \tag{2}$$

$$\underline{c}_v = \min_{e \in E_v} c_e \qquad \forall v \in V \tag{3}$$

$$\overline{c}_v = \max_{e \in E_v} c_e \qquad \forall v \in V \tag{4}$$

$$d_v = \overline{c}_v - \underline{c}_v + 1 - \deg_v \qquad \forall v \in V \tag{5}$$

$$|\{e \in E \mid c_e = 0\}| \geq 1 \tag{6}$$

$$c_e \in C \qquad \forall e \in E$$

$$\underline{c}_v \in \{0, \ldots, K - \deg_v\} \qquad \forall v \in V$$

$$\overline{c}_v \in \{\deg_v - 1, \ldots, K-1\} \qquad \forall v \in V$$

$$d_v \in \{0, \ldots, K - \deg_v\} \qquad \forall v \in V$$

Constraints (2) to (5) are the usual constraints for the minimum deficiency problem and are sufficient to model it accurately. Constraint (6) is added to help breaking the *value* symmetries due to shifting all the colors up or down, by simply enforcing that color 0 should be used at least once. That single extra constraint already immensely improves the performance of the model, and does not interfere with the future ones dealing with *variable* symmetries based on graph automorphisms that is the subject of the rest of this paper.

It was proved in [1] that if G is a graph with n vertices, then all edge-colorings with minimum deficiency $d(G)$ use at most $2n-4+d(G)$ colors. A fixed number of colors $K = 3n - 4$ can therefore be used for all practical purposes, based on the conjecture that the minimum deficiency $d(G)$ of G is always at most equal to n. If the conjecture is proven wrong and the model is applied to a graph with minimum deficiency greater than n, then the optimal value $D = \sum_{v \in V} d_v$ produced by the model will be larger than n, and we can then run the model a second time, using $K = 2n-4+D$ instead of $3n-4$ to determine the minimum deficiency of the considered graph.

3 Graph automorphisms

An *automorphism* of a graph $G = (V, E)$ is a permutation σ of its vertex set V such that $(u, v) \in E \Leftrightarrow (\sigma(u), \sigma(v)) \in E$. This reordering of the vertices of G thus preserves the adjacency matrix. The set of all automorphisms of a graph G forms the permutation group $Aut(G)$. This group acts naturally on V, by its definition. More interestingly, it can act on the edge set E by using the *edge action* h_E that maps a permutation $\sigma \in Aut(G)$ of vertices to a permutation $\sigma_E = h_E(\sigma)$ of the edges, where $\sigma_E((u, v)) = (\sigma(u), \sigma(v))$.

In what follows, we use the standard cycle notations for permutations [21]. It expresses a permutation as a product of cycles corresponding to the orbits of the permutation; since distinct orbits are disjoint, this is referred to as a decomposition into disjoint cycles. Cycles of length 1 are commonly omitted from the cycle notation. The identity permutation, which consists only of 1-cycles will be denoted by Id. Two solutions to the minimum deficiency problem that can be obtained one from the other by a permutation in $Aut(G)$ are called *equivalent solutions.*

Consider for example the chain P_4 on four vertices in Figure 1(a). Permutation $(v_1, v_4)(v_2, v_3) \in Aut(P_4)$ translates into permutation (e_1, e_3) of the edge set, and these two permutations both indicate that the two bottom optimal edge-colorings s_3 and s_4 are equivalent. As another example, consider the clique K_3 on three vertices in Figure 1(b). The two permutations (v_1, v_2) and (v_2, v_3) are generators for the permutation group $Aut(K_3)$, and these two permutations translate into permutations (e_2, e_3) and (e_1, e_2) of the edge set. In this case, the permutations in $Aut(K_3)$ indicate that all optimal edge-colorings are equivalent.

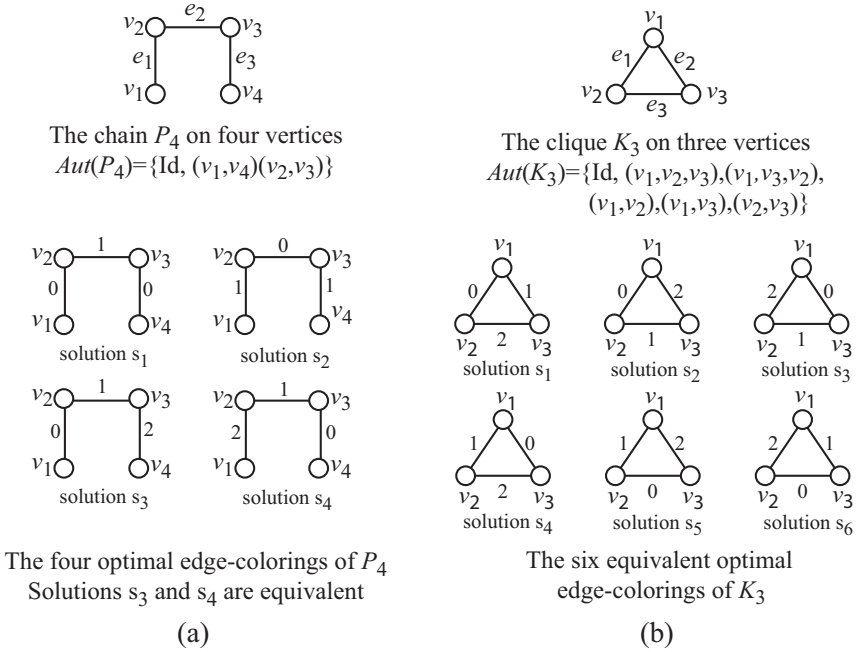


Figure 1: Automorphisms and optimal edge-colorings for P_4 and K_3.

$Aut(G)$ also acts on families of vertex (resp. edge) variables which have a one-to-one mapping with the set of vertices (resp. edges) of the graph, such as $\underline{c}_v$, $\overline{c}_v$ and d_v (resp. c_e). For example, consider again the chain on four vertices in Figure 1(a). Permutation $(v_1, v_4)(v_2, v_3)$ in $Aut(P_4)$ defines the four following variable permutations $(\underline{c}_{v_1}, \underline{c}_{v_4})(\underline{c}_{v_2}, \underline{c}_{v_3})$, $(\overline{c}_{v_1}, \overline{c}_{v_4})(\overline{c}_{v_2}, \overline{c}_{v_3})$, $(d_{v_1}, d_{v_4})(d_{v_2}, d_{v_3})$, and (c_{e_1}, c_{e_3}).

Constructing $Aut(G)$ is at least as difficult (in terms of computational complexity) as solving the graph isomorphism problem. Just counting the automorphisms is polynomial-time equivalent to graph isomorphism [15]. It is therefore unknown whether there is a polynomial time algorithm for constructing $Aut(G)$.

4 Methods for identifying automorphisms

4.1 nauty

Given an input graph G, the NAUTY library created by Brendan McKay[16, 17] outputs a description of $Aut(G)$ in terms of a generating set. The permutations that are part of this set tend to be fairly simple (when possible, a combination of disjoint transpositions). However, the set is not necessarily minimal for generating $Aut(G)$. For example in Figure 2, NAUTY outputs generators $(v_2, v_3), (v_3, v_4), (v_5, v_6), (v_7, v_8)$ and $(v_5, v_7)(v_6, v_8)$; (v_7, v_8) is redundant since it can be obtained by applying $(v_5, v_7)(v_6, v_8)$ followed by (v_5, v_6) and then $(v_5, v_7)(v_6, v_8)$ again.

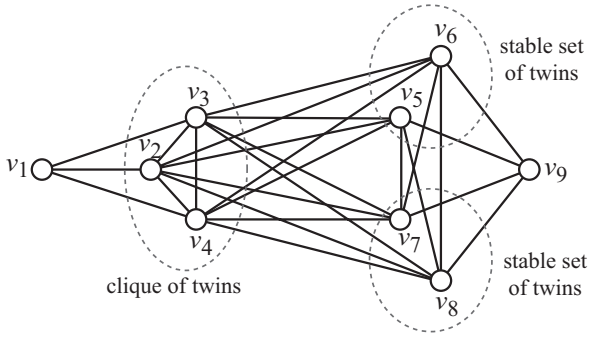


Figure 2: A graph with clique and stable sets of twins.

