Drawing Graphs Symmetrically in Three Dimensions*

Seok-Hee Hong

Basser Department of Computer Science, University of Sydney, Australia.
shhong@cs.usyd.edu.au

Abstract. In this paper, we investigate symmetric graph drawing in three dimensions. We show that the problem of drawing a graph with a maximum number of symmetries in three dimensions is NP-hard. Then we present a polynomial time algorithm for finding maximum number of three dimensional symmetries in planar graphs.

1 Introduction

Symmetry is one of the most important aesthetic criteria for Graph Drawing. It clearly reveals the structure of a graph. Symmetric graph drawing has been investigated by a number of authors [2,4,5,6,7,8,9,10,13,16,17,18]. However, most of previous work on symmetric graph drawing has mainly focused on two dimensions [2,4,5,6,7,8,13,16,17,18].

Symmetry in three dimensions is much richer than that in two dimensions. For example, a maximal symmetric drawing of the icosahedron in two dimensions shows 6 symmetries. However, the maximal symmetric drawing of the icosahedron in three dimensions shows 120 symmetries.

In this paper, we investigate symmetric graph drawing in three dimensions. First, we show that the problem of drawing a graph with a maximum number of symmetries in three dimensions is NP-hard. Then we present a polynomial time algorithm for finding maximum number of three dimensional symmetries in planar graphs.

To draw a graph symmetrically in three dimensions, there are two steps: first find the three dimensional symmetries, then construct a drawing which displays these symmetries. The first step is the more difficult one. This paper concentrates on the first step.

This paper is organized as follows. In the next section, we explain necessary background. We show that the problem of drawing a graph with a maximum number of symmetries in three dimensions is NP-hard in Section 3. In Section 4, Section 5, Section 6, and Section 7, we present an algorithm for finding maximum number of three dimensional symmetries of planar graphs. Section 8 concludes.

* In this extended abstract, many proofs are omitted. This research has been supported by a grant from the Australian Research Council. Three dimensional drawings are available from http://www.cs.usyd.edu.au/~shhong/research7.htm.

2 Symmetric Graph Drawing in Three Dimensions

In this section, we explain the types of three dimensional symmetry and describe our model for drawing graphs symmetrically in three dimensions.

2.1 Symmetries in Three Dimensions

Symmetry in three dimensions is richer and more complex than symmetry in two dimensions. The types of symmetry in three dimensions can be classified as *rotation*, *reflection*, *inversion* and *rotary reflection* [19]. A rotational symmetry in three dimensions is a rotation about an *axis* and a reflectional symmetry in three dimensions is a reflection in a *plane*. Inversion is a reflection in a *point*. Rotary reflection is a composition of a rotation and a reflection.

A *finite rotation group* in three dimensions is one of following three types: a *cyclic group (C_n)*, a *dihedral group (D_n)* and the rotation group of one of the *Platonic solids* [19]. There are only five regular Platonic solids, the *tetrahedron*, the *cube*, the *octahedron*, the *dodecahedron* and the *icosahedron*.

The *full symmetry group* of a finite object in three dimensions is much more complex than the rotation group. The complete list of all possible symmetry groups in three dimensions can be found in [19].

2.2 Symmetric Drawing in Three Dimensions

For the purpose of drawing graphs in three dimensions, we require three *non-degeneracy* conditions: no two vertices are located at the same point, no two edges overlap, and no vertex lies on an edge with which it is not incident.

A symmetry of a three dimensional drawing D of a graph G induces a permutation of the vertices, which is an *automorphism* of the graph; this automorphism is *displayed* by the symmetry. We say that the automorphism is a *three dimensional symmetry* of a graph G.

The symmetry group of a three dimensional graph drawing D of a graph G induces a subgroup of the automorphism group of G. The subgroup P is a *three dimensional symmetry group* if there is a three dimensional drawing D of G which displays every element of P as a symmetry of the drawing.

3 General Graphs

In this section, we show that the problem of drawing a graph with a maximum number of symmetries in three dimensions is NP-hard.

The aim of this paper is to investigate graph drawings in three dimensions that show a maximum number of symmetries. For this purpose, we need to find a three dimensional symmetry group which has maximum size. We now define our main problem.

Three Dimensional Symmetry Group Problem (3DSGP)
Input: A graph G.
Output: A three dimensional symmetry group P of G which has maximum size.

In general, $3DSGP$ is NP-hard; the following subproblem is NP-complete.

Three Dimensional Size 3 Symmetry Group Problem ($3D3SGP$)
Input: A graph G.
Question: Does G have a three dimensional symmetry group of size at least 3?

Theorem 1. *The problem* $3D3SGP$ *is NP-complete, and the problem* $3DSGP$ *is NP-hard.*

To show that $3D3SGP$ is NP-complete, we consider the following problems. A vertex is *fixed* by a given three dimensional symmetry if it is mapped onto itself by that symmetry.

