Alcom-FT Technical Report Series
ALCOMFT-TR-01-123

Detecting Symmetries by Branch & Cut*

Christoph Buchheim and Michael Jünger

Universität zu Köln, Institut für Informatik,
Pohligstraße 1, 50969 Köln, Germany
{buchheim,mjuenger}@informatik.uni-koeln.de

Abstract. We present a new approach for detecting automorphisms and symmetries of an arbitrary graph based on branch & cut. We derive an IP-model for this problem and have a first look on cutting planes and primal heuristics. The algorithm was implemented within the ABACUS-framework; its experimental runtimes are promising.

1 Introduction

The display of symmetries is one of the most desirable properties of a graph drawing [7]. Each recognizable symmetry reduces the complexity of the drawing for the human viewer, see Fig. 1. Furthermore, the existence of a symmetry can be an important structural property of the data being displayed.

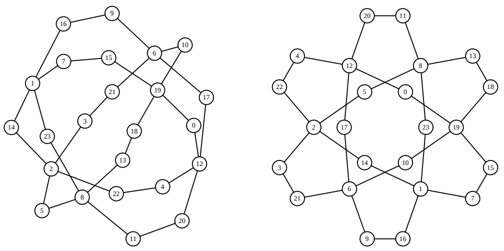

Fig. 1. Two drawings of the same abstract graph, both computed by a spring embedder. On the left, the initial drawing was chosen randomly; on the right, the displayed symmetries have been computed explicitly before

Since detecting symmetries of an arbitrary graph is NP-hard [6], most algorithms are either restricted to special classes of graphs [4] or have an implicit tendency to display symmetries while failing to find them in general [2, 3]. Explicit algorithms consist of two steps: First, compute an abstract symmetry

* This work was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

of the graph, i.e., an automorphism that can be displayed by some drawing of the graph in the plane; second, compute such a drawing with regard to the usual aesthetical requirements like a small number of edge crossings or edge bends. In this article, we focus on the first step, presenting an algorithm that can be applied to every graph and surely finds its best symmetry, i.e., a symmetry of maximum order. The algorithm uses branch & cut to solve an integer programming (IP) formulation of the symmetry detection problem. By the NP-hardness of the problem, one cannot expect polynomial runtime, but the experimental runtimes are promising.

In the next section, we list the required definitions and facts concerning automorphisms and symmetries. In Sect. 3, we derive the IP-model for symmetry detection in arbitrary graphs. In Sect. 4, we explain the technique of node labeling that improves our algorithm at many points. The components of the branch & cut-approach are presented in Sect. 5. After discussing experimental runtimes in Sect. 6 and possible extensions in Sect. 7, we summarize in Sect. 8 what has been achieved and what can still be done.

2 Preliminaries

Throughout this article, a *graph* is a simple undirected graph $G = (V, E)$ with $n = |V|$, see Sect. 7 for other types of graphs.

An *automorphism* of G is a permutation π of V with $(i, j) \in E$ if and only if $(\pi(i), \pi(j)) \in E$ for all $i, j \in V$. The set of automorphisms of G forms a group with respect to composition, denoted by $\mathrm{Aut}(G)$. The *order* of $\pi \in \mathrm{Aut}(G)$ is $\mathrm{ord}(\pi) = \min\{k \in \mathbf{N} \mid \pi^{\mathbf{k}} = \mathrm{id}_{\mathbf{V}}\}$, where id_V denotes the identity permutation of V. For a node $i \in V$, the set $\mathrm{orb}_\pi(i) = \{\pi^k(i) \mid k \in \mathbf{N}\}$ is the π-*orbit* of i. Finally, the *fixed* nodes are those in $\mathrm{Fix}(\pi) = \{i \in V \mid \pi(i) = i\}$.

