Algorithmica (2000) 26: 100–125
DOI: 10.1007/s004539910006

Algorithmica

Efficient Orthogonal Drawings of High Degree Graphs[1]

A. Papakostas[2] and I. G. Tollis[3]

Abstract. Most of the work that appears in the two-dimensional orthogonal graph drawing literature deals with graphs whose maximum degree is four. In this paper we present an algorithm for orthogonal drawings of simple graphs with degree higher than four. Vertices are represented by rectangular boxes of perimeter less than twice the degree of the vertex. Our algorithm is based on creating groups/pairs of vertices of the graph. The orthogonal drawings produced by our algorithm have area at most $(m-1) \times (m/2+2)$. Two important properties of our algorithm are that the drawings exhibit a small total number of bends (less than m), and that there is at most one bend per edge.

Key Words. Graph drawing, Orthogonal graph drawing, Algorithms, Information visualization, Graphical user interfaces.

1. Introduction. Most of the work that appears in the two-dimensional orthogonal graph drawing literature deals with graphs whose maximum degree is four [3], [8], [12], [14]–[16], [18], [19], [22]. The drawings produced by these algorithms require at least two bends per edge. This is a big restriction since in most applications graphs generally have degree higher than four. Orthogonal drawings of graphs of high degree are useful for visualizing database schemas or the internal structure of large software systems. In these applications, vertices are boxes called *tables* containing fields, which are placed vertically one below the other. These fields can be attributes of some entity (in the case of a database) or a specification list (in the case of a software module). For a survey of graph drawing algorithms and other related results, see [5]. In this paper we consider graphs with n vertices and m edges. Also, the area of a drawing is expressed as $width \times height$.

Fößmeier and Kaufmann [11] presented an extension of Tamassia's algorithm [20] for minimizing the total number of bends of planar embedded graphs of maximum degree four to planar graphs of arbitrary degree. The vertices are represented by squares of size depending on the degree of the vertex. No discussion is made on the area or the number of bends per edge of the resulting orthogonal drawings. Their algorithm runs in $O(n^2 \log n)$ time.

GIOTTO [6] is another algorithm for orthogonal drawings of graphs of degree higher than four. It is also based on Tamassia's algorithm [20] for minimizing the total number of bends. Dummy vertices are used to represent crossings, and all vertices of the drawing are represented by boxes. The disadvantage of GIOTTO is that the boxes may grow arbitrarily in size, regardless of the degree of the vertex they represent. Experimental

[1] This research was supported in part by NIST, Advanced Technology Program Grant Number 70NANB5H11.
[2] Network Management Division, NEC America, 1525 W. Walnut Hill Lane, Irving, TX 75038, USA. papakos@asl.dl.nec.com.
[3] Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688, USA. tollis@utdallas.edu.

Received January 15, 1997; revised February 1, 1998. Communicated by T. Nishizeki, R. Tamassia, and D. Wagner.

results of GIOTTO's performance with respect to various aesthetic measures on a large database of about 11,500 graphs can be found in [6].

Even and Granot [9] presented two algorithms for placing rectangular modules and connections between them in an orthogonal fashion in the plane. The size of the modules and the positions of the terminals around the modules are part of the input. The edges of the graph are attached to the terminals of the modules. Their first algorithm produces planar orthogonal drawings starting from a visibility representation [17], [21] of the given graph. Their second algorithm places the modules diagonally in the plane and routes the edges around them. In both cases, the final orthogonal drawing has area at most $(W+m)\times(H+m)$ where W (H) is the total width (height) of all modules, and m is the number of edges of the graph. No edge has more than four bends and both algorithms run in $O(m)$.

There is a short discussion in [3] about producing a drawing of a general (i.e., not necessarily planar) biconnected graph with degree higher than four. The authors extend their algorithm for maximum degree four graphs and they propose an approach in which each vertex is represented by a vertical line segment consisting of multiple grid points. The area of the resulting drawing is at most $(m-n+1)\times(m-n/2+n_2/2)$, where n_2 is the number of vertices of degree two. If we change the vertex representation to that of a box of the same height as the line segment and width one, we obtain an orthogonal drawing with area at most $(m+1)\times(m-n/2+n_2/2)$. The total number of bends is at most $2m-2n+4$ and each edge has at most two bends. Drawings resulting from this approach turn out to be very tall, skinny, and they have many crossings.