1. $3DROT$: Does G have a drawing D in three dimensions which shows any nontrivial rotational symmetry?
2. $3DROT_0$: Does G have a drawing D in three dimensions which shows any nontrivial rotational symmetry with no fixed vertex?
3. $3DREF$: Does G have a drawing D in three dimensions which shows any reflectional symmetry?
4. $3DREF_0$: Does G have a drawing D in three dimensions which shows any reflectional symmetry with no fixed vertex?
5. $3DINV$: Does G have a drawing D in three dimensions which shows an inversion?
6. $3DINV_0$: Does G have a drawing D in three dimensions which shows an inversion with no fixed vertex?
7. $3DINV_1$: Does G have a drawing D in three dimensions which shows an inversion with one fixed vertex?

We now show that each of these problems, except $3DREF$ is NP-complete. For $3DREF$, note that any drawing of a graph in a plane displays reflectional symmetry in three dimensions.

First we consider a three dimensional symmetry with no fixed vertex.

Theorem 2. $3DREF_0$, $3DINV_0$ *and* $3DROT_0$ *are NP-complete.*

Proof. This immediately follows from [14] that the detection of *fixed point free* automorphism is NP-complete.

We now consider the problem of three dimensional symmetry with fixed vertex.

Theorem 3. $3DINV$, $3DINV_1$ *and* $3DROT$ *are NP-complete.*

Proof. We can reduce these problems to the corresponding problem in two dimensions, which is NP-complete [18].

To prove the general complexity results in Theorem 1, note that any symmetry group of size greater than 2 must include a symmetry that is either a rotation or an inversion. Thus Theorem 1 follows from Theorem 3.

However, if we restrict the problem into subclasses of planar graphs such as trees and series-parallel digraphs, then $3DSGP$ is solvable in linear time [9, 10]. For drawings of planar graphs in three dimensions, we require two further properties. First, two nonadjacent edges should not intersect, that is, we do not allow edge crossings. Second, the drawing should be *linkless*, that is, no two undirected cycles form a nontrivial link. In the next section, we show that $3DSGP$ is solvable in polynomial time for planar graphs.

4 Planar Graphs

In this section, we present an algorithm for finding three dimensional symmetries in planar graphs. We use connectivity to divide the problem into cases.

1. G is triconnected.
2. G is biconnected.
3. G is one-connected.

Each case relies on the result of the previous case. The following theorem summarizes the result of this section.

Theorem 4. *There is a polynomial time algorithm which computes a maximum size three dimensional symmetry group of planar graphs.*

The proof of this theorem is outlined in the remainder of this paper. The first case is the simplest and described in Section 5. The biconnected case and the one-connected case are more difficult and described in Section 6 and Section 7 respectively.

5 The Triconnected Case

If G is a triconnected planar graph, then we simply use automorphism to find three dimensional symmetry. This is based on the following theorem.

Theorem 5. *[1,15] Every triconnected planar graph G can be realized as the 1-skeleton of a convex polytope P in $\mathbb{R}^3$ such that all automorphisms of G are induced by isometries of P.*

From Theorem 5, we can compute three dimensional symmetry group of triconnected planar graph G from the automorphism group of G. Automorphism of triconnected planar graphs can be computed in linear time [12]. However we need $O(n^2)$ space to represent the automorphism group.

6 The Biconnected Case

If the input graph G is biconnected, then we break it into *triconnected components* in a way that is suitable for the task. The overall algorithm is composed of three steps.

```
Algorithm 3DBiconnected_Planar
1. Construct the SPQR-tree $T_1$ of $G$, and root $T_1$ at its center.
2. Reduction: For each level $i$ of $T_1$ (from the lowest level to the root level)
   a) For each leaf node on level $i$, compute labels.
   b) For each leaf node on level $i$, label the corresponding virtual edge of the
      parent node with the labels.
   c) Remove the leaf nodes on level $i$.
3. Compute a maximum size three dimensional symmetry group at the labeled
   center.
```

We briefly sketch the idea of the algorithm. The algorithm begins by constructing the SPQR-tree for the input biconnected planar graph. Then we use a kind of "reduction" [11]. The operation traverses the SPQR-tree from the leaf nodes to the center level by level. First it computes the labels for the leaf nodes. The labels are a pair of integers, a list of integers and boolean values that capture some information of the symmetry of the leaf nodes. Then it labels the corresponding virtual edge in the parent node and delete each leaf node. This reduction process stops at the root (the center of the SPQR-tree). The center may be a node or an edge. Using the information encoded on the labels, we compute a maximum size three dimensional symmetry group at the center.

We now explain each step of the algorithm.

6.1 SPQR-Tree

We briefly review the definition of the SPQR-tree. For details, see [3].