A *reflection* of G is an automorphism $\pi \in \mathrm{Aut}(G)$ with $\pi^2 = \mathrm{id}_V$, i.e., an automorphism of order 1 or 2. For $k \in \{1, \dots, n\}$, a k-*rotation* of G is an automorphism $\pi \in \mathrm{Aut}(G)$ such that $|\mathrm{orb}_\pi(i)| \in \{1, k\}$ for all $i \in V$ and $|\mathrm{Fix}(\pi)| \leq 1$ if $k \neq 1$. Observe that each 2-rotation is a reflection, but not vice versa, and that the identity id_V is both a reflection and a 1-rotation.

If there exists a drawing of G in the plane and an isometry of the plane that maps nodes to nodes and edges to edges, this drawing induces an automorphism of G. Any automorphism induced like this is called a *geometric automorphism* or a *symmetry* of G. We have the following characterization, for a proof see [2]:

Lemma 1. *An automorphism of a graph G is a symmetry if and only if it is a rotation or a reflection.*

Obviously, the symmetries displayed by a single drawing of G form a subgroup of $\mathrm{Aut}(G)$, but the set of all symmetries of G is not closed under composition. The following lemma characterizes the sets of symmetries that can be displayed together, for a proof see [2] again:

Lemma 2. *A set of automorphisms can be displayed by a single drawing of G if and only if it generates a subgroup of $Aut(G)$ that is generated by a single symmetry or by a rotation π_1 and a reflection π_2 satisfying $\pi_2\pi_1 = \pi_1^{-1}\pi_2$.*

3 The IP-Model

In this section, we derive an integer linear program (ILP) modeling the set of symmetries of the given graph $G = (V, E)$. First observe that permutations of V can be described by the following program:

$$\begin{aligned} x_{ij} &\in \{0,1\} && \text{for all } i,j \in V \\ \textstyle\sum_{j\in V} x_{ij} &= 1 && \text{for all } i \in V \\ \textstyle\sum_{i\in V} x_{ij} &= 1 && \text{for all } j \in V\,. \end{aligned} \tag{1}$$

A value of 1 for a variable x_{ij} is interpreted as mapping node i to node j. The equations of the first type make sure that each node i is mapped to exactly one node j, so that the variables induce a function $V \to V$. By the equations of the second type, this function is bijective.

Observe that by relaxing the integrality constraints in (1) we get the assignment polytope. All vertices of this polytope have only integer components. Unfortunately, this does not remain true if the model is restricted to automorphisms. It is easy to see that an automorphism of G is a permutation of V such that the corresponding matrix $X = (x_{ij})$ commutes with the adjacency matrix A_G of G. Hence automorphisms are singled out by adding the n^2 equations $A_G X = X A_G$ to (1).

Next, we model the characterization of Lemma 1 to restrict the search to symmetries. For this, we introduce new variables $l_k \in \{0,1\}$ for $k \in \{1,\ldots,n\}$, where a value of 1 for l_k means an order of k for the symmetry induced by the variables x_{ij}. Besides requiring $\sum_{k=1}^{n} l_k = 1$, we first have to ensure that all orbits have length one or k if $l_k = 1$. For this, let $C = (i_1, \ldots, i_p)$ be a cycle in V with $p \geq 2$, and define $x(C) = \sum_{r=1}^{p} x_{i_r i_{r+1}}$ with $i_{p+1} = i_1$. Then

$$x(C) \leq p - 1 + l_p \tag{2}$$

is valid for any symmetry, since we always have $x(C) \leq p$, and $x(C) = p$ implies that C is an orbit of length $p \geq 2$, hence $l_p = 1$. The set of all inequalities of type (2) guarantees that all non-trivial orbits have the size given by the variables l_k. Since there are exponentially many cycles of V, we have to add these inequalities in the separation phase, see Sect. 5.3.

