Alcom-FT Technical Report Series
ALCOMFT-TR-03-96

An Integer Programming Approach to Fuzzy Symmetry Detection ⋆

Christoph Buchheim and Michael Jünger

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

Abstract. The problem of exact symmetry detection in general graphs has received much attention recently. In spite of its NP-hardness, two different algorithms have been presented that in general can solve this problem quickly in practice [5, 2]. However, as most graphs do not admit any exact symmetry at all, the much harder problem of fuzzy symmetry detection arises: a minimal number of certain modifications of the graph should be allowed in order to make it symmetric. We present a general approach to this problem: we allow arbitrary edge deletions and edge creations; every single modification can be given an individual weight. We apply integer programming techniques to solve this problem exactly or heuristically and give runtime results for a first implementation.

1 Introduction

An important aim in Automatic Graph Drawing is the display of symmetric structure inherent in a graph. Empirical studies show that symmetric drawings of a graph are much easier to understand than asymmetric ones [14]. However, two different problems occur when trying to develop algorithms for symmetric graph drawing: first, the symmetry detection problem is NP-hard for general graphs [13]. Second, most graphs do not admit any exact symmetry at all; in order to create nearly symmetric drawings, one has to define some kind of relaxation of exact symmetry.

For the first problem, two different algorithms with exponential runtime in general but fast runtime in practice have been proposed recently. In [4], we presented an approach based on integer programming techniques; see [5] and [3] also. Later, Abelson et al. [2] devised an algorithm based on group-theoretic methods, running even faster.

The second problem is motivated by the observation that nearly symmetric drawings still help the user to understand the structure of a graph; see Fig. 1 for some examples. Unfortunately, mostly negative results about this problem have been published so far. In fact, detecting fuzzy symmetries is much harder than

⋆ Partially supported by the Future and Emerging Technologies Programme of the European Union under contract number IST-1999-14186 (ALCOM-FT).

detecting exact symmetries in general, as shown by Chen et al. [6]. In their approach, different types of modifications of the graph are allowed in order to make it symmetric, namely, to delete nodes, to delete edges, and to contract edges. They consider the problem of finding a minimum number of such operations needed to obtain a graph admitting an axial or rotational symmetry. Among other results, they show that some of the corresponding problems are NP-hard even for trees. On the other hand, exact planar symmetry detection is possible in linear time for planar graphs [9–12].

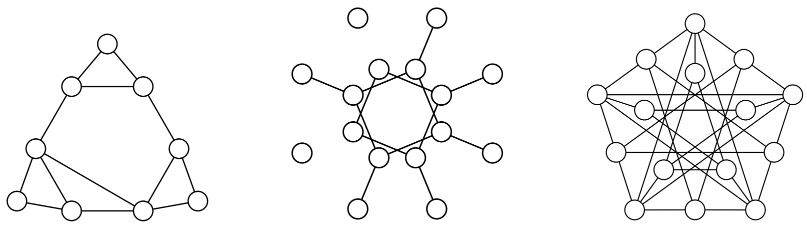

Fig. 1. Some nearly symmetric drawings

In this paper, we follow a similar approach: we allow a minimum number of edge deletions and edge creations in order to obtain a graph that admits a symmetric drawing. A more precise description of this problem, along with a generalization, is given in Sect. 3. Before, we gather some definitions concerning symmetries in Sect. 2. In Sect. 4, we present integer linear programs (ILPs) modeling the fuzzy rotation and fuzzy reflection problems. The corresponding polytopes are investigated in Sect. 5. Finally, we roughly explain the framework of a branch & cut-algorithm for fuzzy symmetry detection in Sect. 6. There we also report runtime results for a first implementation of this algorithm. For more details or any proof, the reader is referred to [3].

2 Preliminaries

For ease of exposition, we only consider simple undirected graphs in the following. An *automorphism* of a graph $G = (V, E)$ 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 an automorphism π is $\mathrm{ord}(\pi) = \min\{k \in \mathbb{N} \mid \pi^k = \mathrm{id}_V\}$, where id_V denotes the identity permutation of V. For a node $i \in V$, the set $\mathrm{orb}_\pi(i) = \{\pi^k(i) \mid k \in \mathbb{N}\}$ 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 one or two. 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 $k \neq 1$. Each 2-rotation is a reflection, but not vice versa; the identity id_V is both a reflection and a 1-rotation.