The SPQR-tree represents a decomposition of a biconnected planar graph into triconnected components. There are four types of nodes in the SPQR-tree T_1 and each node v in T_1 is associated with a graph which is called as the *skeleton* of v (*skeleton*(v)). The node types and their skeletons are:

1. Q-node: The skeleton consists of two vertices which are connected by two multiple edges.
2. S-node: The skeleton is a simple cycle with at least 3 vertices.
3. P-node: The skeleton consists of two vertices connected by at least 3 edges.
4. R-node: The skeleton is a triconnected graph with at least 4 vertices.

In fact, we use slightly different version of the SPQR-tree. We use the SPQR-tree without Q nodes. Also we root the SPQR-tree T_1 at its center. The SPQR-tree is unique for each biconnected planar graph. Let v be a node in T_1 and u is a parent node of v. The graph *skeleton*(u) has one common *virtual edge* with *skeleton*(v), which is called as a *virtual edge* of v.

6.2 Reduction Process

The reduction process takes the SPQR-tree of a biconnected graph. The SPQR-tree T_1 is rooted at the center, based on the following theorem.

Theorem 6. *The center of the SPQR-tree is fixed by a three dimensional symmetry group of a biconnected planar graph.*

Reduction proceeds from the nodes of maximum level toward the center, deleting each leaf node v at the lowest level at each iteration.

The reduction process clearly does not decrease the three dimensional symmetry group of the original graph. This is not enough; we need to also ensure that the three dimensional symmetry group is not increased by reduction. This is the role of the labels. As a leaf v is deleted, the algorithm labels the virtual edge e of v in $skeleton(u)$ where u is a parent of v. Roughly speaking, they encode information about the deleted leaf to ensure that every three dimensional symmetry of the labeled reduced graph extends to a three dimensional symmetry of the original graph.

The reduction process stops when it reaches the root.

6.3 The Labels and Labeling Algorithms

We now define the labels which play very important role in the reduction process.

Our aim is to label the virtual edges so that we can compute a maximum size three dimensional symmetry group of the original graph by computing a maximum size three dimensional symmetry group of the labeled center.

At each stage of the reduction process, labels for the deleted leaf nodes are needed. This is because the reduced graph overestimates a three dimensional symmetry group of the original graph. By checking labels whether they preserve the three dimensional symmetry or not, we can compute a three dimensional symmetry group of the original graph exactly from the reduced graph.

Let v be an internal node of T_1. We say that a virtual edge e of $skeleton(v)$ is a *parent (child) virtual edge* if e corresponds to a virtual edge of u which is a parent (child) node of v. We define a *parent separation pair* $s = (s_1, s_2)$ of v as the two endpoints of a parent virtual edge e.

When we compute the labels of v, we need to delete the parent virtual edge e from $skeleton(v)$. We denote the resulting graph by $skeleton^-(v)$.

Suppose that nodes $v_1, v_2, \dots v_k$ of the SPQR-tree T_1 are deleted at one iteration of the reduction process. These nodes correspond to virtual edges $e_1, e_2, \dots, e_k$ in the level above the current level. For each e_i, we need to compute the following labels.

1. isomorphism code: a pair $Iso(e_i)$ of integers.
2. rotation code: a list L_{e_i} indicating the size of possible rotation groups of $skeleton^-(v_i)$ that fixes the parent separation pair.
3. reflection code:

   a) $Ref_{swap}(e_i)$: a boolean label indicating whether $skeleton^-(v_i)$ has a reflectional symmetry that *swaps* the parent separation pair.
   b) $Ref_{fix}(e_i)$: a boolean label indicating whether $skeleton^-(v_i)$ has a reflectional symmetry that *fixes* the parent separation pair.
4. inversion code: a boolean label $Inv(e_i)$ indicating whether $skeleton^-(v_i)$ has an inversion that swaps the parent separation pair.

Note that we need these labels when the virtual edge is fixed by a three dimensional symmetry of the parent node. We now describe each labeling algorithm.

Computation of an Isomorphism Code. The isomorphism code $Iso(e)$ consists of a pair of integers. This is because the $skeleton(v)$ has an orientation with respect to the parent separation pair [11]. The isomorphism code can be computed in linear time using a planar graph isomorphism algorithm [12].

Computation of a Rotation Code. In addition to the isomorphism code, we attach further information about a rotational symmetry of $skeleton^-(v)$ which fixes the parent separation pair. The algorithm computes a list L_e, the size of the possible rotation groups. In fact, each element of L_e consists of a pair of integers: one which indicates the size of the rotation group and the other which indicates whether the rotational symmetry has a fixed edge.

