Drawing Algorithms for Series-Parallel Digraphs in Two and Three Dimensions[*]

Seok-Hee Hong[1], Peter Eades[2], Aaron Quigley[2], and Sang-Ho Lee[1]

[1] Department of Computer Science and Engineering,
Ewha Womans University, Korea.
{shhong, shlee}@cs.ewha.ac.kr
[2] Department of Computer Science and Software Engineering,
University of Newcastle, Australia.
{eades, aquigley}@cs.newcastle.edu.au

1 Introduction

Series parallel digraphs are one of the most common types of graphs: they appear in flow diagrams, dependency charts, and in PERT networks. Algorithms for drawing series parallel digraphs have appeared in [2,3].

In this paper we describe algorithms which can draw series parallel digraphs in two and three dimensions. Sample drawings are in Figure 1. Specific variations of the algorithms can be used to obtain symmetric drawings, or drawings in which the "footprint" (that is, the projection in the xy plane) is minimized.

This extended abstract is organized as follows. In the next section, we summarize the necessary background for series parallel digraphs. Then concepts for symmetric drawings, especially with respect to series parallel digraphs, are presented in Section 3. The two dimensional algorithm is given in Section 4; building on the two dimensional algorithm, the three dimensional algorithm is given in Section 5.

2 Series Parallel Digraphs

First we review some of the fundamental notions for series parallel digraphs. A digraph consisting of two vertices u and v joined by a single edge is a series parallel digraph, and if G_1 and G_2 are series parallel digraphs, then so are the digraphs constructed by each of the following operations:

- *series* composition: identify the sink of G_1 with the source of G_2.
- *parallel* composition: identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.
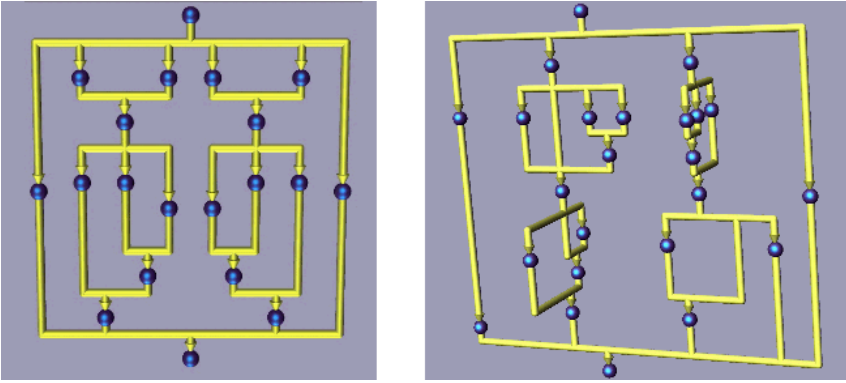

Fig. 1. *Drawings output by the algorithms described in this paper.*

The subgraphs G_1 and G_2 are *components* of G. Series parallel digraphs may be represented as decomposition trees [13], such as in Figure 2(a). Leaf nodes in the tree represent edges in the series parallel digraph, and internal nodes are labeled S or P to represent series or parallel compositions.

Because parallel composition is commutative and both series and parallel compositions are associative, there may be more than one binary decomposition tree for a series parallel digraph. This means that an algorithm based on the binary decomposition tree cannot fully display symmetries. To overcome this we construct the *structure tree* (sometimes called a *canonical decomposition tree*) in which the same composition operations are placed on the same level. Figure 2(b) shows the structure tree corresponding to Figure 2(a). The structure tree can be computed in linear time using the algorithm of Valdes *et al.* [13,14] followed by a simple depth-first-search restructuring operation. The structure tree is unique up to the ordering of siblings.

The Δ-*algorithm* [2,3] draws series parallel digraphs. The algorithm produces grid drawings with straight-line edges. It has been claimed that this algorithm can be varied to display symmetries. However, at best, the Δ-algorithm displays a subset of the set of possible symmetries. The method in Section 4 below displays all possible symmetries.

3   Symmetric Drawings in Two Dimensions

To ensure that *all possible* symmetries are displayed, it is important to use a rigorous model for the intuitive concept of symmetry display. In this section we describe such a model, derived from those introduced by Manning [8,9,10] and Lin [6].