In this paper we present a different approach for dealing with orthogonal drawings of simple graphs with degree higher than four. Vertices are represented by rectangular boxes of perimeter less than twice the degree of the vertex. Our algorithm is based on creating groups/pairs of vertices of the graph both before and during the construction of the graph drawing. The orthogonal drawings produced by our algorithm have area at most $(m-1)\times(m/2+2)$. Two important properties of our algorithm are that the drawings exhibit small total number of bends (less than m), and that there is at most one bend per edge. For more details on applications of this technique and for results on 3D orthogonal drawing of high degree graphs see [13].

Independently, Biedl [1] recently presented a technique that produces orthogonal drawings of high degree graphs with at most m bends and $((m+1+3n)/2)\times((m+1+3n)/2)$ area [2]. For rather dense graphs, the above area bounds are better than than the ones in this paper. However, for sparse graphs that have some nodes of high degree, our area bounds are better than the ones of [1]. Notice that in practice, graphs that are involved in visualization applications are typically rather sparse. For example, in the large experimental study reported in [6] the average degree in all 11,500 graphs considered was about 2.7 (in other words, the total number of edges was about $1.35n$). This is also the case with many other collections of graphs.

2. A Box Representation for Vertices and a Simple Algorithm. Clearly, a point representation for vertices of a graph does not suffice if we want to remove any restriction about the degree of its vertices. In this paper, we use a *rectangular box* to represent vertices of the graph. Using boxes to represent vertices has the advantage that the area inside the box can be used to record information (e.g., label, database tables, etc.) pertaining to some vertex. From an aesthetic point of view, using boxes reduces significantly the number of bends in the drawing, both total and per edge. We call the boundary edges

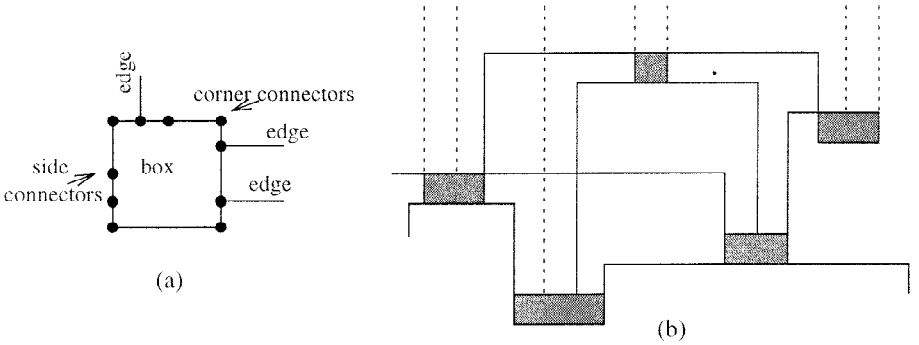

Fig. 1. (a) A box with its sides and connectors, (b) a sample orthogonal drawing produced by the Simple Algorithm.

of a box *sides*. Figure 1(a) shows a box with its four sides, *top*, *left*, *bottom*, and *right*. Each side has a number of *connectors* where all the edges of the graph incident to the vertex that this box represents are attached to. Each connector point can be incident to only one edge (except for the four corner connectors).

When a box is used to represent a vertex in an orthogonal drawing, its sides lie on lines of the underlying integer grid and its connectors have integer coordinates. Also, the area of the box is sufficiently large so that all the incident edges can be attached to different connectors of the box boundary. We present now a Simple Algorithm for producing an orthogonal drawing. The Simple Algorithm inserts the vertices in the drawing, one vertex at a time. For simplicity, we assume that the given graph is biconnected, and that an st-numbering [10] has been computed on the graph. Note that the edges of the graph are directed from lower to higher numbered vertices, as a result of the st-numbering. The size (and also the area) of each box to be inserted (say v) is decided when v is the next vertex to be inserted in the drawing. The box size depends on the number of incoming and outgoing edges associated with vertex v.

All outgoing edges of vertex v are attached to the top side connectors (see Figure 2(a)). This implies that the width of the box is at least equal to the number of outgoing edges of the vertex. If the box has only one outgoing edge, we still use two columns for the box (i.e., a box with width one, see Figure 2(b)). We also use a box of width one for the unique sink, that is the vertex with no outgoing edges.