Note that the rotational symmetry should respect the isomorphism code of the child virtual edge. Further, if the rotational symmetry fixes a child virtual edge, then we need to compute the intersection of the rotation groups. We now state the algorithm.

```
Algorithm Compute_Rotation_Code
1. Compute the list $L_e$ of rotation groups of $skeleton^-(v)$ which fixes the parent
   separation pair and respects the isomorphism codes of child virtual edges.
2. For each element $\rho$ of $L_e$, if there is a child virtual edge $e_j$ in $skeleton^-(v)$
   which is fixed by $\rho$, then compute the intersection of the rotation groups of
   $L_e$ and $L_{e_j}$.
```

Step1 uses the triconnected case in Section 5 if $skeleton^-(v)$ is triconnected. Otherwise it uses `3DBiconnected_Planar` recursively. Step 2 can be computed in linear time using a bit array representation [9].

Computation of a Reflection Code. Further, we need information about a reflectional symmetry. This can be divided into two cases: *swaps* the parent separation pair or *fixes* the parent separation pair. First we describe an algorithm for $Ref_{swap}(e)$.

Note that the reflectional symmetry should respect the isomorphism code of the child virtual edge. Further, if the reflectional symmetry fixes a child virtual edge, then we need to test its label. We now state the algorithm.

```
Algorithm Compute_Reflection_Code_Swap
1. Test $skeleton^{-}(v)$ whether it has a reflectional symmetry $\alpha$ which swaps the
   parent separation pair and respects the isomorphism codes of child virtual
   edges.
2. If $\alpha$ exists, then
   a) For each child virtual edge $e_j$ that is fixed by $\alpha$, check followings:
        i. if $\alpha$ fixes the endpoints of $e_j$, then $Ref_{fix}(e_j) = true$.
       ii. if $\alpha$ swaps the endpoints of $e_j$, then $Ref_{swap}(e_j) = true$.
   b) If one of these properties fails,
      then $Ref_{swap}(e) := false$; else $Ref_{swap}(e) := true$.
   else $Ref_{swap}(e) := false$.
```

An algorithm for computing $Ref_{fix}(e)$ is very similar to the algorithm for computing $Ref_{swap}(e)$. We omit this algorithm from this extended abstract.

Computation of an Inversion Code. Further, we need information about an inversion which swaps the parent separation pair. The algorithm is similar to the case of reflectional symmetry.

```
Algorithm Compute_Inversion_Code
1. Test $skeleton^{-}(v)$ whether it has an inversion $\beta$ which swaps the parent
   separation pair and respects the isomorphism codes of child virtual edges.
2. If $\beta$ exists, then
      If for each child virtual edge $e_j$ that is fixed by $\beta$, $Inv(e_j) = true$,
      then $Inv(e) := true$; else $Inv(e) := false$.
   else $Inv(e) := false$.
```

Note that these labeling algorithms are for R-nodes. When v is a P-node, then we use similar algorithms to the case of parallel compositions in series parallel digraphs [10]. When v is an S-node, then we use similar algorithms to the case of series compositions in series parallel digraphs [10]. We omit these algorithms from this extended abstract.

Based on each labeling algorithm, now we are ready to find three dimensional symmetry at the center.

6.4 Finding Three Dimensional Symmetry at the Center

In this section, we briefly describe how to compute a maximum size three dimensional symmetry group at the labeled center.

Note that the center of the SPQR-tree may be a node c or an edge e. If the center is a node c, then we can further divide into three cases by its type. If c is a R-node, then we use the triconnected case in Section 5 to compute a three dimensional symmetry group. See Figure 1 (a) for example. We construct a three dimensional drawing of $skeleton(c)$ and then replace each child virtual edge e_j by a drawing of $skeleton^{-}(v_j)$. We repeat this process recursively. Note that we place a drawing of $skeleton^{-}(v_j)$ on a plane to maximize symmetry.

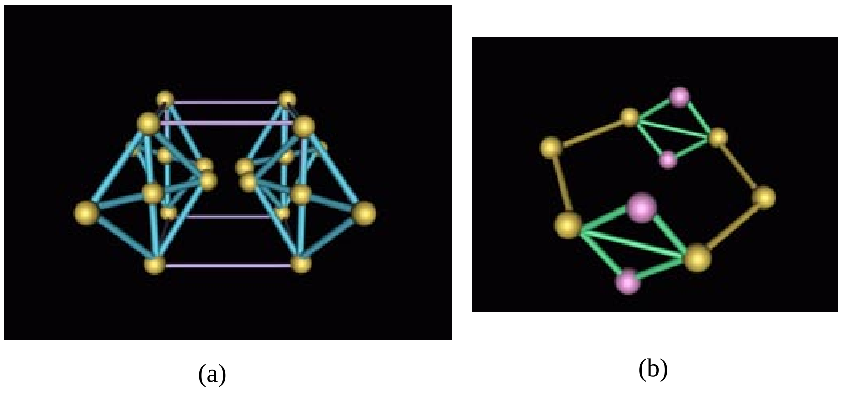

Fig. 1. *Example of (a) R-node and (b) S-node.*

If c is a P-node, then we use similar algorithm to the case of a parallel composition in series parallel digraphs [10]. See Figure 2 (a) for example. If c is an S-node, then we use similar algorithm to the case of labeled cycle. See Figure 1 (b). We omit this algorithm from this extended abstract.