Symmetries of a graph drawing correspond to automorphisms of the graph. However, some automorphisms cannot be displayed as symmetries of any graph layout. Further, it is possible to have two automorphisms, each of which can be

displayed, but for which there is no drawing which displays both. See [4,6] for examples. For these reasons, we define "geometric automorphism group" in the following section, and indicate how this notion relates to symmetry groups of graph drawings.
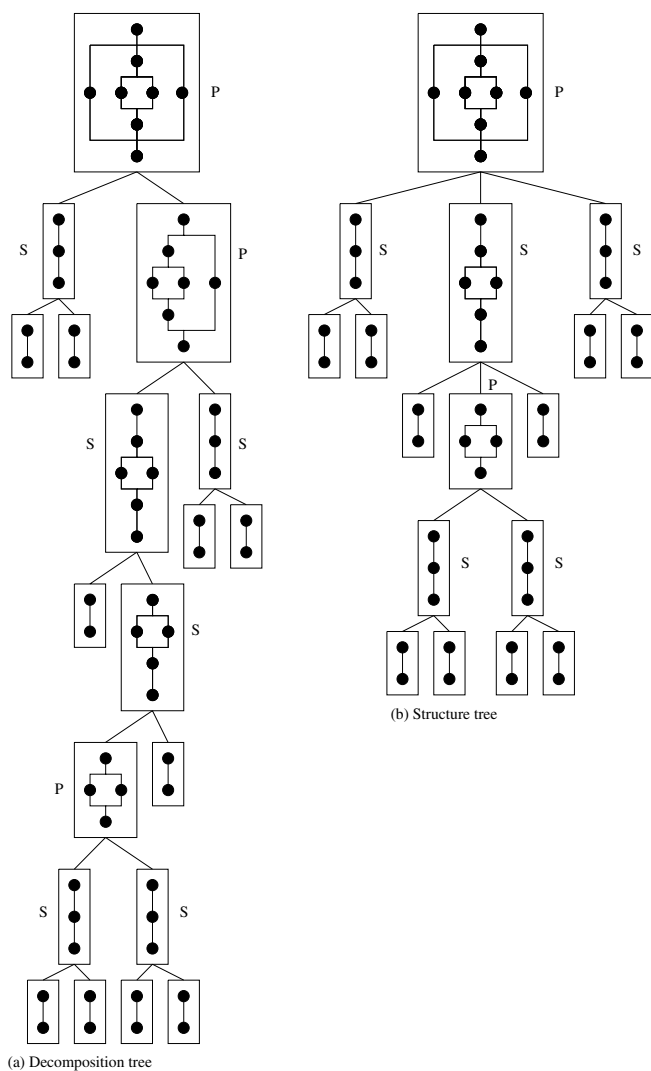

(b) Structure tree

(a) Decomposition tree

Fig. 2. *(a) A binary decomposition tree, and (b) the equivalent structure tree.*

3.1 Geometric Automorphisms of Graphs and Symmetries of Graph Drawings

We need some of the terminology of permutation groups; for more details see [15]. We denote the identity permutation by I. The group generated by $a_1, a_2, \ldots, a_k$ is denoted by $< a_1, a_2, \ldots, a_k >$. If a permutation p acting on a set V has a *fixed element* $v \in V$, that is, $p(v) = v$, then p induces a permutation p_v on $V - \{v\}$. A permutation group P is *semiregular* if each non-identity permutation in P does not have a fixed element.

A permutation p on V is a *rotational* permutation if either $< p >$ or $< p_v >$ (for some $v \in V$) is semiregular, and $| < p > | > 1$. Note that a rotational permutation has at most one fixed element. A permutation p on V is an *axial* permutation if $p^2 = I$ and p is a non-identity permutation. A permutation is a *geometric* permutation if it is either an axial permutation or a rotational permutation.

A permutation group P on V is *geometric* if it is one of the following types:

1. $P =< q >$ where q is an axial permutation; or
2. $P =< p >$ where p is a rotational permutation; or
3. $P =< p, q >$ such that:
   (a) p is a rotational permutation and q is an axial permutation, and
   (b) $< p > \cap < q >= \{I\}$, and
   (c) $qp = p^{-1}q$.