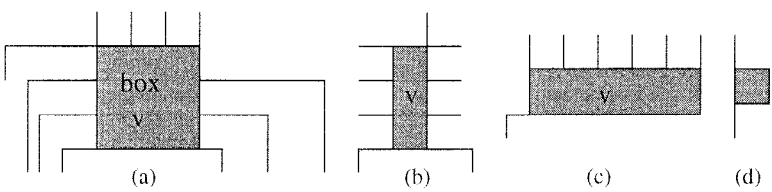

Fig. 2. Various types of box v: (a) seven incoming and four outgoing edges, (b) only one outgoing edge, (c) only one incoming edge. (d) A degree two vertex represented by a box.

The incoming edges of v are split between the right and left side connectors. More specifically, if v has $indeg(v)$ incoming edges, then $\lfloor indeg(v)/2 \rfloor$ incoming edges are attached to the right side and the remaining $\lceil indeg(v)/2 \rceil$ incoming edges are attached to the left side of box v (see Figure 2(a)). If v has one or two incoming edges, we still use two rows for the box (i.e., a box with height one, see Figure 2(c)). We also use a box of height one for the unique source, that is the vertex with no incoming edges. Each incoming edge of v originally has vertical direction since it is an outgoing edge of some other box which has already been placed. Then it bends only once, and finally assumes horizontal direction before it is attached to the appropriate (right or left) side connector of v.

At most $\lceil indeg(v)/2 \rceil$ new rows and $outdeg(v)$ (i.e., number of outgoing edges of v) new columns need to open up when the Simple Algorithm inserts the next vertex v. Vertices with less than two outgoing edges are the exception since two new columns need to open up to accommodate their boxes. Also, vertices with at most two incoming edges are exceptions, since two new rows need to open up for their boxes.

Before we describe how boxes are placed, we give some definitions. If v is the next vertex to be inserted, we locate v's incoming edges and the columns which they are assigned to. The vertices of the drawing where v's incoming edges come from, are v's *predecessors*. Let us assume first that $indeg(v)$ is even. Since all the columns of the current drawing are always ordered from left to right, we find the two columns c_1 and c_2 holding the incoming edges e_1 and e_2, respectively, of v with the following properties:

- c_1 is to the left of c_2,
- there are $indeg(v)/2 - 1$ columns holding incoming edges of v to the left of c_1, and
- there are $indeg(v)/2 - 1$ columns holding incoming edges of v to the right of c_2.

Edges e_1 and e_2 are called *median incoming edges* of vertex v, and more specifically, e_1 is the *left median incoming* edge and e_2 is the *right median incoming* edge of v. If $indeg(v)$ is odd, there is only one median incoming edge e; in this case, if c is the column holding e, there are $\lfloor indeg(v)/2 \rfloor$ columns holding incoming edges of v to the left and right of c. The function of the median incoming edge(s) is to establish where to split the incoming edges between the right and left side of v.

When the Simple Algorithm places vertex v, it first creates the box to represent v and then it opens up the appropriate number of new columns between v's median incoming edges and new rows above the current drawing, and places v there. If v has only one median incoming edge, v is placed to the right of this edge. Figure 1(b) shows a few vertices placed by the Simple Algorithm, as part of a larger orthogonal drawing. If we are given an n-vertex m-edge biconnected graph, then the following theorem holds for the Simple Algorithm:

THEOREM 2.1. *Let G be a biconnected graph with n vertices and m edges. Consider each edge as being oriented from its lower to its higher numbered vertex, for a given st-numbering of G. The Simple Algorithm produces an orthogonal drawing of G in $O(m)$ time. Each vertex v with degree $deg(v)$ is represented by a box whose perimeter is less than $2 \times deg(v)$. The width of the resulting orthogonal drawing is at most $m + n_{\text{out}\,1}$, and the height is at most $m/2 + n_{\text{odd}}/2 + n_{\text{in}\,1} + n_{\text{in}\,2}$, where $n_{\text{out}\,1}$ is the number of vertices with one outgoing edge, n_{odd} is the number of vertices with an odd number of incoming*

edges, and $n_{\text{in}\,1}$ ($n_{\text{in}\,2}$) *is the number of vertices with one* (*two*) *incoming edge*(*s*). *The drawing has at most m bends, while no edge bends more than once.*

PROOF. If we use the data structure proposed by Dietz and Sleator [7], we can answer column order queries for two columns in $O(1)$ time. This means that the (left or right) median incoming edge(s) can be computed in $O(indeg(v))$ time, where v is the vertex that is being inserted. Hence, the total running time of the Simple Algorithm is $O(m)$. For each vertex v, the box representing v has width at most $outdeg(v)$ and height at most $\lceil indeg(v)/2 \rceil$. Hence, the perimeter of each box is at most $2 \times deg(v)$.