Finally, we have to deal with the fixed nodes. If $l_k = 1$ for $k \geq 3$, the number $\sum_{i\in V} x_{ii}$ of fixed nodes is at most one by definition of a rotation; if $l_2 = 1$, it is at most $n - 2$. Summing up, the following ILP characterizes symmetries:

$$\begin{aligned} x_{ij} &\in \{0,1\} && \text{for all } i,j \in V \\ l_k &\in \{0,1\} && \text{for all } k \in \{1,\ldots,n\} \\ \textstyle\sum_{j\in V} x_{ij} &= 1 && \text{for all } i \in V \\ \textstyle\sum_{i\in V} x_{ij} &= 1 && \text{for all } j \in V \\ A_G X &= X A_G && \\ \textstyle\sum_{k=1}^{n} l_k &= 1 && \\ x(C) &\leq |C| - 1 + l_{|C|} && \text{for all cycles } C \text{ of } V \text{ with } |C| \geq 2 \\ \textstyle\sum_{i\in V} x_{ii} &\leq (n-1) l_1 + (n-3) l_2 + 1\,. && \end{aligned} \tag{3}$$

It remains to specify an objective function that selects a "best" symmetry. Our goal is to maximize the order of the symmetry first; within the symmetries of maximum order, we want to minimize the number of fixed nodes. Both is realized by the following objective function:

$$\max\ (n+1)\sum_{k=1}^{n} kl_k - \sum_{i\in V} x_{ii}\ .$$

4 Node Labelings

Our approach makes extensive use of *labelings*. A labeling of a graph $G=(V,E)$ is a coloring $c\colon V^2 \to \mathbf{N}$, such that for each automorphism $\pi \in \mathrm{Aut}(G)$ and all nodes $i,j \in V$ we have $c(i,j) = c(\pi(i),\pi(j))$. We define a corresponding *node labeling* $c\colon V \to \mathbf{N}$ by $c(i) = c(i,i)$. For reasons to be explained later, we always prefer a fine labeling, i.e., one with many different colors, to a coarse labeling with less colors. For every graph G, there is a finest node labeling, called the *automorphism partitioning* of G, but its computation is isomorphism-complete [8]. Nevertheless, there are heuristics finding the automorphism partitioning in most cases. We use the algorithm of Bastert [1] to compute a labeling once before starting our branch & cut-process. This algorithm needs $O(n^3 \log n)$ time and $O(n^2)$ space. Observe that we always have a trivial labeling of G by defining

$$c(i,j) = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1 & \text{if } i \neq j \text{ and } (i,j) \in E \\ 2 & \text{if } i = j\ . \end{cases}$$

In the following, we always assume that the computed labeling is at least as fine as the trivial one.

One reason for using node labelings is the following: If $c(i) \neq c(j)$ for two nodes $i,j \in V$, no automorphism will map i to j, by definition of c. Hence the variable x_{ij} in (3) will always be zero, so that we can leave it out from the beginning. This often reduces the size of the ILP immensely. If the automorphism group is trivial and the node labeling assigns a different color to each node, all x-variables except for the x_{ii} can be deleted.

Furthermore, we may be able to omit many of the variables l_k: For a color $c_0 \in \mathbf{N}$, consider the set $P = \{i \in V \mid c(i) = c_0\}$ and let $p = |P|$. Any symmetry π of G with $\mathrm{ord}(\pi) \geq 3$ has at most one fixed node, and all other orbits have length $\mathrm{ord}(\pi)$. By definition, we know that $\pi(P) = P$ for any $\pi \in \mathrm{Aut}(G)$, hence the set $P \setminus \mathrm{Fix}(\pi)$ can be divided into orbits of length $\mathrm{ord}(\pi)$, so that $\mathrm{ord}(\pi)$ must divide $|P \setminus \mathrm{Fix}(\pi)| \in \{p-1, p\}$. For all $k \geq 3$ neither dividing p nor $p-1$, we can leave out l_k, and this holds for any $c_0 \in \mathbf{N}$.