4.2 clusters

We define two vertices u and v as *twins* if all vertices $w \neq u, v$ are either adjacent to both u and v or to none of them. A *stable set of twins* is a set of at least two pairwise non-adjacent twins, and a *clique of twins* is a set of at least two pairwise adjacent twins. For illustration, vertices v_2, v_3, v_4 in Figure 2 form a clique of twins, while both sets $\{v_5, v_6\}$ and $\{v_7, v_8\}$ are stable sets of twins. The following theorem shows that there is a partition of the vertex set of a graph so that every block of the partition is a maximal stable set of twins, a clique of twins, or a singleton.

Theorem 1 *The intersection of a stable set of twins and a clique of twins is empty.*

Proof: Suppose, by contradiction, that there is a stable set S of twins and a clique K of twins such that $I = S \cap K \neq \emptyset$. Clearly, I contains at most one vertex since vertices in S are non-adjacent, while those in K are adjacent. So let $\{w\} = I$ and let $u \neq w$ be a second vertex in S. All vertices $v \neq w$ in K must be adjacent to u since u and w have the same set of adacent vertices. But no vertex $v \neq w$ in K can be adjacent to u since v and w have the same neighborhood. Hence K contains only one vertex, a contradiction. □

Finding a partition of the vertex set into maximal stable sets of twins, maximal cliques of twins and singletons is an easy task. Indeed, it is sufficient to observe that every maximal stable set of twins corresponds to a maximal set of identical lines of the adjacency matrix, and every maximal clique of twins corresponds to a maximal set of identical lines of the matrix obtained from the adjacency matrix by changing to 1 every element of its diagonal. All elements that are not in a stable set of twins or in a clique of twins are singletons of the partition. For example, the partition of the vertex set for the graph in Figure 2 is $\{\{v_1\}, \{v_2, v_3, v_4\}, \{v_5, v_6\}, \{v_7, v_8\}, \{v_9\}\}$.

Note that a permutation of a subset of vertices in a stable set of twins or in a clique of twins corresponds to an automorphism. More precisely, let $T = \{v_{i_1}, v_{i_2}, \ldots, v_{i_p}\}$ be a stable set of twins or a clique of twins. The $p - 1$

permutations (v_{i_1}, v_{i_2}), $(v_{i_2}, v_{i_3}),\ldots,$ $(v_{i_{p-1}}, v_{i_p})$ define a set of *generators* for all possible permutations of a subset of vertices in T. We call CLUSTERS the procedure that produces these generators. For the example of Figure 2, CLUSTERS would produce the set $\{(v_2, v_3), (v_3, v_4), (v_5, v_6), (v_7, v_8)\}$ of generators. Observe that permutation $(v_5, v_7)(v_6, v_8)$ (i.e. swapping the two stable sets of twins) found by NAUTY cannot be obtained by these generators.

The union of the generators produced by CLUSTERS defines the *cluster-automorphism* group $Aut_{\mathcal{C}}(G)$, which is a subgroup of $Aut(G)$. Hence, if $\mathcal{C}$ is the set containing all stables sets of twins and all cliques of twins, we have $|Aut_{\mathcal{C}}(G)| = \prod_{T\in\mathcal{C}}(|T|\,!) \leq |Aut(G)|$, and the total number of generators for $Aut_{\mathcal{C}}(G)$ is $\sum_{T\in\mathcal{C}}(|T|-1) \leq n - |\mathcal{C}|$.

It is known that for a graph G on n vertices, $Aut(G)$ can be specified by no more than $n-1$ generators. However, as mentioned in [18], NAUTY possibly requires an exponentional time to provide such a set of generators. For comparison, we have observed above that the partition of the vertex set into maximal stable sets of twins, maximal cliques of twins and singletons can be obtained in polynomial time, which means that CLUSTERS produces generators for $Aut_{\mathcal{C}}(G)$ in polynomial time.

5 gamblle constraints

The *Lex-Leader constraints* [6] deal with variable symmetries. They use a vector representation of the variables of a solution, and constrain all permutations of this vector under a set of symmetries to be lexicographically smaller to the first one. This amounts to reducing the solution space to one element for each equivalence class defined by symmetries. Such a method can produce a huge number of constraints, while a subset of these constraints can already be useful, for example when limited to the symmetries due to graph automorphisms.

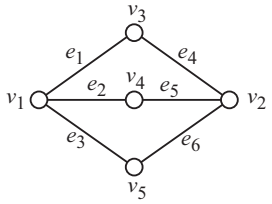


Figure 3: The $K_{2,3}$ bipartite graph, with $|Aut_{\mathcal{C}}(K_{2,3})| = |Aut(K_{2,3})| = 12$.

Consider for example the graph $K_{2,3}$ of Figure 3 and the following ordering of the variables of the model: $(\underline{c}_{v_1},\ldots,\underline{c}_{v_5},\overline{c}_{v_1},\ldots,\overline{c}_{v_5},d_{v_1},\ldots,d_{v_5},c_{e_1},\ldots,c_{e_6})$. Both CLUSTERS and NAUTY produce the set $\{(v_1, v_2), (v_3, v_4), (v_4, v_5)\}$ of generators. Permutation (v_3, v_4) imposes the following constraint :

$$(\underline{c}_{v_1},\underline{c}_{v_2},\underline{c}_{v_3},\underline{c}_{v_4},\underline{c}_{v_5},\overline{c}_{v_1},\overline{c}_{v_2},\overline{c}_{v_3},\overline{c}_{v_4},\overline{c}_{v_5},d_{v_1},d_{v_2},d_{v_3},d_{v_4},d_{v_5},c_{e_1},c_{e_2},c_{e_3},c_{e_4},c_{e_5},c_{e_6})$$
$$\preceq_{\text{lex}}(\underline{c}_{v_1},\underline{c}_{v_2},\underline{c}_{v_4},\underline{c}_{v_3},\underline{c}_{v_5},\overline{c}_{v_1},\overline{c}_{v_2},\overline{c}_{v_4},\overline{c}_{v_3},\overline{c}_{v_5},d_{v_1},d_{v_2},d_{v_4},d_{v_3},d_{v_5},c_{e_2},c_{e_1},c_{e_3},c_{e_5},c_{e_4},c_{e_6}).$$

When the head of the solution vector comprises all the elements of a family of permutable variables, part of the effect of a permutation of the vertices or of the edges is a rearrangment of this head. In the above example, the effect of permutation (v_3, v_4) is to change $(\underline{c}_{v_1}, \underline{c}_{v_2}, \underline{c}_{v_3}, \underline{c}_{v_4}, \underline{c}_{v_5}, \ldots)$ into $(\underline{c}_{v_1}, \underline{c}_{v_2}, \underline{c}_{v_4}, \underline{c}_{v_3}, \underline{c}_{v_5}, \ldots)$. The first simplification (called *trimming*) is to only explicit the first non-trivial comparison of each lexicographical constraint. In the previous example, the constraint becomes

$$\underline{c}_{v_3} \leq \underline{c}_{v_4}.$$

If we rearrange the variables so that $(c_{e_1}, c_{e_2}, c_{e_3}, c_{e_4}, c_{e_5}, c_{e_6}, \ldots)$ is the head of a solution vector, then it is changed to $(c_{e_2}, c_{e_1}, c_{e_3}, c_{e_5}, c_{e_4} c_{e_6}, \ldots)$ by permutation (v_3, v_4), and the constraint resulting from the trimming is

$$c_{e_1} \leq c_{e_2}.$$

In some cases, it is possible to use original constraints in the model to further strengthen a constraint to a strict inequality, as with the edge color variables when the considered edges have an endvertex in common. This results from the fact that such variables appear together in an allDifferent constraint. This special case makes the trimmed constraint equivalent to the full original lexicographical one. In the considered example, the constrained programming model imposes that c_{e_1}, c_{e_2} and c_{e_3} must be all different, and permutation (v_3, v_4) therefore gives