Now assume that there exist an injective node placement $p: V \to \mathbb{R}^2$ and an isometry $\varphi: \mathbb{R}^2 \to \mathbb{R}^2$ of the Euclidean plane with the following properties: for each $i \in V$ there exists a node $i' \in V$ with $\varphi(p(i)) = p(i')$, and for $i, j \in V$ we have $(i, j) \in E$ if and only if $(i', j') \in E$; for straight-line drawings of simple graphs, this means nodes are mapped to nodes and edges are mapped to edges. Then p and φ induce an automorphism π of G by setting $\pi(i) = i'$. Any automorphism induced like this is called a *geometric automorphism* or a *symmetry* of G. An automorphism of a graph G is a symmetry if and only if it is a rotation or a reflection [8]. Obviously, the symmetries displayed by a single placement p form a subgroup of $\mathrm{Aut}(G)$, but the set of all symmetries of G is not closed under composition. Observe that this definition of a symmetry, following the one given in [8], allows degenerate drawings: nodes may be located on non-incident edges and edges may overlap. In fact, in the case of a reflection, the subgraph induced by the fixed nodes may be arbitrarily large, so that degeneracies may be unavoidable. However, all nodes must have different positions.

3 The Fuzzy Symmetry Detection Problem

The general idea of our approach to fuzzy symmetry detection is to allow to change the adjacency of every pair of nodes in G. More precisely, we allow to create an edge between each pair of non-adjacent nodes and to delete the edge between each pair of adjacent nodes. Creating or deleting an edge between two nodes i and j of G involves a penalty of $w_{ij} \in \mathbb{R}$. The aim is to find a graph $G' = (V, E')$ such that G' is symmetric and such that the total penalty for transforming G into G' is minimal. In other words, let $E \triangle E' = (E \setminus E') \cup (E' \setminus E)$ denote the symmetric difference between E and E'. Then our objective is to minimize the total penalty

$$\sum_{(i,j) \in E \triangle E'} w_{ij}$$

over all graphs $G' = (V, E')$ that admit any non-trivial symmetry. We call this problem the *weighted fuzzy symmetry detection* problem.

Observe that we do not require positive weights. However, allowing negative weights does not increase the power of the approach, since all negative weights can be replaced by their absolute values after changing the adjacency of the corresponding edges in G. The presence of zero weights means that the adjacency of the corresponding node pairs does not matter at all. By using large enough weights, we can also forbid certain modifications absolutely.

The weighted fuzzy symmetry detection problem is NP-hard, even if all weights are one. This follows from the NP-hardness of the exact symmetry detection problem. In practice, fuzzy symmetry detection is much harder than exact symmetry detection—at least for our solution methods.

Finally notice that we could also allow node deletions in our approach. For simplicity, however, we restrict ourselves to edge modifications here.

4 An Integer Linear Programming Model

In order to describe fuzzy symmetries by integer linear programs, we split up the problem in the following way: we fix an integer $k \in \{2, \ldots, n\}$ and consider the problem of finding a graph G' that admits a symmetry of order k and that differs minimally from G. By applying this approach to all feasible orders one after another, we can solve the original problem.

Hence we set up a separate *fuzzy rotation ILP* for each $k \geq 3$ such that k divides either n or $n-1$ and a *fuzzy reflection ILP* for $k = 2$. All these programs are based on the *fuzzy automorphism ILP* that uses two different types of variables to model fuzzy automorphisms.

The *mapping variables* are used to specify a permutation π of V: we define a real $n \times n$-matrix $M(\pi)$ by

$$M(\pi)_{ij} = \begin{cases} 1 & \text{if } \pi(i) = j \\ 0 & \text{otherwise,} \end{cases}$$

yielding a monomorphism M of the group S_n of permutations of V into the general linear group $\mathrm{GL}_n(\mathbb{R})$. The matrices in $M(S_n)$ are called *permutation matrices* and can be characterized as the set of $n \times n$-matrices $X = (x_{ij})$ with $x_{ij} \in \{0, 1\}$ and

$$\sum_{j \in V} x_{ij} = 1 \text{ for all } i \in V \quad \text{and} \quad \sum_{i \in V} x_{ij} = 1 \text{ for all } j \in V \; . \tag{1}$$

In other words, for the desired ILP, we can use a binary mapping variable x_{ij} for each pair $(i, j) \in V^2$ and add the constraints (1). A value of one for the mapping variable x_{ij} is interpreted as mapping node i to node j.