5 The Branch & Cut-Approach

Branch & Cut is a powerful method for solving hard combinatorial optimization problems. Given an ILP-formulation of the problem, branch & cut proceeds as follows: First, the integrality constraints of the ILP are ignored and the remaining *LP-relaxation* is solved. If the computed optimal solution is integer, the algorithm stops, since this is an optimal solution for the ILP, too. Otherwise, there must be a linear inequality that is valid for the original ILP but violated by the computed fractional solution. If such a *cutting plane* can be found by some *separation* algorithm, it is added to the LP-relaxation as a new constraint, and the new LP is solved again. This is the *cutting phase*. If no more cutting planes can be (or shall be) computed, one has to *branch*: A variable x with a fractional LP-value $\overline{x}$ is selected and two subproblems are created. In the first, the constraint $x \leq \lfloor \overline{x} \rfloor$ is added, in the second, $x \geq \lceil \overline{x} \rceil$ is added. Both problems are solved recursively.

Another ingredient is *primal heuristics*. With the help of the fractional LP-solution, these try to find a feasible solution for the ILP. If the LP-solution in a subproblem has an objective value smaller – when maximizing – than the objective value of some feasible solution, this subproblem cannot contain the optimal solution and can thus be deleted.

In this section, we first describe two kinds of cutting planes that are very easy to handle, see Sect. 5.1. In Sects. 5.2 and 5.3, we present more sophisticated classes of cutting planes; the first one is valid for every automorphism, the second one is in particular useful to reject non-geometric automorphisms. In Sect. 5.4, we present primal heuristics.

5.1 Simple Inequalities

The valid inequalities presented in this section are easy to handle; they can be added to ILP (3) from the beginning, since their number is at most n. Hence no separation is necessary. As in Sect. 4, consider $P = \{i \in V \mid c(i) = c_0\}$ for $c_0 \in \mathbf{N}$ and assume $p = |P| \geq 2$. Let π be any symmetry of G with $\mathrm{ord}(\pi) \geq 3$. In Sect. 4, we argued that $\mathrm{ord}(\pi)$ must divide p or $p-1$, since by definition we have $\pi(P) = P$. In the first case, we have $P \cap \mathrm{Fix}(\pi) = \emptyset$, hence we derive

$$\sum_{i \in P} x_{ii} \leq p(l_1 + l_2) + \sum_{k \geq 3,\ k \mid (p-1)} l_k \ .$$

Furthermore, we know that at least ($p \bmod \mathrm{ord}(\pi)$) nodes of P are fixed, since the non-fixed nodes are divided into orbits of length $\mathrm{ord}(\pi)$, hence

$$\sum_{i \in P} x_{ii} \geq \sum_{k=1}^{n} (p \bmod k) l_k$$

is a valid inequality for (3).

5.2 Homomorphism Inequalities

The only constraints in (3) depending on the edges of the graph are the equations $A_G X = X A_G$. In this section, we present a class of cutting planes that focus on the edges, too, and that is equivalent to $A_G X = X A_G$ for integer solutions. Nevertheless, this is not true for fractional solutions. In our branch & cut-algorithm, the best strategy is to use both types of constraints.

Let c be the labeling of G again. Up to now, we only considered the node colors $c(i)$. In the following, we make use of all colors $c(i,j)$. Let $I \subseteq V^2$ be a set of node-pairs such that for all $(i,j),(k,l) \in I$ we have $c(i,k) \neq c(j,l)$. By definition, no automorphism can map both i to j and k to l in this case. Hence the following *homomorphism inequality* is valid for ILP (3):

$$\sum_{(i,j)\in I} x_{ij} \leq 1 \,. \tag{4}$$

We now explain how to find valid constraints of type (4) that are violated by the current LP-solution $\overline{x}_{ij}$. This is a special independent set problem: Consider the graph $G_c = (V^2, E_c)$ with $((i,j),(k,l)) \in E_c$ if and only if $c(i,k) = c(j,l)$. Assign the weight $\overline{x}_{ij}$ to the node (i,j) of G_c. Then (4) is valid if and only if I is an independent set in G_c, and it is violated if and only if the total weight of I is greater than 1. We can thus use any maximum weight independent set heuristic for the heuristical separation of homomorphism inequalities.