$$c_{e_1} < c_{e_2}.$$

The second simplification consists in adding constraints for only a few permutations from the automorphism group. Consider a permutation in $Aut(G)$ and let π be its effect on a family of permutable variables. Permutation π can be written as a product of disjoint cycles. Let C be the cycle in π that contains the variable with smallest index, say u_i. For every u_j in C with $j \neq i$ we do the following: if the model does not imply $u_i \neq u_j$, we add inequality $u_i \leq u_j$ to the set of constraints; otherwise, we add the strict inequality $u_i < u_j$. Every permutation π thus produces $|C| - 1$ constraints, to account for the whole orbit of the variable u_i created by repeated application of π. We name the resulting inequalities Graph AutoMorphism-Based Lex-Leader Enforcing (GAMBLLE) constraints. This is summarized in Algorithm 1. Line 3 guarantees that the inequality compares the first differing pair of the underlying lexicographical constraint based on the solution vector headed by the family $\mathcal{F}$ of permutable variables.

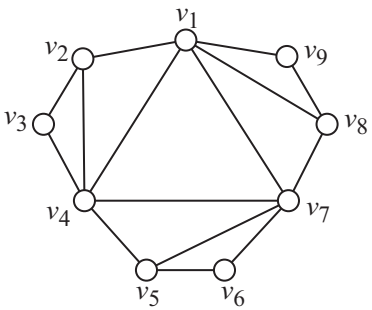
For illustration, consider the graph G in Figure 4 with $|Aut(G)| = 3$. It is the smallest graph with all permutations $\pi \neq \mathrm{Id}$ in $Aut(G)$ having no transposition (i.e., cycle with only two elements). NAUTY generates permutation $(v_1, v_4, v_7)(v_2, v_5, v_8)(v_3, v_6, v_9)$ with three cycles. The corresponding permutation of the $\underline{c}_v$ variables is $\pi = (\underline{c}_{v_1}, \underline{c}_{v_4}, \underline{c}_{v_7})(\underline{c}_{v_2}, \underline{c}_{v_5}, \underline{c}_{v_8})(\underline{c}_{v_3}, \underline{c}_{v_6}, \underline{c}_{v_9})$. The cycle with the variable of smallest index is $C = (\underline{c}_{v_1}, \underline{c}_{v_4}, \underline{c}_{v_7})$ and we therefore add constraints

$$\underline{c}_{v_1} \leq \underline{c}_{v_4} \text{ and } \underline{c}_{v_1} \leq \underline{c}_{v_7}.$$

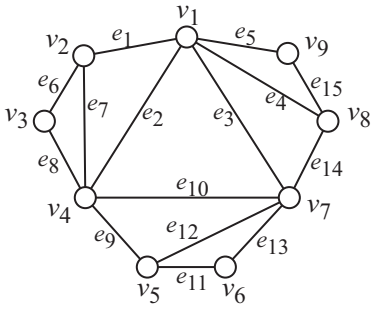
Algorithm 1: Generation of GAMBLLE constraints for a family of permutable variables

input : Graph G, a subset $P \subseteq Aut(G)$, a family $\mathcal{F}$ of ordered permutable variables
output: A set L of GAMBLLE inequality constraints

```
Let P^F be the set of permutations of the elements of F obtained from P
foreach π ≠ Id in P^F do
    Let C be the cycle of π with the smallest indexed variable u_i
    foreach u_j ∈ C with j ≠ i do
        if the model implies u_i ≠ u_j then
            Add inequality u_i < u_j to L
        else
            Add the inequality u_i ≤ u_j to L
```

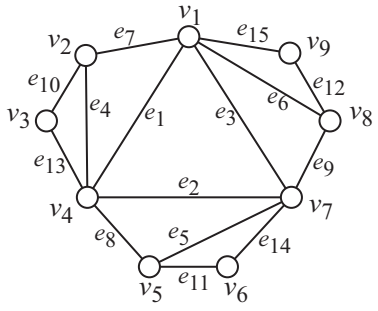


$Aut(G)$={Id, $(v_1,v_4,v_7)(v_2,v_5,v_8)(v_3,v_6,v_9)$,$(v_1,v_7,v_4)(v_2,v_8,v_5)(v_3,v_9,v_6)$}



$(v_1,v_4,v_7)(v_2,v_5,v_8)(v_3,v_6,v_9)$
translates into
$(e_1,e_9,e_{14})(e_2,e_{10},v_3)(e_4,e_7,e_{12})$
$(e_5,e_8,e_{13})(e_6,e_{11},e_{15})$

GAMBLLE constraints for the c_e variables
$c_{e_1} \le c_{e_9}$ and $c_{e_1} \le c_{e_{14}}$



$(v_1,v_4,v_7)(v_2,v_5,v_8)(v_3,v_6,v_9)$
translates into
$(e_1,e_2,e_3)(e_4,e_5,v_6)(e_7,e_8,e_9)$
$(e_{10},e_{11},e_{12})(e_{13},e_{14},e_{15})$

GAMBLLE constraints for the c_e variables
$c_{e_1} < c_{e_2}$ and $c_{e_1} < c_{e_3}$

Figure 4: Illustration of the generation of GAMBLLE *constraints.*

If we are interested in the edge colors, we first have to order the edges. One possible way is to order them using the lexicographical ordering of their pair of endvertices. Hence e_1 is the edge linking v_1 with v_2, e_2 is the edge linking v_1 with v_4, and so on. Such an ordering is shown on the leftside of Figure 4. Permutation $(v_1, v_4, v_7)(v_2, v_5, v_8)(v_3, v_6, v_9)$ translates into the following permutation of the c_e variables:

$$(c_{e_1}, c_{e_9}, c_{e_{14}})(c_{e_2}, c_{e_{10}}, c_{e_3})(c_{e_4}, c_{e_7}, c_{e_{12}})(c_{e_5}, c_{e_8}, c_{e_{13}})(c_{e_6}, c_{e_{11}}, c_{e_{15}}).$$

The cycle with smallest index is $C = (c_{e_1}, c_{e_9}, c_{e_{14}})$ and we therefore add constraints

$$c_{e_1} \leq c_{e_9} \text{ and } c_{e_1} \leq c_{e_{14}}.$$

Note that the ordering of the variables has an impact on the generated constraints. Indeed, for the same graph, we give on the rightside of Figure 4 a different labelling, where $(v_1, v_4, v_7)(v_2, v_5, v_8)(v_3, v_6, v_9)$ translates into

$$(c_{e_1}, c_{e_2}, c_{e_3})(c_{e_4}, c_{e_5}, c_{e_6})(c_{e_7}, c_{e_8}, c_{e_9})(c_{e_{10}}, c_{e_{11}}, c_{e_{12}})(c_{e_{13}}, c_{e_{14}}, c_{e_{15}}).$$

The cycle with smallest index is then $C = (c_{e_1}, c_{e_2}, c_{e_3})$ and we therefore obtain the following strenghtened constraints:

$$c_{e_1} < c_{e_2} \text{ and } c_{e_1} < c_{e_3}.$$

As a second example, consider the complete binary tree of Figure 5. NAUTY produces permutations $(v_2, v_3)(v_4, v_6)(v_5, v_7), (v_4, v_5)$ and (v_6, v_7) of the vertex set which correspond to permutations $(e_1, e_2)(e_3, e_5)(e_4, e_6), (e_3, e_4)$ and (e_4, e_6) of the edges when they are ordered as shown on the leftside of Figure 5, according to the lexicographical ordering of their pair of endvertices. The GAMBLLE constraints associated with the c_e variables are $c_{e_1} < c_{e_2}$, $c_{e_3} < c_{e_4}$ and $c_{e_5} < c_{e_6}$. The reverse ordering of the edges gives different GAMBLLE constraints since one of them is not a strict inequality. Indeed, the three permutations of the vertex set translate into permutations $(e_1, e_3)(e_2, e_4)(e_5, e_6), (e_3, e_4)$ and (e_1, e_2) of the edge set, and the associated GAMBLLE constraints are $c_{e_1} \leq c_{e_3}$, $c_{e_3} < c_{e_4}$ and $c_{e_1} < c_{e_2}$.