A subgroup P of the automorphism group of a graph G is *geometric* if P is a geometric permutation group on V.

Next we consider graph drawings. The symmetries of a bounded set of points in the plane (such as a two dimensional graph drawing) form a group called the *symmetry group* of the set. A symmetry α of a drawing D of a graph G *induces* an automorphism p of G if the restriction of α to the points representing vertices of G is p. A drawing D of a graph G *displays* a geometric automorphism p of G if there is symmetry α of D which induces p; D *displays* a geometric automorphism group P of a G if D displays every element of P. It is easy to see that the symmetry group of a graph drawing induces a geometric subgroup of the automorphism group of the graph. The converse was proved by Lin [6] and Manning [10]: for every geometric automorphism group P of a graph G, there is a drawing D of G which displays P. In Section 4 we show that for every geometric automorphism group P of a series parallel digraph G, there is a planar drawing D of G, of the type illustrated in Figure 1, which displays P. To apply this result, however, we must compute geometric automorphism groups of series parallel digraphs. In general, the problem of finding a geometric automorphism of a graph is NP-hard [7,10]; it may be strictly harder than the problem of finding the automorphisms of graphs in general (which is merely isomorphism hard [11]). The next section shows that for the case of series-parallel digraphs, finding geometric automorphisms is not difficult.

3.2   Geometric Automorphisms of Series Parallel Digraphs

In this section we sketch an algorithm which finds geometric automorphisms for series parallel digraphs. The geometric automorphism group obtained in this way is used explicitly to draw the graph symmetrically.

For the purposes of this paper, an automorphism of a digraph either maintains the direction of all directed edges, or reverses all directed edges. Thus such an automorphism maps a source to either a sink or a source; further, it maps cut vertices to cut vertices. This leads directly to the next lemma.

Lemma 1. *The automorphism group of a series parallel digraph contains at most two axial geometric automorphisms and at most one rotational geometric automorphism (which must have degree 2.)*

Proof. Omitted.

An important consequence of this Lemma is that for series parallel graphs, there is a single maximal geometric subgroup of the automorphism group. Further, we can derive a method for finding all geometric automorphisms of a series parallel digraph. Roughly speaking, the method proceeds as follows.

1. We construct the structure tree [13,14].
2. We label the structure tree. The labeling is canonical, in the sense that isomorphic blocks have equal labels. The labeling can be computed in linear time by adapting the tree isomorphism algorithm [1,13]. This labeling step is the critical part of the algorithm.
3. We check for the existence of each of the geometric automorphisms mentioned in Lemma 1.

The complete algorithm can be implemented in linear time; details will appear in the full version of this paper.

4   The Two Dimensional Drawing Algorithm

First we describe a simple procedure for giving a visibility representation [12] of a series parallel digraph G. In the representation that we construct, the horizontal line segment for the source is a vertical translation of the horizontal line segment of the sink.

For a graph which consists of single edge, such a representation is simple. Suppose that D_1 and D_2 are visibility representations of series parallel digraphs G_1 and G_2 respectively. If G is a series composition of G_1 and G_2, then we can construct a representation D of G by "stretching" the narrower of D_1 and D_2 and identifying the source of one with the sink of the other; see Figure 3(a). If G is a parallel composition of G_1 and G_2, then we can construct a representation D of G by "stretching" the shorter of D_1 and D_2 and identifying their sources and sinks; see Figure 3(b).