For technical reasons, we replace the permutation constraints (1) by the weaker constraints

$$\sum_{j \in V} x_{ij} \leq 1 \text{ for all } i \in V \quad \text{and} \quad \sum_{i \in V} x_{ij} \leq 1 \text{ for all } j \in V \; . \tag{2}$$

Thus the permutation corresponding to the mapping variables may be partially or totally undefined. However, by adding a large value to all objective function coefficients of mapping variables, every optimal solution will still correspond to a well-defined rotation or reflection.

The second type of variables is used to model all graphs G' with node set V. We use a binary *edge variable* y_{ij} for each pair of nodes $i, j \in V$ that is set to one if and only if the nodes i and j are adjacent in G'.

The connection between the two types of variables is established as follows: we want the permutation π of V given by the mapping variables to be an automorphism of the graph G' given by the edge variables. In other words, if two nodes i and j are adjacent in G', then the same must be true for $\pi(i)$ and $\pi(j)$, and vice versa. This can be translated into

$$x_{i_1 j_1} + x_{i_2 j_2} \leq 2 \pm y_{i_1 i_2} \mp y_{j_1 j_2} \text{ for all } i_1, i_2, j_1, j_2 \in V \; . \tag{3}$$

Indeed, these constraints are trivial if the left hand side is at most one. If the left hand side is two, then $\pi(i_1) = j_1$ and $\pi(i_2) = j_2$. In this case, the right hand sides make sure that $y_{i_1 i_2} = y_{j_1 j_2}$, i.e., either both pairs (i_1, i_2) and (j_1, j_2) are adjacent or both are non-adjacent in G'.

In summary, fuzzy automorphisms correspond to the solutions of

$$\begin{array}{rll} x_{ij} \in \{0,1\} & \text{for all } i,j \in V \\ y_{ij} = y_{ji} \in \{0,1\} & \text{for all } i,j \in V,\ i \neq j \\ \sum_{j\in V} x_{ij} \leq 1 & \text{for all } i \in V \\ \sum_{i\in V} x_{ij} \leq 1 & \text{for all } j \in V \\ x_{i_1 j_1} + x_{i_2 j_2} \leq 2 \pm y_{i_1 i_2} \mp y_{j_1 j_2} & \text{for all } i_1, i_2, j_1, j_2 \in V,\ i_1 \neq i_2,\ j_1 \neq j_2\ . \end{array} \tag{4}$$

Next, we are going to restrict the ILP (4) to rotations of order $k \geq 3$. The first condition for π to be a k-rotation is $|\mathrm{orb}_\pi(i)| \in \{1, k\}$ for all $i \in V$. All other orbit lengths are forbidden. In order to set up linear constraints to enforce this property of π, we define a *forbidden orbit* of G as a circular list $C = (i_1, \ldots, i_p)$ of pairwise distinct nodes in V with $k \neq p \geq 2$. Let $|C| = p$ and

$$x(C) = \sum_{t=1}^{p-1} x_{i_t i_{t+1}} + x_{i_p i_1}\ .$$

Then the condition $|\mathrm{orb}_\pi(i)| \in \{1, k\}$ for all $i \in V$ is equivalent to

$$x(C) \leq |C| - 1 \ \text{ for all forbidden orbits } C\ . \tag{5}$$

Indeed, the constraint (5) implies that for every forbidden orbit C at least one variable in $x(C)$ is zero, so that C is not an orbit of π. To this point, we have ensured that all orbits of π have length k or 1. Any k-rotation has to meet the additional condition $|\mathrm{Fix}(\pi)| \leq 1$. We can express this by

$$\sum_{i\in V} x_{ii} \leq 1\ , \tag{6}$$

as the sum on the left hand side equals the number of fixed nodes. Combining the constraints (4), (5), and (6), an ILP describing fuzzy k-rotations of G is

$$\begin{array}{rll} x_{ij} \in \{0,1\} & \text{for all } i,j \in V \\ y_{ij} = y_{ji} \in \{0,1\} & \text{for all } i,j \in V,\ i \neq j \\ \sum_{j\in V} x_{ij} \leq 1 & \text{for all } i \in V \\ \sum_{i\in V} x_{ij} \leq 1 & \text{for all } j \in V \\ x_{i_1 j_1} + x_{i_2 j_2} \leq 2 \pm y_{i_1 i_2} \mp y_{j_1 j_2} & \text{for all } i_1, i_2, j_1, j_2 \in V,\ i_1 \neq i_2,\ j_1 \neq j_2 \\ x(C) \leq |C| - 1 & \text{for all forbidden orbits } C \\ \sum_{i\in V} x_{ii} \leq 1\ . \end{array} \tag{7}$$