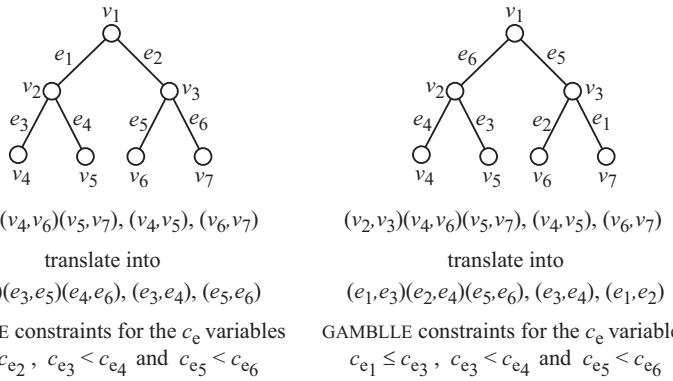


Figure 5: GAMBLLE *constraints for two different edge orderings of a binary tree.*

In the experiments reported in the next section, we consider all permutations obtained using NAUTY and CLUSTERS, and the edges are ordered according to the lexicographical ordering of their pair of endvertices. For example, consider again the P_4 of Figure 1(a). CLUSTERS does not generate any permutation since the graph does not contain any stable set of twins or clique of twins. NAUTY generates permutation $(v_1, v_4)(v_2, v_3)$ that translates into permutation $(\underline{c}_{v_1}, \underline{c}_{v_4})(\underline{c}_{v_2}, \underline{c}_{v_3})$ of the $\underline{c}_v$ variables. The considered cycle C is therefore $(\underline{c}_{v_1}, \underline{c}_{v_4})$ which gives the GAMBLLE constraint $\underline{c}_{v_1} \leq \underline{c}_{v_4}$ that forbids solution s_4. Similarly, with the $\overline{c}_v$ variables, we get the GAMBLLE constraint $\overline{c}_{v_1} \leq \overline{c}_{v_4}$ which forbids solution s_4. For the d_v variables, we get the GAMBLLE constraint $d_{v_1} \leq d_{v_4}$ that does not break any symmetry since $d_{v_1} = d_{v_4} = 0$ in both s_3 and s_4. The permutation of the c_e variables associated with $(v_1, v_4)(v_2, v_3)$ is (c_{e_1}, c_{e_3}), and the associated GAMBLLE constraint $c_{e_1} \leq c_{e_3}$ again forbids s_4.

Consider now the clique K_3 of Figure 1(b). Both CLUSTERS and NAUTY produce permutations (v_1, v_2) and (v_2, v_3). The corresponding GAMBLLE constraints for the $\underline{c}_v$ variables are $\underline{c}_{v_1} \leq \underline{c}_{v_2} \leq \underline{c}_{v_3}$, which remove all optimal solutions except s_1 and s_2. The GAMBLLE constraints for the $\overline{c}_v$ variables are $\overline{c}_{v_1} \leq \overline{c}_{v_2} \leq \overline{c}_{v_3}$ which are only satisfied by solutions s_1 and s_4. With the d_v variables, we get $d_{v_1} \leq d_{v_2} \leq d_{v_3}$ satisfied by s_4 and s_5. The only variables that leave exactly one solution among the six equivalent ones are the c_e variables. Indeed, their associated GAMBLLE constraints are $c_{e_1} < c_{e_2} < c_{e_3}$ which forbid all optimal solutions except s_1. Note that GAMBLLE constraints associated with different sets of permutable variables cannot be combined, since all solutions to the minimum deficiency problem are then possibly forbidden. For example, for the clique K_3, the union of the GAMBLLE constraints $d_{v_1} \leq d_{v_2} \leq d_{v_3}$ and $c_{e_1} < c_{e_2} < c_{e_3}$ forbids all solutions: $d_{v_2} \leq d_{v_3}$ and $c_{e_3} > \max\{c_{e_1}, c_{e_2}\}$ is equivalent to $c_{e_3} - c_{e_1} \leq c_{e_3} - c_{e_2}$ which implies $c_{e_2} \leq c_{e_1}$.

6 Computational experiments

6.1 Experimental setup

To generate GAMBLLE constraints, two parameters come into play. The first one is the method to obtain a set of permutations, either through the NAUTY library (`N`) or the CLUSTERS method (`C`). The second is the family of permutable variables used, which can be the color c_e of the edges (`col`), the minimum color $\underline{c}_v$ at the vertices (`min`), the maximum color $\overline{c}_v$ at the vertices (`max`), or the deficiency d_v at the vertices (`def`). The set of extra constraints and the resulting algorithm when using them are both denoted by putting the letters of the options together in the format `G<method><family>`. Also, `none` denotes the original model solved without any GAMBLLE constraints.

The following two datasets are considered in our experiment: **D1**, the complete set of connected simple graphs of size 4 to 9, and **D2**, a series of random connected simple graphs, 8 for each pair (n, p) with $4 \leq n \leq 100$ vertices and density in $(p - 0.05, p + 0.05]$ with $p \in \{0.1, 0.2, \ldots, 0.9\}$.

The tests were run on a Lenovo Thinkpad X300 laptop, with Intel Core 2 Duo CPU at 1.2 GHz and 4Gb of RAM. Given a graph and a family of permutable variables, the pre-processing step generates the GAMBLLE constraints, using either NAUTY (v2.4r2) or CLUSTERS. Using IBM/ILOG's optimization suite, the basic model is expressed in *OPL* (Optimization Programming Language). It is then instanciated with the graph and augmented with the additional constraints. This object is solved with either *cp optimizer* (v12.2) to competitively find the deficiency, or *CPLEX* (v12.2) to find the list of optimal solutions. For the latter the generation of graph images in post-processing relies on *Graphviz*. This whole process and the batch processing are both orchestrated by programs written in *Ruby*.

6.2 clusters versus nauty

Given a graph G, it may happen that $Aut_{\mathcal{C}}(G)$ is strictly contained in $Aut(G)$. Also, we possibly have $Aut_{\mathcal{C}}(G) = \{Id\}$, which means that G does not contain any stable set of twins or clique of twins. In order to justify the use of CLUSTERS, we first show that most graphs have $Aut_{\mathcal{C}}(G) = Aut(G)$. For this purpose, we distinguish the following four cases, denoted A1, A2, A3 and A4.

	$1 = \lvert Aut_{\mathcal{C}}(G)\rvert$	$1 < \lvert Aut_{\mathcal{C}}(G)\rvert$
$Aut_{\mathcal{C}}(G) = Aut(G)$	A1 `GC<fam>` $\equiv$ `GN<fam>` $\equiv$ `none`	A2 `GC<fam>` $\equiv$ `GN<fam>`
$Aut_{\mathcal{C}}(G) \subsetneq Aut(G)$	A3 `GC<fam>` $\equiv$ `none`	A4

The graph $K_{2,3}$ of Figure 3 is in A2 since both CLUSTERS and NAUTY produce the 3 generators for the 12 permutations in $Aut(K_{2,3})$. The graph in Figure 4 belong to A3 since CLUSTERS does not produce any permutation while NAUTY does. The graph in Figure 2 is in A4 since NAUTY produces permutation $(v_5, v_7)(v_6, v_8)$ that does not belong to $Aut_{\mathcal{C}}(G)$ and $|Aut_{\mathcal{C}}(G)| = 24$. An example of graph in A1, is shown in Figure 6.