Observe that a fine labeling is favorable again: The finer the labeling, the sparser the graph G_c, the larger the feasible sets I, the more restrictive the corresponding inequalities (4). In general, the homomorphism inequalities are no facets of the symmetry polytope. Nevertheless, they perform well, even if the underlying labeling is the trivial one. Leaving out the homomorphism inequalities increases runtime significantly.

5.3 Decomposition Inequalities

The inequalities presented in this section improve and generalize the constraints of type (2) in Sect. 3. They aim at eliminating non-geometric automorphisms and are based on the fact that all non-trivial orbits of a symmetry have the same length.

In the general case, let $P \subseteq V$ be any set of nodes of G and $I \subseteq P^2$. Let $p = |P|$. We can consider $H = (P, I)$ as a directed graph with loops. A *cycle decomposition* of H is a set of node-disjoint directed cycles in H, such that every node is contained in one of these cycles. Now define $K(H)$ as the set of all $k \in \{1, \ldots, n\}$ such that there exists a cycle decomposition of H with all cycles of length one or k and, for $k \geq 3$, at most one cycle of length one. Using these definitions, we have the following *decomposition inequality*:

$$\sum_{(i,j)\in I} x_{ij} \leq p - 1 + \sum_{k \in K(P,I)} l_k \,. \tag{5}$$

Indeed, the sum on the left hand side is at most p. If it is equal to p, the symmetry π induced by the x_{ij} satisfies $\pi(P) = P$. Hence by definition of $K(P, I)$ we must have $l_k = 1$ for some $k \in K(P, I)$.

The constraints of type (2) are special decomposition inequalities: Consider a cycle $C = (i_1, \ldots, i_p)$ of V again. For the ease of exposition, we will view C as a function by defining $C(i_r) = i_{r+1}$ for $r < p$ and $C(i_p) = i_1$. If we set $P = \{i_1, \ldots, i_p\}$ and $I = \{(i, j) \in P^2 \mid C(i) = j\}$, we have $K(P, I) = \{p\}$, so in this case (5) becomes (2). But, without generalization, these constraints perform poorly. We will illustrate this by an example. Assume that the current LP-solution has an orbit of length five, say $C = (1, 2, 3, 4, 5)$, but l_5 is zero. The constraint $x(C) \leq 4 + l_5$ will not allow this; after adding it to the LP, the additional value of one often escapes to the cycle $C^2 = (1, 3, 5, 2, 4)$ in the following sense: In the next LP-solution, we get $x_{i,C(i)} = \frac{4}{5}$ and $x_{i,C^2(i)} = \frac{1}{5}$ for $i \in \{1, 2, 3, 4, 5\}$, which can be prevented neither by $x(C) \leq 4 + l_5$ nor by $x(C^2) \leq 4 + l_5$.

To solve this problem, we aim at combining several constraints of type (2) in one constraint of type (5). We will give two examples of this strategy. First, consider $I = \{(i, j) \in P^2 \mid i \neq j\}$. Then we have $k \in K(P, I)$ if and only if $k \geq 2$ and $k \mid p$, hence

$$\sum_{i,j \in P,\ i \neq j} x_{ij} \leq p - 1 + \sum_{k \geq 2,\ k \mid p} l_k$$

is a valid inequality. Second, let $e, f \in \{1, \ldots, p\}$ with $e < f$ and $\gcd(f - e, p) = 1$. For $I = \{(i, j) \in P^2 \mid C^e(i) = j \text{ or } C^f(i) = j\}$, i.e., for the combination of C^e and C^f, one can show $K(P, I) = \{p/\gcd(e, p), p/\gcd(f, p)\}$. In particular, if p is odd and $p \neq 1$, we get $x(C) + x(C^2) \leq p - 1 + l_p$ as a valid inequality.

In general, it is difficult – even for very restricted classes of graphs (P, I) – to develop an algorithm for computing $K(P, I)$, let alone a simple formula as above. We are convinced that the algorithm can be improved at this point by finding other classes of graphs that can be treated efficiently.