The bounds on the area and the number of bends of the orthogonal drawing follow from the above discussion. More specifically, the width involves the total number of outgoing edges, m, and the fact that boxes with one outgoing edge require two columns. The height involves half the total number of incoming edges, $m/2$, and the following two facts: boxes with one or two incoming edges take up two rows; a box v with an odd number of incoming edges requires $indeg(v)/2 + \frac{1}{2}$ rows. □

Notice that if the graph is of constant degree, then at most $O(1)$ time is spent each time a vertex (box) is inserted, for a total of $O(n)$ running time. The reason we do not attach any incoming edges to the bottom side of the box (except the corner connectors) is twofold: First, the incoming edges of a box are not necessarily on contiguous grid columns. Second, if some incoming edge was using v's bottom side, then we would have to stretch the box which this edge was coming from (say u) horizontally, in order to create space for the rest of u's outgoing edges.

3. Algorithm BOX_ORTHOGONAL: Preprocessing Phase. In this section we describe a new algorithm which produces orthogonal drawings for any given graph, with better bounds than the Simple Algorithm of the previous section. We call this new algorithm BOX_ORTHOGONAL, and it has many similarities to the Simple Algorithm. It uses boxes to represent vertices of the graph, except for vertices of degree one, two, some vertices of degree three, and some special cases of degree four, which are represented by points. We use the point representation for some small degree vertices mainly for aesthetic reasons. If a degree two vertex with one incoming and one outgoing edge is represented by a box (see Figure 2(d)), then there is clearly a waste of space.

The *preprocessing phase* of the algorithm consists of two operations: First, we compute a numbering of the vertices of the graph. This numbering specifies the order that the vertices will follow to enter the drawing. The second operation is to perform a *grouping/pairing* of some of its vertices. A *pair* consists of two vertices and a *group* consists of more than two vertices. All vertices belonging to the same group/pair are considered and placed together by Algorithm BOX_ORTHOGONAL. As we will see later, grouping/pairing contributes to orthogonal drawings with better bounds in terms of area.

We start by describing the vertex numbering. We compute an *initial numbering* for the given graph; this numbering may be modified (if necessary) resulting to the *final numbering*. The numbering provides the vertex order required by the algorithm. Let us assume that we are given a connected n-vertex graph G. If G is biconnected, then the

initial numbering is an st-numbering [10] of the vertices of G. If G is not biconnected, then we break G into its biconnected components, and compute a numbering for G so that:

- there is a unique source (the source is assigned number 1),
- there are one or more sinks,
- the numbering is an st-numbering within each biconnected component, and
- there is exactly one sink per biconnected component, which has the highest number among all vertices of the component.

The initial numbering of the vertices implies an orientation of the edges of G, so that every edge that connects two vertices is directed from the lower numbered vertex to the higher numbered vertex. It is also clear that the initial numbering can be computed in $O(m)$ time, where m is the number of edges of G. A vertex v with b incoming and a outgoing edges is called a b, a-*vertex*.

The purpose of modifying the initial numbering to the final numbering is to eliminate specific patterns of vertex types that result in drawings with large area under the Simple Algorithm. More specifically, we consider each b, 1-vertex whose outgoing edge enters a vertex w which is neither a sink, nor a 1, a-vertex ($a > 1$), nor another b, 1-vertex ($b \geq 1$). From these b, 1-vertices, we form a set B consisting of b, 1-vertices, v, that satisfy exactly one of the following:

- b is odd and $b \geq 5$,
- $b = 1$ or $b = 3$, and w is a 2, a-vertex ($a > 1$) so that w's other predecessor is not a b', 1-vertex with $b' = 1$ or $b' = 3$.

We scan the vertices of directed graph G looking for b, 1-vertices that belong to set B. We reverse the direction of the edge between the b, 1-vertex and w. As a result, the b, 1-vertex becomes a sink $(b + 1)$, 0-vertex, and $b + 1$ is an even number. Notice that no edge reversal creates a new b, 1-vertex. Also, no directed cycle is formed since one of the vertices affected by the edge reversal is now a sink.