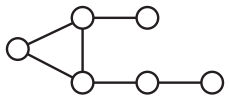
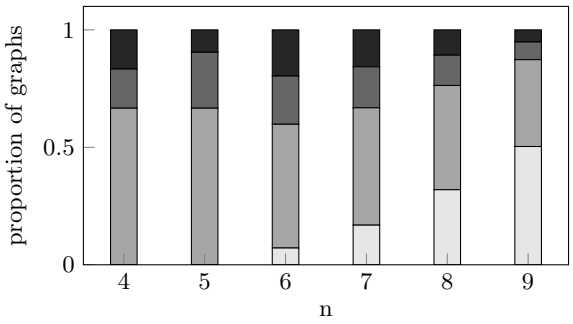


Figure 6: The smallest graph G with $Aut_{\mathcal{C}}(G) = Aut(G) = \{Id\}$.

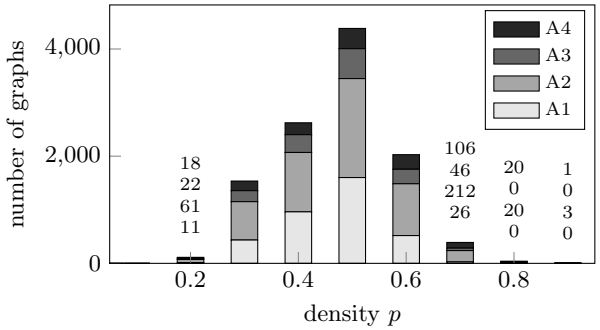
When G is in class A1 or A3, `GC<family>` is equivalent to `none`, and when G is in A1 or A2, `GC<family>` and `GN<family>` produce the same results.

As shown in Figure 7, the proportion of graphs in **D1** that belong to A1 increases monotonically with the the number of vertices. This becomes even more clear in Figure 8 for the random graphs in **D2**. When $|Aut(G)| > 1$, class A2 seems to dominate, which suggests that most automorphisms are due to stable sets of twins and cliques of twins.

In **D1**, when fixing the number n of vertices and varying the density d, the middle range of density tends to have a bigger proportion of graphs with no automorphism, contrary to the extremes that mostly have some. Figure 7 shows the case of $n = 8$. This observation seems to stay true for larger graphs (see Figure 8 with $n = 20$).



(a) Distribution for every $n \in \{4, 5, \ldots, 9\}$.

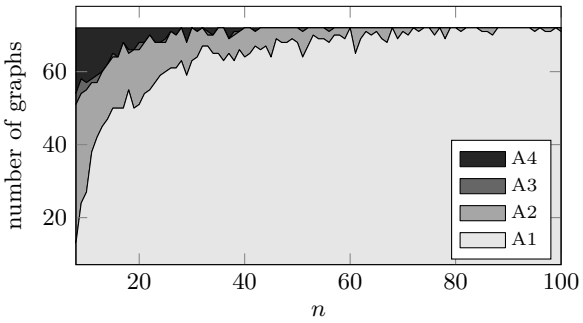


(b) Distribution for $n = 8$ and $p \in \{0.1, 0.2, \ldots, 0.9\}$.

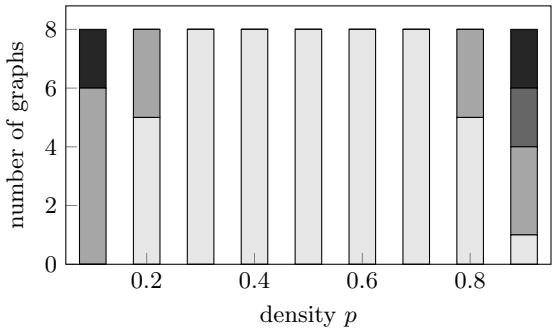
*Figure 7: Distribution of the automorphism group classes for dataset **D1**.*

Let $g_N(G)$ (respectiveley $g_C(G)$) denote the number of generators produced by NAUTY (respectively CLUSTERS) when applied to G. Figures 7 and 8 clearly show that most graphs belong to A1 and A2, in which case we have $g_N(G) = g_C(G)$. If G belongs to A3 or A4, $g_C(G)$ tends to be usually only slightly smaller than $g_N(G)$, as shown in Table 1, where we indicate the number of graphs G with $n = 8$ vertices for every pair $(g_N(G), g_C(G))$. Indeed among the 11117 graphs, only 228 of them (i.e., 2%) have $g_N(G) - g_C(G) > 1$. By summing the numbers on the diagonal, we obtain that 8565 graphs out of 11117 (i.e. 77%) belong to A1 or A2. These observations justify the use of CLUSTERS since $Aut_{\mathcal{C}}(G) = Aut(G)$ in a majority of cases.

Using again **D1**, we analyze in Table 2 the relationship between the minimum deficiency $d(G)$ and the automorphism group class $a(G) \in \{\text{A1,A2,A3,A4}\}$ to which G belongs, as well as the relationship between $d(G)$ and the number $g_N(G)$ of generators produced by NAUTY. We indicate the percentage of graphs



(a) Distribution for every $n \in \{4, 5, \ldots, 100\}$.



(b) Distribution for $n = 20$ and $p \in \{0.1, 0.2, \ldots, 0.9\}$.

*Figure 8: Distribution of the automorphism group classes for dataset **D2**.*

$g_C(G)$ \ $g_N(G)$	0	1	2	3	4	5	6	7
7								1
6							8	1
5						39	4	
4					177	22	1	1
3				555	71	1		
2			1408	263	14			
1		2825	690	48	2			
0	3552	1273	144	16	1			

Table 1: $g_N(G)$ versus $g_C(G)$ for $n = 8$

with $n = 8$ vertices for every pair $(d(G), g_N(G))$ and every pair $(d(G), a(G))$. Notice first that most graphs (97.7%) have minimum deficiency 0. Among the 257 graphs (2.31%) with $d(G) > 0$, only 6 of them (0.05 % of 11117, which is also 2.33% of 257) have no automorphism (i.e., belong to A1). Also, it appears

that the average value of $g_N(G)$ tends to increase when the minimum deficiency increases : it is equal to 1.11 for graphs with $d(G) = 0$, to 2.39 for graphs with $d(G) = 1$, and to 4.5 for graphs with $d(G) = 2$.

$d(G)$	$g_N(G)$								automorphism class			
	0	1	2	3	4	5	6	7	1	2	3	4
0	31.9	36.34	19.46	7.33	2.1	0.44	0.1	0.03	31.9	43.28	12.48	10.04
1	0.05	0.52	0.71	0.6	0.27	0.1	0.02		0.05	1.08	0.42	0.72
2					0.02	0.02				0.03		0.01

Table 2: $d(G)$ versus $g_N(G)$ and the automorphism group classes.

6.3 Impact on the size of the optimal solution space

In this section, we analyze the impact of adding symmetry breaking constraints on the size of the set of optimal solutions. For this purpose, let $L_{\texttt{none}}(G)$ be the set of optimal solutions to the minimum deficiency problem when no extra constraints are added to the constrained programming model of Section 2. Such a set can be represented by a graph $H_{\texttt{none}}(G)$ where each vertex is a solution in $L_{\texttt{none}}(G)$, and two solutions are adjacent if one can be transformed into the other by swapping the two colors along a path or cycle of alternating colors. These are the two most fundamental neighborhoods used in heuristics for solving the minimum deficiency problem [4]. They leave unaltered the deficiency in their interior vertices, so there can only be a potential effect on the deficiency at the two ends of a path. Consider for example the graph in Figure 9. The coloring on the leftside is optimal since the deficiency is zero. A swapping of colors 1 and 2 on the cycle containing vertices v_1, v_2, v_5, v_6 produces a new optimal solution. These two solutions are equivalent, one being obtained from the other by permuting vertices v_2 and v_5 which belong to a stable set of twins. The two solutions are therefore linked by an edge in $H_{\texttt{none}}(G)$. A swapping of colors 1 and 2 on a path is shown on the rightside of Figure 9. In this case, we obtain a deficiency at vertex v_8. Hence, this third coloring is not optimal and therefore not represented in $H_{\texttt{none}}(G)$.