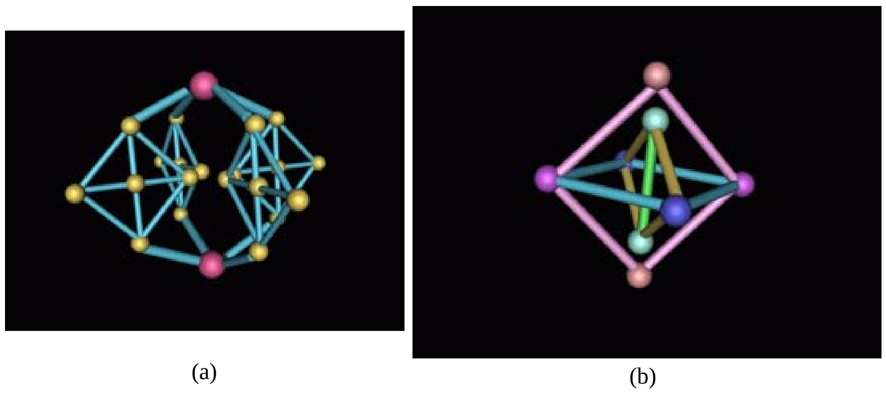

Fig. 2. *Example of (a) P-node and (b) Special case.*

However, there may exist some other node v which is fixed by a three dimensional symmetry group. See Figure 2 (b). We call this special case as *enclosing case*. Thus to find a maximum size three dimensional symmetry group at the center c, we compute these two cases and then find the maximum.

If the center is an edge, then we find the maximum among three cases: parallel composition, reduction composition and enclosing composition. Parallel composition means that we construct a drawing with two labeled edges such as a parallel composition in series parallel digraphs. Reduction composition means that we compute labels of one node u and then delete u by labeling the corresponding virtual edge e of the other node v. Then we compute a three dimensional symmetry group at v. Enclosing composition means that we construct a drawing such as the special case.

We conclude this section by analyzing the time complexity.

Theorem 7. `3DBiconnected_Planar` *takes* $O(n^2)$ *time.*

7 The One-Connected Case

In this section, we give a symmetry finding algorithm for one-connected planar graph G. The algorithm is similar to the biconnected case: we use reduction approach. We also use algorithm `3DBiconnected_Planar` as a subroutine.

The method proceeds from the leaves of the *block-cut vertex tree* (BC-tree) to the center; we may regard the BC-tree as rooted at the center. The overall algorithm is thus composed of three steps.

```
   Algorithm 3DOneconnected_Planar
1. Construct the BC-tree $T_2$ of $G$, and root $T_2$ at its center.
2. Reduction: For each level $i$ of $T_2$ (from the lowest level to the root level)
   a) For each leaf node (block or cut vertex) on level $i$, compute labels.
   b) Remove all the leaf nodes on level $i$.
3. Compute a maximum size three dimensional symmetry group at the labeled
   center.
```

The reduction process is similar to the biconnected case. In this case we take the BC-tree and then compute labels at each leaf node (block or cut vertex). However, the labels are different. We now define the labels.

7.1 The Labels and Labeling Algorithms

We need three types of labels: isomorphism code, rotation code and reflection code. However, these are further divided into the case of a cut vertex or a block. Let B_i represent a block and c_i represent a cut vertex.

1. isomorphism code : an integer $Iso_B(B_i)$ (or $Iso_C(c_i)$).
2. rotation code : a list L_{B_i} (or L_{c_i}) indicating the size of possible rotation groups of B_i (or c_i) which fixes the parent node.
3. reflection code : an integer $Ref_B(B_i)$ (or $Ref_C(c_i)$) indicating whether B_i (or c_i) has a reflectional symmetry which fixes the parent node.

Note that we need these labels when the block or cut vertex is fixed by a three dimensional symmetry of the parent node. We now describe each labeling algorithm.