Fuzzy reflections of G are easier to model than k-rotations: instead of using the forbidden orbit constraints, we just have to ensure that the automorphism π to be represented satisfies $\pi^2 = \mathrm{id}_V$, i.e., that in our model we have

$$x_{ij} = x_{ji} \ \text{ for all } i,j \in V\ . \tag{8}$$

In other words, we only need a single mapping variable for each pair of nodes. Hence an ILP modeling fuzzy reflections is given by

$$\begin{array}{rll} x_{ij} = x_{ji} \in \{0,1\} & \text{for all } i,j \in V & \\ y_{ij} = y_{ji} \in \{0,1\} & \text{for all } i,j \in V,\ i \neq j & \\ \sum_{j \in V} x_{ij} \leq 1 & \text{for all } i \in V & (9) \\ \sum_{i \in V} x_{ij} \leq 1 & \text{for all } j \in V & \\ x_{i_1 j_1} + x_{i_2 j_2} \leq 2 \pm y_{i_1 i_2} \mp y_{j_1 j_2} & \text{for all } i_1, i_2, j_1, j_2 \in V,\ i_1 \neq i_2,\ j_1 \neq j_2 \ . & \end{array}$$

Observe that both (7) and (9) do not depend on the structure of G; they are determined by the number of nodes of G. The information about edges of G is stored in the objective function: we minimize the total weight of the modifications that are necessary to get from G to G', i.e., we minimize

$$\sum_{(i,j) \in V^2 \setminus E} w_{ij} y_{ij} + \sum_{(i,j) \in E} w_{ij}(1 - y_{ij}) \ .$$

Independently, we can assign arbitrary weights to the mapping variables. In the case $k = 2$, fixed nodes should be punished, since otherwise the graph G together with id_V forms an optimal solution of (9). However, the penalty for fixed nodes has to be attuned to the penalties for edge deletion or creation in this case.

5 The Fuzzy Symmetry Polytopes

The ILPs (7) and (9) only depend on the number n of nodes and the desired order k, but not on the structure of G. Thus we may define the *fuzzy symmetry polytope* $\mathrm{FSP}(k,n)$ to be the polytope corresponding to (7), if $k \geq 3$, or the polytope corresponding to (9), if $k = 2$. More precisely, these polytopes are given as the convex hulls of the sets of solutions of these ILPs.

Theorem 1. *The polytope FSP(k,n) is full-dimensional.*

For designing the branch & cut-algorithm for fuzzy symmetry detection, it is important to investigate the polyhedral structure of $\mathrm{FSP}(k,n)$. In particular, we have to find valid inequalities that induce facets of $\mathrm{FSP}(k,n)$, i.e., maximal faces of this polytope. Before starting to present classes of such inequalities, observe

Lemma 1. *If a constraint H is valid or facet-inducing for FSP(k,n), then the same is true after replacing each variable y_{ij} in H by $1 - y_{ij}$.*

Lemma 2. *If a constraint H is valid or facet-inducing for FSP(k,n), then the same is true after replacing each variable x_{ij} in H by x_{ji}.*

Both results follow from symmetry immediately. In the remainder of this section, we will usually consider only one of the four constraints without explicitly mentioning the other three.

Theorem 2. *The trivial constraints $x_{ij} \geq 0$, $y_{ij} \geq 0$, and $y_{ij} \leq 1$ induce facets of FSP(k,n). The same is true for the weak permutation constraints (2).*

To obtain further results about the polytope $\mathrm{FSP}(k,n)$, we will treat rotations and reflections separately in the following subsections.