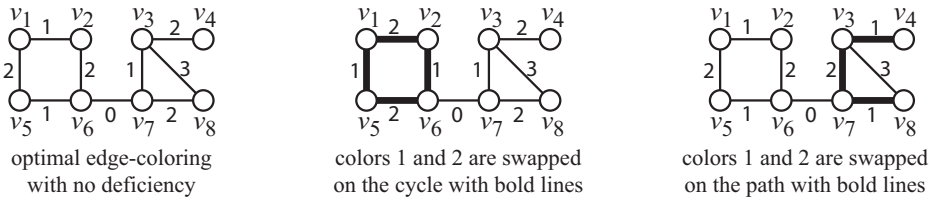


Figure 9: Illustration of swappings on cycles and paths.

Let `alg` be any of the proposed algorithms. Since `alg` adds constraints to the original constrained programming model, it ideally forbids some solutions in $L_{\texttt{none}}(G)$ to produce a subset $L_{\texttt{alg}}(G)$ of optimal solutions. Therefore, the graph $H_{\texttt{alg}}(G)$ representing the links between the optimal solutions when the extra constraints produced by `alg` are taken into account is always an induced subgraph of the original graph $H_{\texttt{none}}(G)$.

One way to compare the performance of the algorithms is to observe their impact of their respective extra constraints on the solution space, and in particular on the the sets $L_{\texttt{alg}}(G)$ of optimal solutions. In the best case, a set of symmetry breaking constraints can at most divide the size of the original optimal solution space by $|Aut(G)|$. Hence,

$$\frac{|L_{\texttt{none}}(G)|}{|Aut(G)|} \leq |L_{\texttt{alg}}(G)| \leq |L_{\texttt{none}}(G)|.$$

These bounds are not tight. Indeed, consider for example the P_4 in Figure 1(a). It has two automorphisms, the identity permutation and $(v_1\ v_4)(v_2\ v_3)$, and four optimal solutions. As already mentioned, the last solution is equivalent to the third one and is removed by our algorithms. Hence, every proposed algorithm `alg` breaks all symmetries while we have

$$\frac{|L_{\texttt{none}}(P_4)|}{|Aut(P_4)|} = \frac{4}{2} < 3 = |L_{\texttt{alg}}(P_4)| < 4 = |L_{\texttt{none}}(P_4)|.$$

This means that when the lower bound is reached, we have the guarantee that all symmetries due to graph automorphisms have been broken, which is our objective. But it is also possible to attain that goal and still not reach the lower bound.

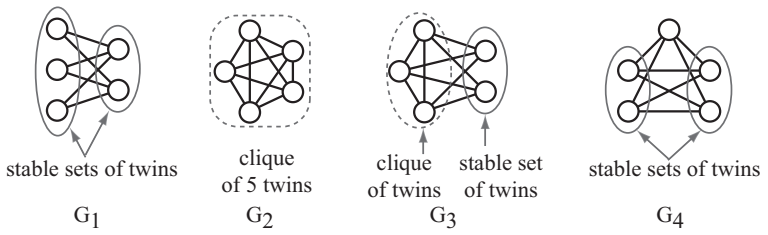


Figure 10: Four graphs with 5 vertices.

We first consider the four graphs of Figure 10 with five vertices. The first one is the $K_{2,3}$ of Figure 3. The second one is a clique on 5 vertices, and hence a clique of twins. The third one is obtained from G_2 by removing one edge, and contains a clique of twins and a stable set of twins. The fourth one is obtained from G_2 by deleting two disjoint edges, and contains two stable sets of twins. The results of our algorithms are shown in Table 3. On the left part of the table, we indicate for every graph G_i the class to which its belong (i.e., A1, A2, A3 or A4), the sizes of the cluster-automorphism $Aut_{\mathcal{C}}(G_i)$ and of the automorphism group $Aut(G_i)$, and the minimum deficiency $d(G_i)$. On the rightside of the

Table, we indicate the size of the sets $L_{\texttt{alg}}(G_i)$. The optimal solution spaces $H_{\texttt{alg}}(G_i)$ are shown in Figures 11, 12, 13 and 14. Solid edges correspond to a swapping on a path, while dashed edges correspond to a swapping on a cycle. The width of each edge is proportional to the number of vertices in the path or cycle.

graph	class	$\lvert Aut_C(G_i)\rvert$	$\lvert Aut(G_i)\rvert$	$d(G_i)$	none	G_col		G_min		G_max		G_def	
						C	N	C	N	C	N	C	N
G_1	A2	12	12	0	12	1	1	1	1	1	1	12	12
G_2	A2	120	120	2	720	18	18	24	24	24	24	96	96
G_3	A2	12	12	1	96	8	8	22	22	22	22	32	32
G_4	A4	4	8	1	48	16	8	12	10	16	8	32	32

Table 3: Size of the optimal solution spaces for the proposed algorithms.

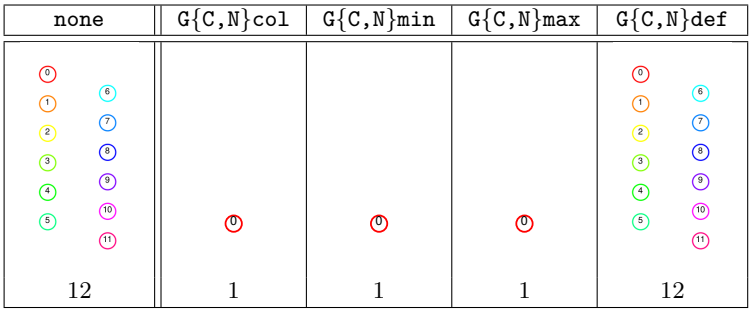


Figure 11: Optimal solutions spaces $H_{alg}(G_1)$ for G_1.

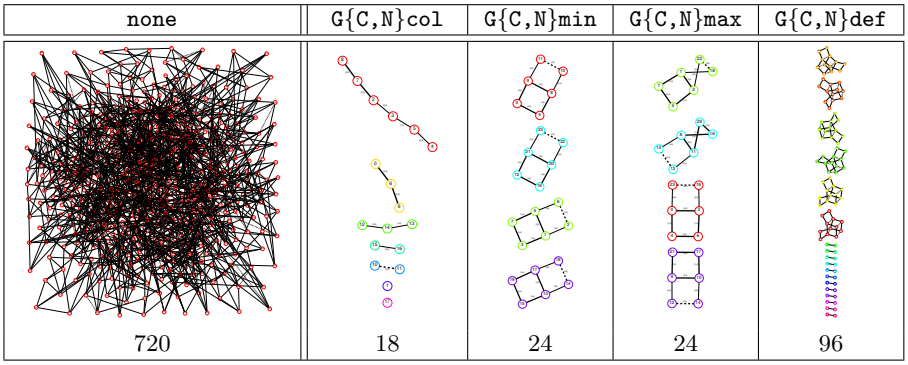


Figure 12: Optimal solution spaces $H_{alg}(G_2)$ for G_2.

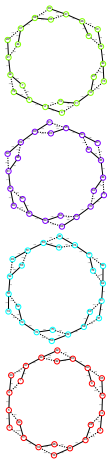
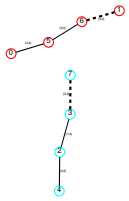
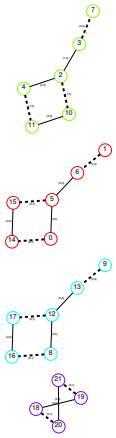
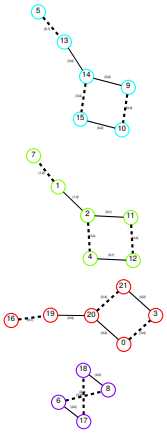
none	G{C,N}col	G{C,N}min	G{C,N}max	G{C,N}def
				
96	8	22	22	32

Figure 13: Optimal solution spaces $H_{alg}(G_3)$ for G_3.