We separate all tractable types of decomposition inequalities by the same straightforward heuristical scheme. We first compute a permutation π of V in the following way: At the beginning, the nodes $\pi(i)$ and $\pi^{-1}(i)$ are undefined for all $i \in V$. Then we traverse all pairs $(i, j) \in V^2$ by descending LP-value of x_{ij}. For the pair (i, j), we set $\pi(i) = j$ and $\pi^{-1}(j) = i$ if $\pi(i)$ and $\pi^{-1}(j)$ are undefined. This yields the permutation π. Now for every π-orbit $C = (i_1, \ldots, i_p)$, we check if the constraint (2) or some improved constraint of type (5) is violated.

5.4 Primal Heuristics

Finding symmetries heuristically is difficult, since a local variation of the graph usually changes the set of its symmetries completely, so that straightforward greedy heuristics do not work. The following lemma helps (see Sect. 5.2 for the definition of the graph G_c):

Lemma 3. *Each maximum clique in G_c has n nodes. There is a one-to-one correspondence between the set of maximum cliques in G_c and $Aut(G)$.*

We skip the simple proof; we just mention that the automorphism corresponding to a maximum clique $Q \subseteq V^2$ is given by $\pi(i) = j$ for all $(i, j) \in Q$.

Lemma 3 tells us that we can use a heuristic for the maximum clique problem in G_c and hope that the computed clique has n nodes so that it corresponds to an automorphism of G. If we are lucky, this automorphism is even a symmetry.

Using this algorithm as a stand-alone heuristic is not very effective, since in most cases, the computed clique will either not be maximum or correspond to the trivial automorphism of G. In the branch & cut-setting, we apply the primal heuristics whenever the LP-solver comes up with a fractional solution $\overline{x}_{ij}$, hoping that this solution will guide us to a nearby integer solution. A good way to use this knowledge in our maximum clique heuristic is to assign the weight $\overline{x}_{ij}$ to the node $(i, j) \in G_c$ (as we did in Sect. 5.2) and to search for a maximum *weight* clique in G_c.

We also use another primal heuristic: We consider the permutation π of V computed for the separation of decomposition inequalities described in Sect. 5.3, then we check if this permutation induces a symmetry of G. This heuristic is very fast; the additional runtime for checking is linear.

6 Experimental Runtimes

The branch & cut-algorithm presented in the previous section has been implemented in C++ using ABACUS [5] with CPLEX. As mentioned before, the problem being solved is NP-hard, so that no polynomial time bound can be given. Instead, we present experimental runtimes in this section. All instances have been solved on a Sun UltraSPARC-II (296 MHz) machine. We are still working on improving the algorithm at several points and presume that the runtimes will decrease in the future.

Since we do not have test instances from applications, we have to create random instances. This is delicate: Using random graphs is not useful, since symmetries are rare, most graphs do not have any non-trivial automorphism at all. These graphs are usually very easy to handle. But if the user of our algorithm would not expect his graphs to have symmetries with a reasonable probability, he probably would not use it.

Given a fixed number n of nodes, we therefore create graphs with symmetries of any feasible order, i.e., of any order $k \in \{1, \ldots, n\}$ such that $k \mid n$ or $k \mid (n-1)$. To create a symmetry of a given order k, we first choose a number f of fixed nodes, where $f \leq 1$ for $k \geq 3$ and $k \mid (n - f)$. Next, we randomly choose a permutation of $V = \{1, \ldots, n\}$ with f fixed elements and all other orbits of length k. This permutation induces a permutation π of the node-pairs in V^2. For each π-orbit $I \subseteq V^2$, we add an edge between i and j either for all $(i, j) \in I$ or for none. The resulting simple graph has a symmetry of order k by construction, but observe that it may also have a better symmetry by chance. For $k = 1$, the resulting graph is just a random graph with a random number of edges between zero and $n(n - 1)/2$.