Two traversals of the structure tree can be used to compute the visibility representation. One traversal computes the size of the enclosing rectangle for
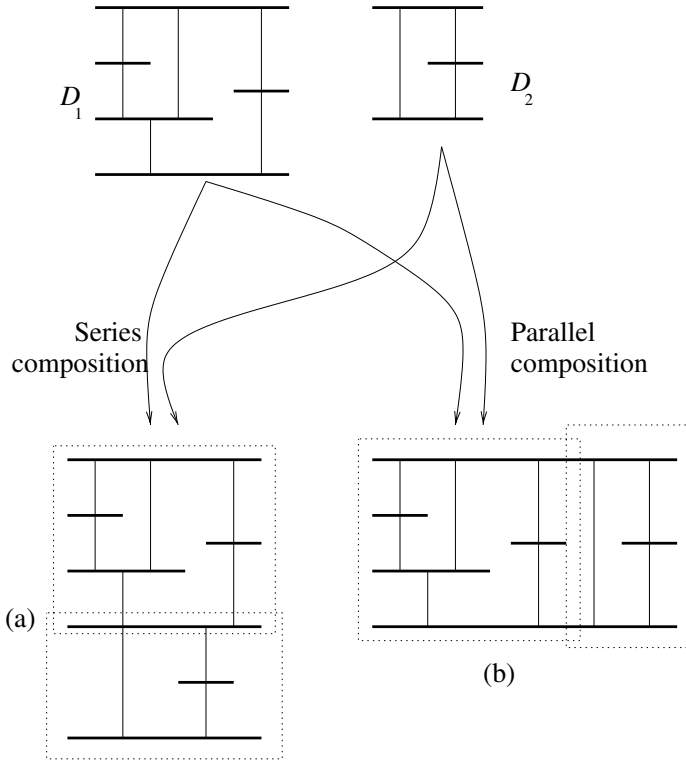
Fig. 3. *Constructing visibility representations of series parallel digraphs.*

each component, the next computes the route for each edge. This works in linear time. The details will appear in the full version of this paper.

This visibility representation can be transformed to a quasi-orthogonal drawing[1] in a simple way; Figure 4 shows a quasi-orthogonal drawing obtained from Figure 3(b). Note that the source and the sink share a vertical line; this is important in Section 5 for drawing in three dimensions.
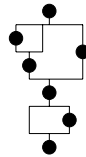

Fig. 4. *Quasi-orthogonal drawing of a series parallel digraph.*

---

[1] Strictly speaking, this is not an orthogonal drawing, since the edges overlap. The precise definition of this class of drawing is involved and we will omit it in this extended abstract.

Our algorithm places parallel components across the page in the same order that they appear in the structure tree. To display symmetry, we need to order the children of each node corresponding to a parallel composition in the structure tree before applying the drawing algorithm. For example, Figure 5(a) has no symmetry; Figure 5(b) is a symmetric drawing of the same graph. The difference between Figure 5(a) and (b) is the left-right order of components of parallel compositions. It can be shown that the order can be chosen to display the any geometric automorphism group of a series parallel digraph.
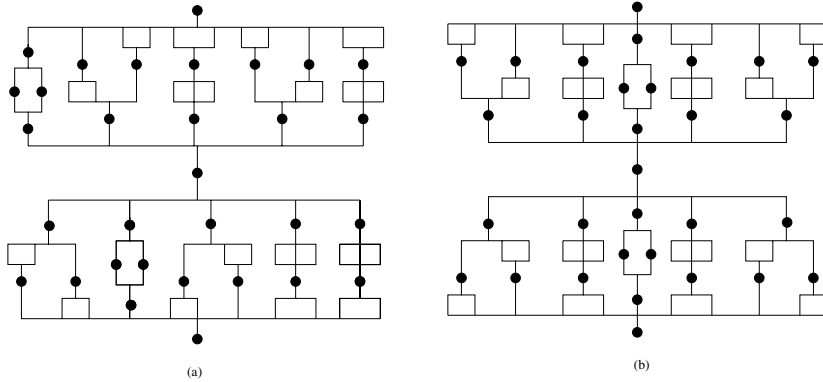

(a)   (b)

Fig. 5. *Symmetric and asymmetric drawings of a series parallel digraph.*

Theorem 1. *Then there is a linear time algorithm which constructs quasi-orthogonal drawings of series parallel digraphs such that the output is planar, and displays every geometric automorphism of the input.*

Proof. The algorithm uses the same labeling technique used for computing geometric automorphisms. Details are omitted in this extended abstract.

The drawings obtained by this algorithm are not grid drawings. However, they do have good area bounds, in the sense that if the minimum distance between a pair of vertices is one, then the drawing is $O(n) \times O(n)$. It is possible to vary the algorithm to give straight-line drawings. However, note that a straight-line drawing may require exponential area (see [2]).