Computation of an Isomorphism Code of a Block. Suppose that $B_1, B_2, \dots, B_m$ are the blocks on the lowest level and $p_1, p_2, \dots, p_m$ are the parent cut vertices for the blocks. We compute isomorphism code $Iso_B(B_i)$ for each B_i using a planar graph isomorphism algorithm which takes linear time [11, 12]. Note that the isomorphism should respect the isomorphism code of the child cut vertex. We now describe the algorithm.

```
Algorithm Compute_Iso_B
for each $B_i, i = 1, 2, \dots, m$,
    if there is an isomorphism $\alpha$ between $B_i$ and $B_j$ such that,
     a) $\alpha(p_i) = p_j$.
     b) for each cut vertex $c_k$ of $B_i$,
          i. $\alpha(c_k)$ is a cut vertex.
         ii. $Iso_C(c_k) = Iso_C(\alpha(c_k))$.
    then assign isomorphism code such that $Iso_B(B_i) = Iso_B(B_j)$.
```

Computation of a Rotation Code of a Block. A rotation code of a block B consists of a list L_B which represents the size of possible rotation groups of B which fixes the parent cut vertex p of B. In fact, each element of L_B consists of a pair of integers: one which indicates the size of the rotation group and the other which indicates whether the rotational symmetry has a fixed edge which is adjacent to p. We need this information when the block is fixed by a rotational symmetry of the parent node. Thus we compute the list of the size of possible rotation groups which fixes the parent cut vertex p of B.

Note that the rotational symmetry should respect the isomorphism code of the child virtual edge. Further, if the rotational symmetry has a fixed child cut vertex c_j, then we need to compute the intersection of the rotation group of B and the rotation group of c_j. Let $c_1, c_2, \dots, c_k$ be the child cut vertices of B. We now describe the algorithm.

```
Algorithm Compute_Rot_B
1. Compute the list $L_B$ of the size of the rotation groups of $B$ which fixes the
   parent cut vertex $p$ and respect the isomorphism code of child cut vertex,
   together with information about the fixed edge which is adjacent to $p$.
2. For each element $\rho$ of $L_B$, if there is a child cut vertex $c_j$ of $B$ whose $\rho(c_j) =$
   $c_j$, then
   a) Let $f$ be the number of fixed edges in $B$ which is adjacent to $c_j$.
   b) Compute $L'_{c_j}$ from the list $L_{c_j}$ of the size of rotation groups of $c_j$ which
      has at most $2 - f$ fixed child blocks with a fixed edge adjacent to $c_j$.
   c) Compute the intersection of $L_B$ and $L'_{c_j}$.
```

At step 1, we use algorithm 3DBiconnected_Planar in Section 6 to compute the rotation group of B.

Computation of a Reflection Code of a Block. The label $Ref_B(B)$ represents whether the block B has a reflectional symmetry which fixes the parent cut vertex p. Let $c_1, c_2, \ldots, c_k$ be the child cut vertices of B.

In fact, the algorithm computes a ternary value for $Ref_B(B)$. The interpretation of $Ref_B(B)$ is:

1. $Ref_B(B) =$ -1: B has no reflectional symmetry which fixes p.
2. $Ref_B(B) = 1$: B has a reflectional symmetry which has a fixed edge adjacent to p.
3. $Ref_B(B) = 0$: B has a reflectional symmetry which has no fixed edge adjacent to p.

First we find a reflectional symmetry α of B which fixes the parent cut vertex using `3DBiconnected_Planar`. Then we check whether each fixed child cut vertex c_j preserves the reflectional symmetry. For this purpose, we need some information about the reflection code $Ref_C(c_j)$. The interpretation of values of $Ref_C(c_j)$ is:

1. $Ref_C(c_j) = 0$: c_j does not preserve α.
2. $Ref_C(c_j) = 1$: c_j preserves α.

Finally we assign the value, depending on the fixed edge which is adjacent to p. We now state the algorithm.

```
    Algorithm Compute_Ref_B
 1. Test $B$ whether it has a reflectional symmetry $\alpha$ such that
    a) $\alpha(p) = p$.
    b) for each child cut vertex $c_j$ of $B$,
         i. $\alpha(c_j)$ is a child cut vertex.
        ii. If $\alpha(c_j) = c_k$, then $Iso_C(c_j) = Iso_C(\alpha(c_k))$.
       iii. If $\alpha(c_j) = c_j$, then $Ref_C(c_j) = 1$.
 2. If $\alpha$ exists, then
       If there is a fixed edge which is adjacent to $p$,
       then $Ref_B(B) := 1$; else $Ref_B(B) := 0$.
    else $Ref_B(B) :=$ -1.
```

Computation of an Isomorphism Code of a Cut Vertex. Suppose that $c_1, c_2, \ldots, c_k$ are the cut vertices on the lowest level. We compute $Iso_C(c_i)$ for each c_i, $i = 1, 2, \ldots, k$, which represents an isomorphism code of c_i. More specifically, $Iso_C(c_i) = Iso_C(c_j)$ if and only if the subgraph which is rooted at c_i is isomorphic to the subgraph which is rooted at c_j. We now state the algorithm.

```
    Algorithm Compute_Iso_C
 1. For each $c_i$:
    a) Let $B_{i1}$, $B_{i2}$, ... , $B_{im}$ be the child blocks of $c_i$.
    b) $s(c_i) := (Iso_B(B_{i1}), Iso_B(B_{i2}), \ldots, Iso_B(B_{im}))$.
    c) Sort $s(c_i)$.
```

2. Let Q be the list of $s(c_i)$, $i = 1, 2, \ldots, k$.
3. Sort Q lexicographically.
4. For each c_i, compute $Iso_C(c_i)$ as follows: Assign the integer 1 to c_i whose $s(c_i)$ is the first distinct tuple of the sorted sequence Q. Assign the integer 2 to c_j whose $s(c_j)$ is the second distinct tuple, and so on.