The influence of order on runtime is shown in Table 1. Here, we created 10 000 graphs with 50 nodes each by the method explained above and sorted the results by the order of the computed symmetry. We display average and maximum cpu-time (in seconds), average and maximum number of generated subproblems, and average and maximum number of solved LPs. Observe that runtime increases with order, at least for high orders. In Table 2, we display the results for $n \leq 80$, where for each n we created 1 000 test instances by the method explained above, with equally many symmetries of each feasible order.

Table 1. Runtimes for $n = 50$, sorted by order

	Time (sec.)		#Subs		#LPs	
Order	avg	max	avg	max	avg	max
1	1.74	1.95	1.00	1	1.00	1
2	2.12	2.82	1.00	1	1.00	1
5	2.10	2.48	1.00	1	1.00	1
7	2.09	2.46	1.00	1	1.00	1
10	2.19	5.85	1.00	1	2.40	21
25	2.44	7.72	1.00	1	1.25	12
49	3.93	27.02	1.00	1	2.23	15
50	8.02	83.27	1.00	1	4.86	34

Table 2. Runtimes for $1 \leq n \leq 80$

	Time (sec.)		#Subs		#LPs	
n	avg	max	avg	max	avg	max
1–10	0.05	30.43	1.51	197	2.52	328
11–20	1.48	2756.89	2.00	707	4.49	1096
21–30	4.00	6976.54	1.09	139	2.49	244
31–40	1.38	720.96	1.00	1	2.03	78
41–50	2.47	143.86	1.00	3	2.02	51
51–60	4.36	147.17	1.00	1	2.00	42
61–70	7.00	208.94	1.00	1	1.94	39
71–80	10.32	763.62	1.00	1	2.01	63

These tables show that on average the branch & cut-algorithm creates very few subproblems. Even the number of LPs to be solved is very small in general. Furthermore, the average number of subproblems and LPs surprisingly decreases for larger graphs. This is due to the following fact: The larger the graph is, the lower is the probability to get some “extra structure” by chance, i.e., some automorphism or similar structure different from the symmetry created explicitly. This extra structure may distract our algorithm. Particularly hard to handle are graphs with many automorphisms but few symmetries. For this reason, we also created graphs with not necessarily geometric automorphisms. For a given number n of vertices, we computed a random permutation of $V = \{1, \ldots, n\}$ and added the edges exactly as in the last step of the symmetry creation. For these instances, the results are significantly worse, see Table 3. We checked 100 instances for each $n \leq 30$.

Table 3. Runtimes for hard instances

	Time (sec.)		#Subs		#LPs	
n	avg	max	avg	max	avg	max
1–5	0.01	0.08	1.23	7	1.55	17
6–10	0.06	1.19	2.01	41	3.75	52
11–15	0.83	193.67	5.40	909	10.51	1172
16–20	4.08	991.59	4.02	433	12.24	1157
21–25	25.99	3533.09	10.60	2523	32.08	7453
26–30	99.42	9197.55	7.68	1295	24.37	2147

This shows that the most important improvement to be achieved is a better rejection of non-geometric automorphisms, for example by better decomposition inequalities, see Sect. 5.3. If we search for arbitrary automorphisms instead of symmetries, the runtimes are much shorter for the same instances.

Finally, we applied the symmetry detection algorithm to all 11 523 graphs of the "Rome Library" [9]. Here, the number of nodes ranges from 10 to 100. We observed an average runtime of 4.26 seconds, the average number of subproblems and LPs was 1.06 and 1.11, respectively. The maximum runtime was 19.29 seconds (for `grafo11203.100` with 100 nodes); the algorithm needed at most three subproblems and five LPs. Nine graphs have a 3-rotation and 7 845 of the remaining graphs have a reflection, with many fixed nodes in general.