5   Drawing Series Parallel Digraphs in Three Dimensions

In this section we present an algorithm for producing three dimensional drawings of series parallel digraphs. The drawings improve on the resolution of the two dimensional drawings. Note that as long as we keep to the rule that the minimum

distance between a pair of vertices is one, improvements to resolution can be obtained by reducing the extent of the drawing in each dimension.

Consider the drawing in Figure 6, obtained from the algorithm described in the previous section. Suppose that this drawing is in the xz plane within a
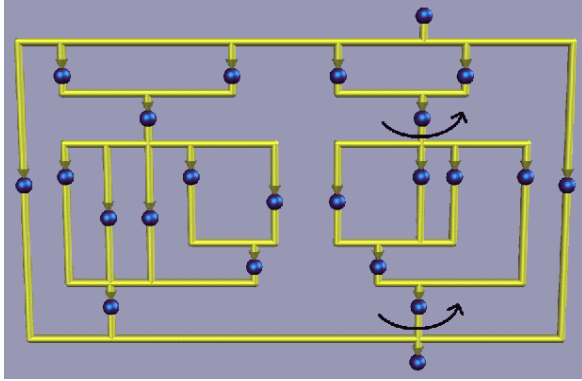

Fig. 6. *Two dimensional drawing, with a rotation indicated.*

three dimensional space, with the z axis vertical on the page. For each parallel node ν in the structure tree, the children of ν are aligned with the x axis. Note the source and sink of each component share a vertical line. We can rotate a component about this line so that it aligns with the y axis; this is illustrated in Figure 7.

The rotation is the basic operation used to improve resolution. For each parallel node ν in the structure tree, we can choose to align the children of ν either in the x direction or in the y direction. Such a choice is illustrated in Figure 8. The result of these choices is illustrated in Figure 9.

The z extent of the three dimensional drawing is fixed by the height of the structure tree; we concentrate on reducing the x and y extents. The *footprint* of a three dimensional graph drawing is the projection of the drawing in the xy plane. To improve the resolution, we need to reduce the size of the footprint. If the minimum enclosing rectangle R of the footprint has dimensions $X \times Y$, then the *size* of the footprint is $\max(X, Y)$.

In fact, we can find a choice of x or y alignment for each parallel composition in such a way that it minimizes the size of the footprint. We use a dynamic programming approach, along the lines of methods for drawing two dimensional "hv-trees" [5].
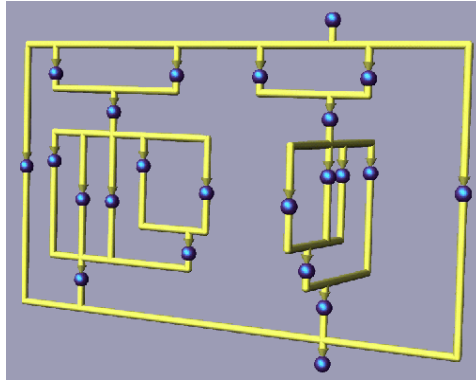
Fig. 7. *Three dimensional drawing obtained by executing the rotation indicated in Figure 6.*
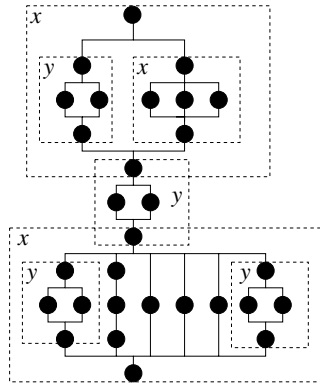

Fig. 8. *Choices of x or y alignment for nodes in the structure tree.*

We say that a layout is *minimal* if its footprint has size $X \times Y$, and there is no layout with footprint of size $X' \times Y'$ where $X' \leq X$, $Y' \leq Y$, and $(X', Y') \neq (X, Y)$. A layout with a (globally) minimum size footprint is among those of minimal footprint. There may be many minimal layouts of a series parallel digraph. The algorithm computes all minimal layouts, and chooses one with a minimum size footprint. The algorithm proceeds from the leaves of the structure tree to the root: at each internal node it computes the minimal layouts of that component.