Computation of a Rotation Code of a Cut Vertex. The rotation code consists of a list L_c which represents the size of the possible rotation groups of the cut vertex c. In fact, each element of L_c consists of a pair of integers: one which indicates the size of the rotation group and the other which indicates the number of fixed child blocks which has a fixed edge adjacent to c.

Let B_p be the parent block of c. We use L_c when c is fixed by a rotational symmetry of B_p. When we compute L_{B_p}, we need to compute the intersection of two rotation groups.

Suppose that ρ is the rotational symmetry of B_p which fixes c. In B_p, c may have 0, 1, or 2 fixed edges. Thus, we compute the rotation group of c with three cases: 0, 1, or 2 fixed blocks which has a fixed edge adjacent to c.

Let $B_1, B_2, \ldots, B_m$ be the child blocks of c. To compute L_c, we need to compute the intersection with the rotation group of the fixed child block B_j. We use L_{B_j} for this purpose. The list L_{B_j} represents the size of the possible rotation groups of B_j that fixes c, with some information about the fixed edge which is adjacent to c.

The algorithm is an adaptation of the *pyramid* case of the tree algorithm in three dimensions [9]. Note that at most two blocks with a fixed edge adjacent to c can be fixed by the rotational symmetry. Further we can fix more blocks outside the fixed block if it does not have a fixed edge which is adjacent to c. For each fixed block, we need to compute the intersection of the rotation groups. We can compute the intersection of the rotation groups in linear time using the method in [9]. We omit this algorithm from this extended abstract.

Computation of a Reflection Code of a Cut Vertex. The label $Ref_C(c)$ represents whether a cut vertex c preserves a reflectional symmetry which fixes the parent block. Let B_p be the parent block of c and $B_1, B_2, \ldots, B_m$ be the child blocks of c. Suppose that α is a reflectional symmetry which fixes B_p. We use $Ref_C(c)$ to decide whether c preserves α of B_p. More specifically, this indicates that whether $B_1, B_2, \ldots, B_m$ can be attached to c, preserving α.

In fact, we assign an integer to represent whether it preserves α. $Ref_C(c) = 1$ if c preserves α. Otherwise $Ref_C(c) = 0$.

To compute $Ref_C(c)$, we use $Ref_B(B_j)$. The label $Ref_B(B_j)$ indicates that whether B_j has a reflectional symmetry which fixes c. Further, it indicates that whether there is a fixed edge adjacent to c. We now state the algorithm.

```
Algorithm Compute_Ref_C
1. Partition $B_1, B_2, \ldots, B_m$ into isomorphism classes $X_l$ using $Iso_B(B_j)$.
2. Let $y_l := |X_l|$.
```

3. If all y_l are even, then $Ref_C(c) := 1$.
4. If there is an y_l which is odd and $Ref_B(B_j) =$ -1 for $B_j \in X_l$,
   then $Ref_C(c) := 0$.
   else $Ref_C(c) := 1$.

7.2 Finding Three Dimensional Symmetry at the Center

In this section, we briefly describe how to compute a maximum size three dimensional symmetry group at the center. We can compute a maximum size three dimensional symmetry group of the whole graph by computing a maximum size three dimensional symmetry group at the labeled center. This is based on the following theorem.

Theorem 8. *The center of the BC-tree is fixed by a three dimensional symmetry group of an one-connected planar graph.*

The center may be a block or a cut vertex. If the center is a block B, then we use algorithm `3DBiconnected_Planar` in Section 6. If there is a fixed cut vertex c, then we need to check the labels of c. See Figure 3 (a) for example.