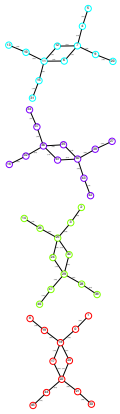
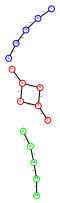
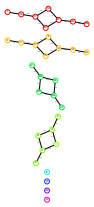
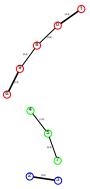
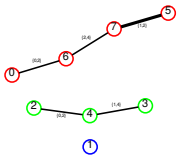
	none	G_col	G_min	G_max	G_def
C					
		16	12	16	32
N					
	48	8	10	8	32

Figure 14: Optimal solution spaces $H_{alg}(G_4)$ for G_4.

For the first graph $G_1 = K_{2,3}$, the 12 original optimal solutions are all permutations of the same one, and G{C,N}col, G{C,N}min and G{C,N}max are able to break all symmetries, as reaching the bound $\frac{|L_{\text{none}}(G_1)|}{|Aut(G_1)|} = \frac{12}{12} = 1$ gives us

that guarantee. This can be observed in Figure 11 where three optimal solution spaces contain only one solution.

The best illustration of the decrease of the size of the optimal solution space is given by G_2. The optimal solution space $H_{\texttt{none}}(G_2)$ at the leftside of Figure 12 clearly shows that many of the 720 optimal solutions are linked with each other. For comparison, $H_{\texttt{GNcol}}(G_2)$ and $H_{\texttt{GCcol}}(G_2)$ contain only 18 vertices grouped into 7 connected components.

Graphs G_2, G_3 and G_4, help to illustrate how the removal of an edge can change the landscape of the solution space dramatically. Note that the bound $\frac{|L_{\texttt{none}}(G_3)|}{|Aut(G_3)|} = \frac{96}{12} = 8$ is reached for G_3 with `G{C,N}col`. The fourth graph G_4 illustrates a case where $1 < |Aut_{\mathcal{C}}(G)| < |Aut(G)|$. Predictably, the NAUTY method gives a smaller set of optimal solutions than CLUSTERS.

The `col` family of permutable variables is clearly the most efficient one for breaking symmetries. This is probably because it is the only one constraining the variables representing the color on an edge (the key decision variables of our problem), as well as the only one that has sometimes strengthened constraints. On the opposite end, using the `def` family of permutable variables has a much smaller impact on the reduction of the optimal solution space. The other two families `min` and `max` seem to have comparable performances, between the aforementioned two extremes.

6.4 Decrease of the computing time

We now consider six larger graphs for comparing the computing times needed to solve the constrained programming model with or without the extra constraints. The six graphs $G_5, \ldots, G_{10}$ are shown in Figure 15. G_5 and G_6 are cliques with 6 and 7 vertices, respectively. G_7 is obtained from G_6 by removing one edge. G_8 (respectively G_9) is obtained from G_6 by removing two incident (respectively disjoint) edges, while G_{10} is obtained from G_6 by removing 3 disjoint edges. These graphs contain cliques of twins shown using boxes with dashed lines, and stable sets of twins shown using boxes with plain lines.

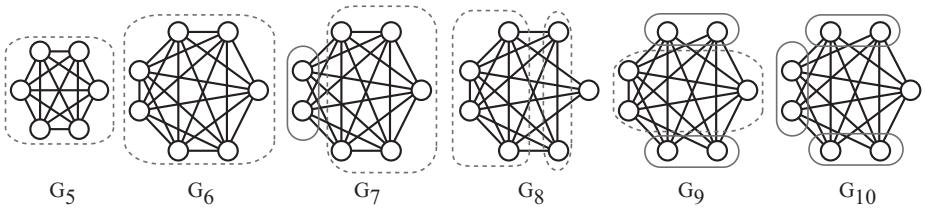


Figure 15: Six graphs with 6 and 7 vertices.

All our previous general observations on the relative performance of the algorithms for the size of the optimal solution space remain true in the case of computing times, except that the methods based on the `min` family of permutable variables are slightly superior to those with the `max` family. Again,

graph	class	$\|Aut_{\mathcal{C}}(G_i)\|$	$\|Aut(G_i)\|$	$d(G_i)$	none	G_col		G_min		G_max		G_def	
						C	N	C	N	C	N	C	N
G_5	A2	720	720	0	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
G_6	A2	5040	5040	3	>7200	11.5	11.5	415.0	415.0	791.3	791.3	1631.0	1631.0
G_7	A2	240	240	2	298.3	2.8	2.8	91.2	91.2	93.1	93.1	177.5	177.5
G_8	A2	48	48	1	7.2	0.2	0.2	3.6	3.6	3.6	3.6	10.0	10.0
G_9	A4	24	48	1	11.5	0.5	0.3	8.8	7.6	8.9	6.4	12.2	12.9
G_{10}	A4	8	48	2	34.6	7.9	2.6	21.1	5.2	18.7	7.7	31.3	30.5

Table 4: Computing times (in seconds) for six graphs.

`col` gives the best results, with computing times reduced by at least one order of magnitude. Also, NAUTY consistently outperforms CLUSTERS, but within the same order of magnitude. Thus, `GNcol` clearly emerges as the best algorithm, combining successfully two ideas, the choice of the family of permutable variables and the method for finding a set of generators for the automorphism group. The graph with larger automorphism group is the clique G_6 on seven vertices with $|Aut(G_6)| = 5040$. While the original constrained programming model does not find a proven optimal solution in 2 hours of computation, `GNcol` solves the problem in 11 seconds.

The impact of the various methods on the computing time is also shown in Table 5 where we analyze the total time needed to solve the minimum deficiency problem for all graphs with $n = 4, 5, 6, 7, 8$ vertices. We do not report the results for $n = 9$ since among the 261084 graphs with 9 vertices, the CP solver has not produced proven optimal solutions within 2 hours of computation for about 100 of them.

n	none	G_col		G_min		G_max		G_def	
		C	N	C	N	C	N	C	N
4	<0.01	<0.01	0.01	0.02	0.01	0.03	0.03	0.03	0.02
5	0.12	0.06	0.06	0.07	0.08	0.1	0.1	0.12	0.12
6	0.3	0.34	0.35	0.42	0.5	0.48	0.47	0.43	0.43
7	4403.9	22.1	17.2	327.8	314.42	541.0	530.8	1092.3	1092.7
8	4702.9	249.4	203.4	1406.6	1290.8	1529.9	1415.3	4526.6	4546.8

Table 5: Total computing times (in seconds) to solve all the graphs with $4, 5, 6, 7$ *and* 8 *vertices*

The orientation of the *less than or equal* ($\leq$) inequality constraints used in section 5 to generate GAMBLLE constraints has an impact on the performance of the algorithms, as it interacts with both the constraint that forces the usage of color 0, and the rule to choose the variable with the smallest index. Also, as already mentioned, the smallest index rule makes the GAMBLLE constraints very sensitive to the labelling of the edges for the algorithms based on the `col` family of permutable variables. This is shown in Table 6 where we compare three different settings for graph G_9. The first case (called $G_9(\leq)$) is the original one, where the vertices are labelled as shown at the leftside of Figure 16, the edges are labelled according to the lexicographical ordering of their pair of endvertices,

and a cyclic permutation C with smallest indexed variable u_i leads to inequalities $u_i \leq u_j$ or $u_i < u_j$ for all u_j in C with $j \neq i$. The second test (called $G_9(\geq)$) uses inequalities $u_i \geq u_j$ or $u_i > u_j$ instead of the $\leq$ or $<$ orignal ones. The third test (called G_9(REV)) uses the $\leq$ or $<$ inequalities, but considers the reverse labelling of the vertices with the corresponding labelling of the edges (according to the lexicographical ordering of their pair of endvertices) as shown at the rightside of Figure 16. While the models remain valid with these changes, we observe that the performance may significantly drop when we do not use the original setting. For example, the problem is solved in 7 seconds with the original settings, while the use of the $\geq$ or $>$ inequalities increases the computing time to 100 seconds.

graph	none	G_col		G_min		G_max		G_def	
		C	N	C	N	C	N	C	N
G_9 ($\leq$)	11.58	0.58	0.37	8.82	7.63	8.90	6.48	12.20	12.94
G_9 ($\geq$)	11.58	6.48	4.04	116.18	100.15	94.93	51.41	14.13	12.77
G_9 REV	17.20	2.75	1.24	12.33	10.96	11.45	10.28	39.52	40.28

Table 6: Computing times for three variants of the proposed algorithms applied to G_9.