The footprint of a leaf in the structure tree (that is, an edge in the graph) has dimensions 1×1. Minimal layouts for a component represented by a node ν in the structure tree can be computed from minimal layouts of the components represented by the children of ν, as follows.
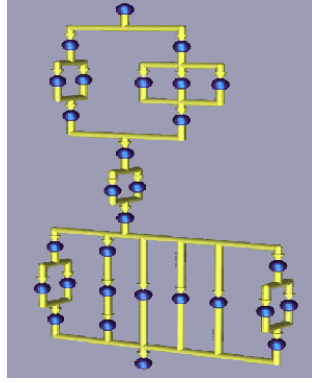
Fig. 9. *The three dimensional drawing resulting from the choices in Figure 8.*

Suppose that ν is a node in the structure tree with children $\mu_1, \mu_2, \ldots, \mu_k$, and the footprint of μ_i has dimensions $X_i \times Y_i$ for $1 \leq i \leq k$.

First suppose that ν represents a series composition. Then the footprint for ν has size $X \times Y$ where

$$X = \max(X_1, X_2, \ldots, X_k),$$
$$Y = \max(Y_1, Y_2, \ldots, Y_k).$$

This is illustrated in Figure 10. As mentioned previously, each component may have many minimal layouts. We store all the minimal layouts for each child μ_i of ν as a list

$$L_{\mu_i} = \big((X_i^1, Y_i^1), (X_i^2, Y_i^2), \ldots, (X_i^{m_i}, Y_i^{m_i})\big)$$
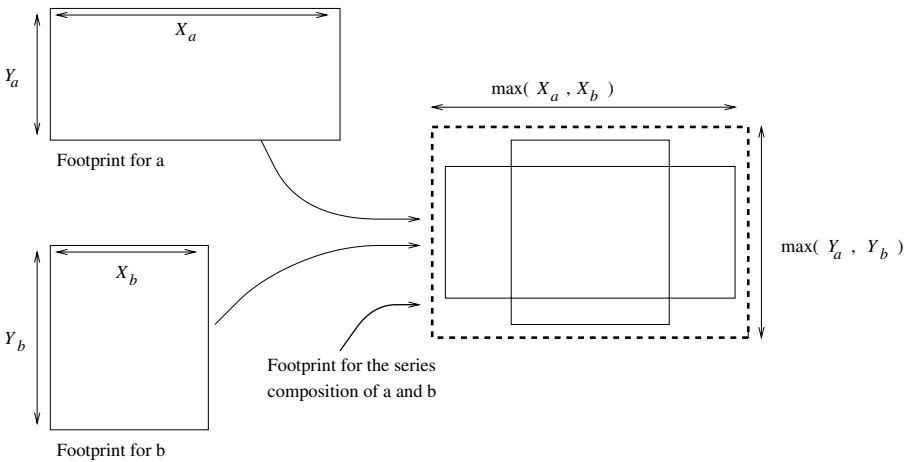

Fig. 10. *The footprint of a series composition.*

of pairs such that L_{μ_i} is decreasing in X coordinate. Note that since each element is minimal, L_{μ_i} is increasing in Y coordinate. A list L_μ of minimal layouts for ν can be computed from $L_{\mu_1}, L_{\mu_2}, \ldots, L_{\mu_k}$ using a kind of merge algorithm below. Here (C_x, C_y) is a candidate for a minimal footprint layout for ν, and p_i is the pointer to the current element of list L_{μ_i}.

1. Choose ℓ such that X_ℓ^1 is maximized; $(C_x, C_y) = (X_\ell^1, Y_\ell^1)$.
2. $L_\mu = ((C_x, C_y))$.
3. For $i = 1, 2, \ldots, k$, $p_i = 1$.
4. $p_\ell = 2$.
5. While $p_i \leq m_i$ for each i:
   (a) $C_x = \max(X^{p_1}, X^{p_2}, \ldots, X^{p_k})$.
   (b) $C_y = \max(Y^{p_1}, Y^{p_2}, \ldots, Y^{p_k})$.
   (c) Suppose that $(LAST_x, LAST_y)$ was the last element appended to L_ν.
       If $LAST_x < C_x$ and $LAST_y > C_y$
       then append (C_x, C_y) to L_ν;
       else replace $(LAST_x, LAST_y)$ by (C_x, C_y) in L_{μ_i}.
   (d) Choose ℓ such that $X_\ell^{p_\ell}$ is maximized.
   (e) $p_\ell = p_\ell + 1$.