7 Extensions

The algorithm presented in this article can be applied to more general types of graphs. In the most general case, we have a colored graph, i.e., a set V of nodes and a coloring $c\colon V^2 \to \mathbf{N}$. Then an automorphism of $G = (V, c)$ is defined as a permutation π of V such that $c(i, j) = c(\pi(i), \pi(j))$ for all $i, j \in V$. By this, we can also model multigraphs (by defining $c(i, j)$ as the number of edges between the nodes i and j) and directed graphs (the coloring c does not have to be symmetric). The number of loops at node i can be stored in $c(i, i)$. Now define the adjacency matrix of G by $A_G = (c(i, j))_{i,j}$, and observe that labelings can be defined and computed for general colored graphs as well as for simple graphs (the coloring c itself is a labeling of G). Then our algorithm carries over without modification.

Another extension concerns Lemma 2 in Sect. 2. After computing the best symmetry π_1 as explained above, we can compute a second symmetry π_2 that can be displayed simultaneously with π_1 without being generated by π_1 – if such a symmetry exists – by the same procedure, except for a slight modification of the ILP (3). Using Lemma 2, one can show that it is necessary and sufficient to enforce $\pi_2^2 = \mathrm{id}_V$, $\pi_2\pi_1 = \pi_1^{-1}\pi_2$, and, if $\mathrm{ord}(\pi_1)$ is even, that $\pi_2 \neq \pi_1^e$ for $e = \mathrm{ord}(\pi_1)/2$. These conditions translate to

$$\begin{aligned} x_{ij} &= x_{ji} && \text{for all } i,j \in V \\ x_{\pi_1(i),j} &= x_{i,\pi_1(j)} && \text{for all } i,j \in V \\ \textstyle\sum_{i\in V} x_{i,\pi_1^e(i)} &\leq n-2 && \text{if } e = \mathrm{ord}(\pi_1)/2 \text{ is integer.} \end{aligned}$$

Finally, we can modify ILP (3) to solve other problems related to symmetry and automorphism detection, such as graph isomorphism or the greatest common subgraph problem. The necessary adjustments of (3) are easy to derive, but to get a fast algorithm, it is indispensable to find new and special cutting planes for each of these problems.

8 Conclusion and Outlook

This presentation should be considered as a first step towards a polyhedral approach to the symmetry detection problem in arbitrary graphs. At the current state, most graphs can be processed fast, but in some cases, the runtimes are exorbitant. We are convinced that the algorithm can be improved sharply, especially by finding new types of cutting planes. For this, deeper insight into the structure of the polytopes describing automorphisms or symmetries is needed.

References

1. O. Bastert. New ideas for canonically computing graph algebras. Technical Report TUM-M9803, Technische Universität München, Fakultät für Mathematik, 1998.
2. P. Eades and X. Lin. Spring algorithms and symmetry. *Theoretical Computer Science*, 240(2):379–405, 2000.
3. H. de Fraysseix. An heuristic for graph symmetry detection. In J. Kratochvíl, editor, *Graph Drawing '99*, volume 1731 of *Lecture Notes in Computer Science*, pages 276–285. Springer Verlag, 1999.
4. S.-H. Hong, P. Eades, and S.-H. Lee. Finding planar geometric automorphisms in planar graphs. In K.-Y. Chwa et al., editors, *Algorithms and computation. 9th international symposium, ISAAC '98*, volume 1533 of *Lecture Notes in Computer Science*, pages 277–286. Springer Verlag, 1998.
5. M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price-algorithms in integer programming and combinatorial optimization. *Software – Practice & Experience*, 30(11):1325–1352, 2000.
6. J. Manning. Computational complexity of geometric symmetry detection in graphs. In *Great Lakes Computer Science Conference*, volume 507 of *Lecture Notes in Computer Science*, pages 1–7. Springer Verlag, 1990.
7. H. Purchase. Which aesthetic has the greatest effect on human understanding? In Giuseppe Di Battista, editor, *Graph Drawing '97*, volume 1353 of *Lecture Notes in Computer Science*, pages 248–261. Springer Verlag, 1997.
8. R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.
9. The Rome library of undirected graphs. Available at: `www.inf.uniroma3.it/people/gdb/wp12/undirected-1.tar.gz`.