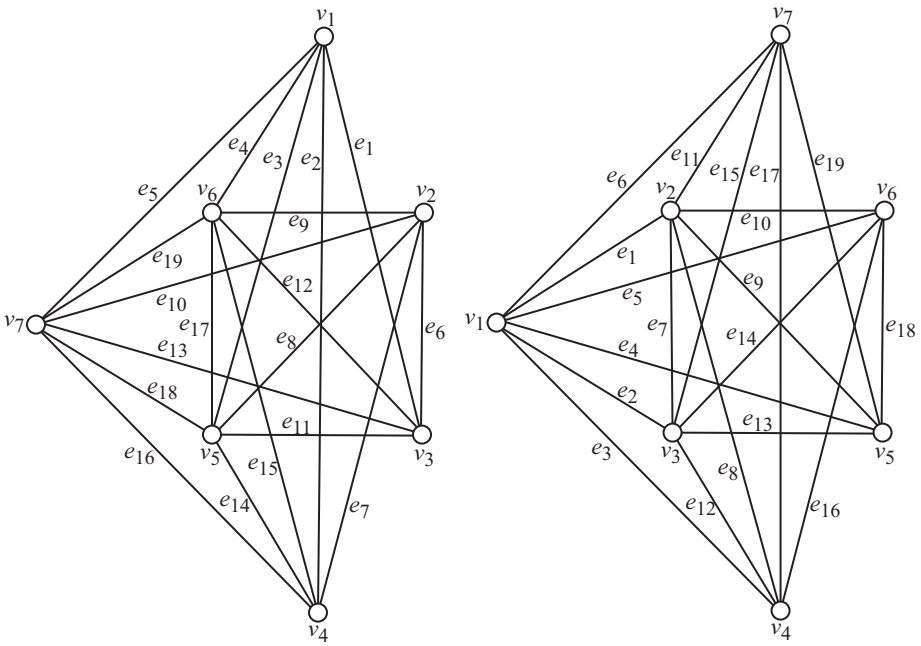


Figure 16: Two labellings of the vertices and of the edges of G_9.

7 Conclusion

The generation of GAMBLLE constraints is a general technique to help solving CP models of graph optimization problems. In the present paper, its potency is demonstrated through an application to the minimum deficiency problem.

A straightforward CP model to find the exact deficiency is hindered by an overwhelming number of equivalent optimal colorings, in part due to the automorphisms of the considered graph. When included in the model, the GAMBLLE constraints help to cut down the solution space by forbidding some equivalent optimal solutions, and thus improve the time performance of the solver. The total number of extra constraints generated remains very small, since it only depends on the number of generators found for the automorphism group, which is in the order of n in the worst case. These generators can be obtained using the famous NAUTY library created by Brendan McKay[16, 17], which possibly requires an exponential computing time. Another possibility is to use the proposed procedure CLUSTERS, which generates in polynomial time a set of generators for a subgroup $Aut_{\mathcal{C}}(G)$ of the automorphism group $Aut(G)$.

Experiments have shown that most graphs have $Aut_{\mathcal{C}}(G) = Aut(G)$, which means that most symmetries are due to cliques and stable sets of twins, and justifies the use of CLUSTERS. Also, four families of permutable variables have been considered, and we have noticed that the best results are obtained with GAMBLLE constraints based on the color of the edges (`col`). We have shown that the `GNcol` and `GCcol` algorithms drastically decrease the size of the optimal solution space, and improve by at least one order of magnitude the basic model. The proposed algorithms are particularly efficient for graphs that have a lot of symmetries.

As last comment, we mentioned in Section 2 the conjecture that the minimum deficiency $d(G)$ of a graph G with n vertices is always at most equal to n. No counterexample was found during our experiments, and the question therefore remains open.

References

[1] S. Altinakar, G. Caporossi, and A. Hertz. A comparison of integer and constraint programming models for the deficiency problem. *Computers & Operations Research*, 68:89–96, 2016. doi:10.1016/j.cor.2015.10.016.

[2] A. Asratian and C. Casselgren. On interval edge colorings of (α, β)-biregular bipartite graphs. *Discrete Mathematics*, 307:1951–1956, 2006. doi:10.1016/j.disc.2006.11.001.

[3] A. Asratian and R. Kamalian. Interval colorings of the edges of a multigraph. *Applied Mathematics*, 5:25–34, 1987. (in Russian).

[4] M. Bouchard, A. Hertz, and G. Desaulniers. Lower bounds and a tabu search algorithm for the minimum deficiency problem. *Journal of Combinatorial Optimization*, 17:168–191, 2009. doi:10.1007/s10878-007-9106-0.

[5] D. Cohen, P. Jeavons, C. Jefferson, K. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems constraints. *Constraints*, 11:115–137, 2006. doi:10.1007/s10601-006-80.

[6] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, California, 1996.

[7] I. Gent, K. Petrie, and J.-F. Puget. Symmetry in constraint programming. In F. Rossi, P. van Beek, and W. T., editors, *Handbook of Constraint Programming*, pages 329–376. Elsevier, Amsterdam, 2006.

[8] K. Giaro. The complexity of consecutive δ-coloring of bipartite graphs: 4 is easy, 5 is hard. *Ars Combinatoria*, 47:287–298, 1997.

[9] K. Giaro. Interval edge-coloring of graphs. In M. Kubale, editor, *Graph Colorings*, pages 105–122. American Mathematical Society, 2004.

[10] K. Giaro, M. Kubale, and M. Malafiejski. On the deficiency of bipartite graphs. *Discrete Applied Mathematics*, 94:193–203, 1999. doi:10.1016/S0166-218X(99)00021-9.

[11] K. Giaro, M. Kubale, and M. Malafiejski. Consecutive colorings of the edges of general graphs. *Discrete Mathematics*, 236:131–143, 2001. doi:10.1016/S0012-365X(00)00437-4.

[12] D. Hanson and C. Loten. A lower bound for interval colouring of bi-regular bipartite graphs. *Bulletin of the Institute of Combinatorics and Applications*, 18:69–74, 1996.

[13] D. Hanson, C. Loten, and B. Toft. On interval colorings of bi-regular bipartite graphs. *Ars Combinatoria*, 50:23–32, 1998.

[14] E. Luks and A. Roy. The complexity of symmetry-breaking formulas. *Annals of Mathematics and Artificial Intelligence*, 41:19–45, 2004. doi:10.1023/B:AMAI.0000018578.92398.10.

[15] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8:131–136, 1979. doi:10.1016/0020-0190(79)90004-8.

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

[17] B. McKay and P. A. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

[18] T. Miyazaki. *The complexity of McKay's canonical labeling algorithm*, volume 28 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 239–256. American Mathematical Society, Providence, 1997.

[19] J.-F. Pujet. Breaking symmetries in all different problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 272–277, 2005.

[20] A. Pyatkin. Interval coloring of $(3, 4)$-biregular bipartite graphs having large cubic subgraphs. *Journal of Graph Theory*, 47:122–128, 2004. doi:10.1002/jgt.20021.

[21] J. Rotman. *Advanced Modern Algebra*. Prentice-Hall, 2002.

[22] A. Schwartz. The deficiency of a regular graph. *Discrete Mathematics*, 306:1947–1954, 2006. doi:10.1016/j.disc.2006.03.059.

[23] S. Sevastianov. On interval edge colouring of bipartite graphs. *Metody Diskretnowo Analiza*, 50:61–72, 1990. (in Russian).