Let G' be the directed graph resulting after all possible edge reversals. G' implies many numberings of the vertices that are consistent with the edge directions. We construct one such numbering using a topological sorting algorithm in $O(m)$ time. This is the order that will be followed when placing the vertices in the drawing. Clearly, the whole process for producing the final numbering takes $O(m)$ time. The second operation performed in the preprocessing phase is grouping/pairing. Vertex grouping/pairing takes place in four passes. In the first pass, we scan the vertices of G' looking for specific patterns of vertices. If a vertex satisfies more than one patterns, we only consider the first pattern it satisfied in the following provided order. More specifically, we look for:

1. b, 1-vertices ($b \geq 1$) whose outgoing edge enters a 1, a-vertex ($a > 1$); we pair the two vertices.
2. c, d-vertices such that $c > 1$ is odd and $d \geq 2$, which are predecessors of at least one 1, a-vertex ($a > 1$); we pair each such vertex with the 1, a-vertex.
3. c, d-vertices such that $c > 1$ is odd and $d \geq 2$, which have at least one predecessor b, 1-vertex ($b \geq 1$); we pair each such vertex with its predecessor b, 1-vertex.

In the second pass, we look for 2, a-vertices ($a > 1$) which have at least one predecessor b, 1-vertex (b even), or two predecessor b, 1-vertices so that $b = 1$ or $b = 3$. In

the first case, we pair the 2, a-vertex with its predecessor b, 1-vertex, and in the second case we group the 2, a-vertex with both its predecessor b, 1-vertices.

In the last two passes we only consider 1, a-vertices ($a > 1$) that have not been grouped/paired in a previous pass. In the third pass, we scan the 1, a-vertices in decreasing order of their assigned number. For as long as we encounter a 1, a-vertex whose predecessor is also a 1, a-vertex, we put them both in the same group. The groups of 1, a-vertices formed this way are also called *chains*. In the fourth and final pass, we scan the remaining 1, a-vertices and we pair 1, a-vertices that have the same predecessor. Vertex grouping/pairing can be completed in $O(m)$ time.

LEMMA 3.1. *After vertex grouping/pairing is complete, any* 1, a*-vertex,* v*, which does not belong to a group/pair has a predecessor,* u*, which is exactly one of the following*:

1. u *is a source,*
2. u *is a* 1, a*-vertex that participates in some group/pair,*
3. u *is a* c, d*-vertex where* $c, d \geq 2$.

Also, there cannot be another 1, a*-vertex,* v'*, which does not belong to any group/pair and has the same predecessor as* v.

PROOF. Let us consider a 1, a-vertex, v, which does not belong to any group/pair. If v's predecessor, u, were a b, 1-vertex, then v and u would have formed a pair in the first pass. Therefore, u can be either a source, or another 1, a-vertex, or a c, d-vertex where $c, d \geq 2$. If u is a 1, a-vertex, it must belong to some group/pair, because if it did not, v and u would have formed a chain. Finally, if there were another 1, a-vertex, v', which did not belong to any group/pair and had vertex u as its predecessor, then v and v' would have been paired in the fourth pass. □

The end of the grouping/pairing operation is also the end of the preprocessing phase. The *drawing phase* is the phase that follows preprocessing. This is the time when we actually draw the resulting (after the preprocessing phase) graph in the plane.

4. Algorithm BOX_ORTHOGONAL: Placement Techniques. One disadvantage of the Simple Algorithm is that vertices with one outgoing edge and sinks contribute extra to the width of the drawings (see Theorem 2.1). In Algorithm BOX_ORTHOGONAL, we follow the general rules of the Simple Algorithm for creating and placing boxes of vertices in the orthogonal drawing. However, we introduce a different placement technique for vertices with one outgoing edge, and this is discussed in Section 4.1. The way we use to insert b, 1-vertices may call for a special placement of some vertices with at least two incoming and at least two outgoing edges. This is presented in Section 4.2. Finally, Section 4.3 discusses how to insert sources and sinks. Except for the situations described in Sections 4.1–4.3 which immediately follow, any other individual vertex insertion in the drawing is handled in a way similar to the Simple Algorithm. In what follows, we assume that v is the next vertex considered for insertion by the drawing algorithm.