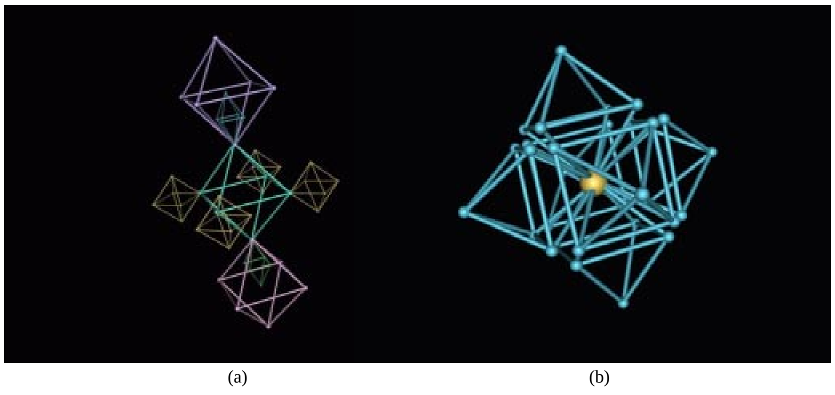

Fig. 3. *Example of (a) B-center and (b) c-center.*

If the center is a cut vertex c, then we use a similar method that was used in the case of trees [9]. If there is a fixed block B, then we need to check the labels of B. See Figure 3 (b). We omit this algorithm from this extended abstract.

We conclude this section by analyzing the time complexity.

Theorem 9. `3DOneconnected_Planar` *takes* $O(n^2)$ *time.*

8 Conclusion

In this paper, we show that the problem of drawing a graph with a maximum number of symmetries in three dimensions is NP-hard. Then we present a polynomial time algorithm for finding maximum number of three dimensional symmetries in planar graphs. As a further work, we would like draw general graphs symmetrically in three dimensions.

References

1. L. Babai, Automorphism Groups, Isomorphism, and Reconstruction, Chapter 27 of *Handbook of Combinatorics*, Volume 2, (Ed. Graham, Groetschel and Lovasz), Elsevier Science, 1995.
2. S. Bachl, Isomorphic Subgraphs, *Graph Drawing*, Lecture Notes in Computer Science 1731, (Ed. J. Kratochvil), pp. 286-296, Springer Verlag, 1999.
3. G. Di Battista and R. Tamassia, On-Line Maintenance of Triconnected Components with SPQR-Trees, *Algorithmica* 15, pp. 302-318, 1996.
4. H. Chen, H. Lu and H. Yen, On Maximum Symmetric Subgraphs, *Graph Drawing*, Lecture Notes in Computer Science 1984, (Ed. J. Marks), pp. 372-383, Springer Verlag, 2001.
5. P. Eades and X. Lin, Spring Algorithms and Symmetry, *Theoretical Computer Science*, Vol. 240 No.2, pp. 379-405, 2000.
6. H. Fraysseix, An Heuristic for Graph Symmetry Detection, *Graph Drawing,* Lecture Notes in Computer Science 1731, (Ed. J. Kratochvil), pp. 276-285, Springer Verlag, 1999.
7. S. Hong, P. Eades and S. Lee, Drawing Series Parallel Digraphs Symmetrically,*Computational Geometry: Theory and Applicatons* Vol. 17, Issue 3-4, pp. 165-188, 2000.
8. S. Hong, P. Eades and S. Lee, An Algorithm for Finding Geometric Automorphisms in Planar Graphs, *Algorithms and Computation,* Lecture Notes in Computer Science 1533, (Ed. Chwa and Ibarra), pp. 277-286, Springer Verlag, 1998.
9. S. Hong and P. Eades, An Algorithms for Finding Three Dimensional Symmetry in Trees, *Graph Drawing*, Lecture Notes in Computer Science 1984, (Ed. J. Marks), pp. 360-371, Springer Verlag, 2001.
10. S. Hong and P. Eades, An Algorithms for Finding Three Dimensional Symmetry in Series Parallel Digraphs, *Algorithms and Computation*, Lecture Notes in Computer Science 1969, (Ed. D. T. Lee and S. Teng), pp. 266-277, Springer Verlag, 2000.
11. J. E. Hopcroft and R. E. Tarjan, Isomorphism of Planar Graphs, *Complexity of Computer Computations,* R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, pp. 131-151, 1972.
12. J. E. Hopcroft and J. K. Wong, Linear Time Algorithm for Isomorphism of Planar Graphs, *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing,* pp. 172-184, 1974.
13. R. J. Lipton, S. C. North and J. S. Sandberg, A Method for Drawing Graphs, In *Proc. ACM Symposium on Computational Geometry*, pp. 153-160, ACM, 1985.
14. A. Lubiw, Some NP-Complete Problems similar to Graph Isomorphism, *SIAM Journal on Computing* 10(1), pp. 11-21, 1981.
15. P. Mani, Automorphismen von Polyedrischen Graphen, *Math. Annalen*, 192, pp. 279-303, 1971.

16. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Trees, *Congressus Numerantium* 64, pp. 159-169, 1988.
17. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Outerplanar Graphs, *Discrete Applied Mathematics* 39, pp. 13-35, 1992.
18. J. Manning, *Geometric Symmetry in Graphs*, Ph.D. Thesis, Purdue Univ., 1990.
19. G. E. Martin, *Transformation Geometry, an Introduction to Symmetry,* Springer, New York, 1982.