5.1 The Fuzzy Rotation Polytopes

Throughout this subsection we assume $k \geq 3$. In this case, we have

Theorem 3. *The constraint (6) induces a facet of FSP(k,n).*

We continue with a review of the forbidden cycle constraints (5). For this, let $C = (i_1, \ldots, i_p)$ be a circular list of pairwise distinct nodes with $k \neq p \geq 2$ again. If k does not divide p, we can improve (5) to

$$\sum_{i,j \in C,\ i \neq j} x_{ij} \leq p - 1\ . \tag{10}$$

Indeed, the sum on the left hand side is at most p by (2). If it equals p for some permutation π, then C is fixed under π, but no single node of C is fixed. Since k does not divide p, some non-trivial orbit of π must have a length not equal to k, so π is no k-rotation.

By the same reasoning, if k does not divide $p-1$ either, we even have the valid inequality

$$\sum_{i,j \in C} x_{ij} \leq p - 1\ . \tag{11}$$

Observe that for $k = n$ the constraint (11) is the subtour elimination constraint for the cut defined by C.

Theorem 4. *Let $2 \leq p \leq n-1$ and assume that k does not divide p. If k divides $p-1$, then (10) induces a facet of FSP(k,n). Otherwise, (11) induces a facet of FSP(k,n).*

Next, let W_1 and W_2 be any two subsets of V. Let $i_1, i_2 \in V \setminus (W_1 \cup W_2)$ with $i_1 \neq i_2$. Then we have the following valid inequality for FSP(k,n):

$$\sum_{j_1 \in W_1} x_{i_1 j_1} + \sum_{j_2 \in W_2} x_{i_2 j_2} \leq 2 - y_{i_1 i_2} + \sum_{j_1 \in W_1,\ j_2 \in W_2} y_{j_1 j_2}\ . \tag{12}$$

Indeed, the left hand side is at most two by (2), while the right hand side is at least one. If the left hand side equals two, we have $x_{i_1 j_1} = 1$ for some $j_1 \in W_1$ and $x_{i_2 j_2} = 1$ for some $j_2 \in W_2$. Thus $y_{i_1 i_2} = 1$ implies $y_{j_1 j_2} = 1$, so that the right hand side is at least two.

Theorem 5. *The constraint (12) induces a facet of FSP(k,n) if and only if*

(a) $|W_1| \neq 0$ *and* $|W_2| \neq 0$,
(b) $|W_2 \setminus W_1| \neq 1$ *and* $|W_1 \setminus W_2| \neq 1$, *and*
(c) $|W_1 \cup W_2| \geq 2$.

In the special case $W_1 = W_2 = \{j_1, j_2\}$, the constraint (12) and its counterpart according to Lemma 1 emerge as

$$x_{i_1 j_1} + x_{i_1 j_2} + x_{i_2 j_1} + x_{i_2 j_2} \leq 2 \pm y_{i_1 i_2} \mp y_{j_1 j_2}.$$

These constraints improve (3); both induce facets of FSP(k,n) by Theorem 5.

Finally, let $W \subseteq V$ and $w \in V \setminus W$. Let $i_1, i_2 \in V \setminus (W \cup \{w\})$ with $i_1 \neq i_2$. Then

$$\sum_{j \in W} (x_{i_1 j} + x_{i_2 j}) + 2x_{i_1 w} + 2x_{i_2 w} \leq 3 - y_{i_1 i_2} + \sum_{j \in W} y_{jw} \tag{13}$$

is a valid inequality for $\mathrm{FSP}(k,n)$. Indeed, the left hand side is at most three by (2), while the right hand side is at least two. If the left hand side is equal to three, one of the nodes i_1 or i_2 is mapped to w, while the other one is mapped to some $j \in W$. Hence $y_{i_1 i_2} = y_{jw}$, so that the right hand side is at least three.

Theorem 6. *The constraint (13) induces a facet of FSP(k,n) if and only if $|W| \geq 2$.*

5.2 The Fuzzy Reflection Polytope

Next, we examine the polytope $\mathrm{FSP}(2,n)$ more closely. It is easy to see that for all odd subsets C of V the *blossom constraint*

$$\sum_{i,j \in C,\ i \neq j} x_{ij} \leq |C| - 1 \tag{14}$$

is valid for the fuzzy reflection polytope. We have

Theorem 7. *Let C be an odd subset of V with $|C| \geq 3$. Then the blossom constraint (14) induces a facet of FSP$(2,n)$.*