Step 5(c) ensures that the elements of L_ν are minimal. The choice of ℓ at step 5(d) can be done using an indexed priority queue; this has amortised constant time per access. Thus the algorithm takes time proportional to the sum of the lengths of the input lists.

In the case that ν represents a parallel composition, we need to choose whether to align the children in the x direction or in the y direction. An x alignment for ν has dimensions $X \times Y$ where

$$X = X_1 + X_2 + \cdots + X_k,$$
$$Y = \max(Y_1, Y_2, \ldots, Y_k),$$

and a y alignment for ν has dimensions $X \times Y$ where

$$X = \max(X_1, X_2, \ldots, X_k),$$
$$Y = Y_1 + Y_2 + \cdots + Y_k.$$

One can use these equations with merge operations in a similar but more complex way to the method for a series composition (see [5]) to compute all minimal footprint layouts for the graph. The details are omitted for this extended abstract.

The complete algorithm works in time $O(n^2)$.

Theorem 2. *There is an algorithm which computes a minimum size footprint layout of a series parallel digraph in time $O(n^2)$.*

Proof. Omitted.

6   Conclusion

In this paper we have introduced two algorithms for drawing series parallel digraphs. One constructs two dimensional drawings which display symmetries, the other constructs three dimensional drawings with a footprint of minimum size.

Future work will include combinations of these two algorithms: we would like to display as much symmetry as possible in a three dimensional drawing of small footprint.

References

1. A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
2. P. Bertolazzi, R.F. Cohen, G. D. Battista, R. Tamassia and I. G. Tollis, How to Draw a Series-Parallel Digraph, *International Journal of Computational Geometry and Applications,* 4 (4), pp 385-402, 1994.
3. R.F. Cohen, G. D. Battista, R. Tamassia and I. G. Tollis, Dynamic Graph Drawing:Trees, Series-Parallel Digraphs, and Planar st-Digraphs, *SIAM Journal on Computing,* 24 (5), pp 970-1001, 1995.
4. P. Eades and X. Lin, Spring Algorithms and Symmetry, *Computing and Combinatorics*, Springer Lecture Notes in Computer Science 1276, (Ed. Jiang and Lee), 202 –211.
5. P. Eades, T. Lin and X. Lin, Minimum Size h-v Drawings, *Advanced Visual Interfaces* (Proceedings of AVI 92, Rome, July 1992), World Scientific Series in Computer Science 36, pp. 386 - 394.
6. X. Lin, *Analysis of Algorithms for Drawing Graphs*, PhD thesis, University of Queensland 1992.
7. A. Lubiw, Some NP-Complete Problems similar to Graph Isomorphism, *SIAM Journal on Computing* 10(1):11-21, 1981.
8. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Trees, *Congressus Numerantium* 64, pp. 159-169, 1988.
9. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Outerplanar Graphs, *Discrete Applied Mathematics* 39, pp. 13-35, 1992.
10. J. Manning, *Geometric Symmetry in Graphs,* PhD Thesis, Purdue University 1990.
11. R.A. Mathon, A Note on Graph Isomorphism Counting Problem, *Information Processing Letters* 8, 1979, pp. 131-132.
12. R. Tamassia and I. G. Tollis, A unified approach to visibility representations of planar graphs, *Discr. and Comp. Geometry* 1 (1986), pp. 321-341.
13. J. Valdes, R. Tarjan and E. Lawler, The Recognition of Series-Parallel Digraphs, *SIAM Journal on Computing* 11(2), pp. 298-313, 1982.
14. J. Valdes, Parsing Flowchart and Series-Parallel Graphs, *Technical Report STAN-CS-78-682*, Computer Science Department, Stanford University, 1978.
15. H. Wielandt, *Finite permutation groups*, Academic Press, 1964.