The constraint (12) is valid for fuzzy reflections as well. However, we can improve it by adding $x_{i_1 i_2}$ to the left hand side: if $x_{i_1 i_2} = 1$, all other variables on the left hand side must be zero. Hence we get

$$x_{i_1 i_2} + \sum_{j_1 \in W_1} x_{i_1 j_1} + \sum_{j_2 \in W_2} x_{i_2 j_2} \leq 2 - y_{i_1 i_2} + \sum_{j_1 \in W_1,\ j_2 \in W_2} y_{j_1 j_2}\ . \tag{15}$$

Theorem 8. *The constraint (15) induces a facet of FSP$(2,n)$ if and only if*

(a) $|W_1| \neq 0$ and $|W_2| \neq 0$,
(b) $|W_2 \setminus W_1| \neq 1$ and $|W_1 \setminus W_2| \neq 1$, and
(c) $|W_1 \cup W_2| \geq 2$.

By Theorem 8, the constraint (3) is improved by the facet-inducing constraints

$$x_{i_1 i_2} + x_{i_1 j_1} + x_{i_1 j_2} + x_{i_2 j_1} + x_{i_2 j_2} \leq 2 \pm y_{i_1 i_2} \mp y_{j_1 j_2}\ .$$

Next, observe that (13) is true for fuzzy reflections as well. Again, we can improve this constraint to

$$2x_{i_1 i_2} + \sum_{j \in W} (x_{i_1 j} + x_{i_2 j}) + 2x_{i_1 w} + 2x_{i_2 w} \leq 3 - y_{i_1 i_2} + \sum_{j \in W} y_{jw}\ . \tag{16}$$

Theorem 9. *The constraint (16) induces a facet of FSP$(2,n)$ if and only if $|W| \geq 2$.*

6 The Branch & Cut-Algorithm

In the following, we roughly describe our branch & cut-algorithm based on the results of the previous sections. We restrict ourselves to separation and primal heuristics; for other topics including the branching strategy and rules for setting variables, see [3]. For branch & cut in general, see [4] or [3].

6.1 Separation

We start the branch & cut-algorithm with the relaxation of (7) or (9) composed of all variables, the constraints (2), and, if $k \geq 3$, the constraint (6). The separation problem consists of finding inequalities that are valid for (7) or (9) but violated by the optimal solution of the current LP-relaxation. If we can find such inequalities, we add them to the relaxation and repeat the process.

We first try to separate constraints of type (12), if $k \geq 3$, or (15), if $k = 2$. For this, we use a straightforward greedy heuristic: for fixed i_1 and i_2, we add nodes to W_1 or W_2 as long as the difference between the left and the right hand side of (12) or (15) can be increased (for the current LP-solution). If we end up with any violated constraints, these are added to the LP-relaxation.

Otherwise, we start separating constraints of type (13), if $k \geq 3$, or (16), if $k = 2$. It is easy to see that these constraints can be separated in polynomial time: for a fixed combination of i_1, i_2, and w, we add a node j to W if and only if the current LP-values of $x_{i_1 j}$ and $x_{i_2 j}$ add up to more than the one of y_{jw}.

If we did not find any violated constraints of type (13) or (16), we separate the different kinds of orbit constraints: for $k \geq 3$, we search for violated constraints of type (10) or (11), which can be done in polynomial time by submodular function minimization. The same is true for the blossom constraints (14) for $k = 2$. If we still did not find any violated constraints, we start branching.

6.2 Primal Heuristics

Primal heuristics exploit the optimal solution of the current LP-relaxation in order to find feasible (but not necessarily optimal) solutions of the original ILP. In our algorithm, we use two heuristics working in an opposite way.

The first one starts with determining a symmetry of the given order k of the complete graph K_n that is close to the current (possibly fractional) values of the mapping variables. For example, we can traverse the mapping variables x_{ij} in descending order according to their value and let i be mapped to j if and only if no other node has been mapped to j before and i has not been mapped to any other node before. Once having fixed the node permutation, we can easily determine optimal values for all edge variables.

The second heuristic first fixes the edges of G' by rounding the values of all edge variables. Then the mapping variables x_{ij} are again traversed in descending order according to their value; the node i is mapped to j if and only if this contradicts neither earlier mapping decisions nor the fixing of edges in G'. Even if this strategy only yields a partially defined permutation of V, we can still use its value as a bound on the optimal solution of the ILP.

6.3 Runtime Results

Our branch & cut-algorithm for fuzzy symmetry detection is not fully developed yet. It is necessary and certainly possible to improve it significantly by future work. Nevertheless, we experimented with an implementation based on the polyhedral results obtained so far. In this evaluation, we searched for fuzzy reflections and fuzzy rotations of order n. The penalty for every modification of the graph was set to one. In the reflection case, any fixed node also involved a penalty of one, i.e., one edge modification was allowed to prevent one fixed node.

For all evaluations, we used an AMD 1400 MHz Athlon processor. Runtime results are given in CPU-seconds. Our implementation is based on ABACUS [1] in combination with CPLEX [7]. We used random simple undirected graphs for our experiments. Each pair of different nodes is adjacent by a probability of 1/4. Observe that random graphs are hard instances for fuzzy symmetry detection, since in general a lot of modifications are necessary to make them symmetric. For every number n of nodes, we tested 100 graphs in each evaluation.

Compared with our results for exact symmetry detection presented in [5], runtimes for fuzzy symmetry detection are rather disappointing. The results for reflection detection are displayed in Table 1. We list average and maximal runtime, number of subproblems, and number of LPs. Since we can use our algorithm heuristically by stopping it at any time, we also list the CPU-time needed to find an optimal solution—without knowing its optimality at this point.

Table 1. Results for fuzzy reflection detection

	runtime		opttime		#subprobs		#LPs	
n	avg	max	avg	max	avg	max	avg	max
8	0.04	0.32	0.00	0	4.0	25	13.2	79
9	0.13	1.58	0.01	1	8.8	57	28.9	201
10	0.53	3.15	0.04	2	20.7	95	68.8	317
11	2.01	14.46	0.43	5	48.4	263	160.9	894
12	11.92	75.08	1.86	49	167.5	931	567.8	3185

Table 1 shows that the runtime, the number of subproblems, and the number of LPs increase sharply already for small graphs. However, the time needed to find the optimal solution is much shorter than the total runtime, i.e., we know the optimal solution much earlier than the fact that it is optimal. We conclude that the primal heuristics used in our algorithm work well, whereas the cutting planes and separation algorithms must be improved in order to get a practically useful algorithm. In fact, a lot of violated cutting planes were usually found and added without changing the objective value of the optimal LP-solution.

The results displayed in Table 1 are much more homogeneous than those for exact symmetry detection; see [5]. This is even more evident if we split up the results by the optimal objective function value, i.e., by the minimial number of modifications and fixed nodes; see Table 2 for $n = 12$.

Table 2. Results for fuzzy reflection detection, $n = 12$

	% of	runtime		opttime		#subprobs		#LPs	
obj	insts	avg	max	avg	max	avg	max	avg	max
1	6	0.17	0.46	0.00	0	9.0	21	26.2	62
2	17	1.18	5.81	0.59	5	30.5	97	99.4	333
3	30	4.12	10.85	1.03	8	75.0	155	249.0	489
4	33	13.44	24.32	1.33	15	192.6	311	660.5	1031
5	13	40.72	60.22	7.77	49	511.0	791	1729.1	2526
6	1	75.08	75.08	0.00	0	931.0	931	3185.0	3185

The results displayed in Table 2 reveal a strong connection between the hardness of an instance and its distance from being reflectional symmetric. This is good news, as the fuzzy symmetry detection algorithm is designed to draw nearly symmetric graphs.

Now consider the case $k = n$. As shown in Table 3, average runtimes for fuzzy rotation detection are even longer—by a factor of more than 25 for $n = 12$. For 15 instances, we could not even find a provably optimal solution within one hour of CPU-time. One reason may be the larger number of mapping variables in the fuzzy rotation ILP compared with the fuzzy reflection ILP.

Table 3. Results for fuzzy n-rotation detection

	runtime		opttime		#subprobs		#LPs	
n	avg	max	avg	max	avg	max	avg	max
8	0.10	1.56	0.00	0	12.1	135	28.6	299
9	1.67	45.07	0.03	2	87.8	1239	232.6	3775
10	6.93	133.58	0.46	7	165.7	2251	541.4	8044
11	23.34	414.23	2.25	56	568.0	9867	1707.9	29759
12	305.64	15.0 %	35.60	711	2616.1	16785	8692.9	56342

To investigate the effect of the number of variables more closely, we finally evaluated a variant of the fuzzy reflection detection algorithm: we only allowed to delete edges but not to create them. In the corresponding ILP, we only needed an edge variable for each adjacent node-pair in the original graph. By construction of our test set, we could thus save about 3/4 of the edge variables. Table 4 shows that a smaller number of edge variables can decrease runtime significantly. Compared with the results given in Table 1, we observed an improvement of average runtime by a factor of more than 14 for $n = 12$.

In summary, if used as an exact algorithm, our branch & cut-method only works for small graphs yet. Its performance is much better for nearly symmetric graphs than for random graphs; restricting the set of allowed modifications also helps. We are optimistic that the runtime can be improved significantly by future work. In fact, for exact symmetry detection, we were able to accelerate our branch & cut-algorithm by a factor of up to 50 until now, compared with [4].

Table 4. Results for fuzzy reflection detection, deleting edges only

	runtime		opttime		#subprobs		#LPs	
n	avg	max	avg	max	avg	max	avg	max
8	0.02	0.14	0.00	0	3.1	15	9.0	38
9	0.05	0.24	0.00	0	5.9	29	17.5	59
10	0.10	0.35	0.00	0	8.6	33	24.0	70
11	0.35	1.53	0.05	1	16.7	53	48.7	156
12	0.84	4.39	0.06	2	28.8	111	85.0	327
13	4.68	20.13	1.23	11	85.5	273	260.3	846
14	8.19	26.94	1.22	8	122.7	371	380.1	1112
15	51.86	161.30	11.95	124	405.5	1149	1280.5	3623
16	79.16	219.41	10.05	107	554.9	1419	1791.2	4521

References

1. ABACUS – A Branch-And-CUt System. `www.informatik.uni-koeln.de/abacus`.
2. D. Abelson, S. Hong, and D. Taylor. A group-theoretic method for drawing graphs symmetrically. In M. Goodrich and S. Kobourov, editors, *Graph Drawing 2002*, volume 2528 of *LNCS*, pages 86–97. Springer-Verlag, 2002.
3. C. Buchheim. *An Integer Programming Approach to Exact and Fuzzy Symmetry Detection*. PhD thesis, Institut für Informatik, Universität zu Köln, 2003. Available at `kups.ub.uni-koeln.de/volltexte/2003/918`.
4. C. Buchheim and M. Jünger. Detecting symmetries by branch & cut. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing 2001*, volume 2265 of *Lecture Notes in Computer Science*, pages 178–188. Springer-Verlag, 2002.
5. C. Buchheim and M. Jünger. Detecting symmetries by branch & cut. *Mathematical Programming, Series B*, 98:369–384, 2003.
6. H.-L. Chen, H.-I. Lu, and H.-C. Yen. On maximum symmetric subgraphs. In J. Marks, editor, *Graph Drawing 2000*, volume 1984 of *LNCS*, pages 372–383. Springer-Verlag, 2001.
7. CPLEX 7.0. `www.ilog.com/products/cplex`.
8. P. Eades and X. Lin. Spring algorithms and symmetry. *Theoretical Computer Science*, 240(2):379–405, 2000.
9. S. Hong and P. Eades. Drawing planar graphs symmetrically II: Biconnected graphs. Technical Report CS-IVG-2001-01, University of Sydney, 2001.
10. S. Hong and P. Eades. Drawing planar graphs symmetrically III: Oneconnected graphs. Technical Report CS-IVG-2001-02, University of Sydney, 2001.
11. S. Hong and P. Eades. Drawing planar graphs symmetrically IV: Disconnected graphs. Technical Report CS-IVG-2001-03, University of Sydney, 2001.
12. S. Hong, B. McKay, and P. Eades. Symmetric drawings of triconnected planar graphs. In *SODA 2002*, pages 356–365, 2002.
13. J. Manning. Computational complexity of geometric symmetry detection in graphs. In N. Sherwani, E. de Doncker, and J. Kapenga, editors, *First Great Lakes Computer Science Conference*, volume 507 of *LNCS*, pages 1–7. Springer-Verlag, 1991.
14. H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Graph Drawing '97*, volume 1353 of *LNCS*, pages 248–261. Springer-Verlag, 1998.