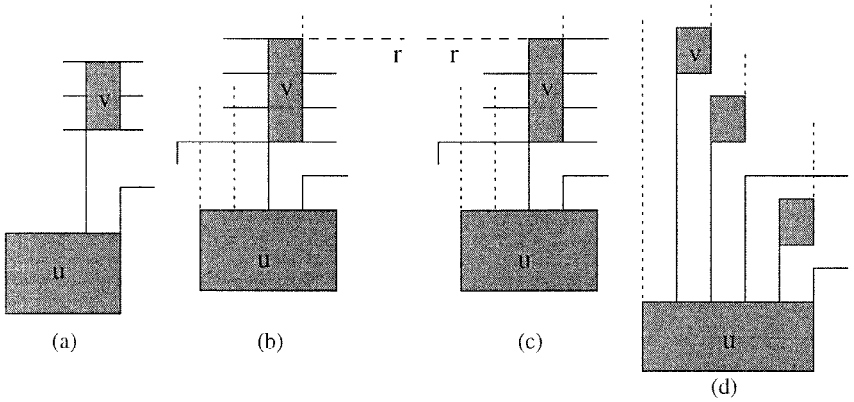

Fig. 3. Placing v with one outgoing edge to the top ot its predecessor u.

4.1. *Vertex v Has Only One Outgoing Edge.* Let us first consider the case where the b, 1-vertex v has three or more incoming edges. In this case, we use a box to represent v. The width of the box is always one. The height of the box is typically $\lfloor indeg(v)/2 \rfloor - 1$. However, if $indeg(v) = 3$ or vertex v is paired with another vertex that follows v in the drawing phase, then the height is $\lfloor indeg(v)/2 \rfloor$. We will see pairs of that type in Section 5.1 (where a b, 1-vertex is paired with a 1, a-vertex) and Section 5.2 (where a b, 1-vertex is paired with a vertex with an odd number of more than one incoming edges). In the case where the b, 1-vertex v has one or two incoming edges, we represent v with a point in the plane.

Before we actually place vertex v in the drawing, we compute its median incoming edge(s). If v has only one median incoming edge, we place v directly above box u where its median incoming edge comes from (see Figure 3(a)). Note that half of v's incoming edges are attached to the left side of v, and the other half to the right side of v. If vertex v has both left and right median incoming edges, we pick one of the two for placing v, depending on which portion of the row of v's top side we want to reuse (if any). For example, placing v directly above the box where its right median incoming edge comes from (see Figure 3(b)) allows reusing the portion of row r to the right of v. Figure 3(c) demonstrates the opposite case where v is placed above the box where its left median incoming edge comes from. This row reuse is important when placing a pair containing a b, 1-vertex.

When we insert b, 1-vertices our goal is to ensure that all boxes placed above a single common predecessor vertex share each other's columns. This is shown in Figure 3(d), where v is the next vertex to be inserted and u is its predecessor above which v is placed. As we can see in Figure 3(d), we use the next available column of u's outgoing edges from right to left, and we attach box v along that column. Note that there is no bend in the edge connecting the box of any b, 1-vertex placed above u and box u itself.

If vertex v is represented by a point, we follow the same procedure except that we attach v along the next available column of u's outgoing edges from left to right (see Figure 4(a)). The reason we do not mix b, 1-vertices represented by points and boxes

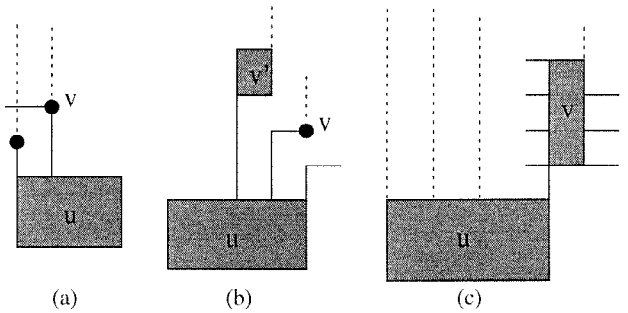

Fig. 4. (a) Placing v above its predecessor vertex u when v is a point, (b) bending an edge in order to reuse a column is avoided, (c) vertex v opens up one new column.

is to avoid having to bend one of the incoming edges of the vertex represented by a point. This is illustrated in Figure 4(b), where we bend edge e in order for box v' (which follows later) to reuse a column. Moreover, if v is a 2,1-vertex, we use its right median incoming edge for placing it (see Figure 4(a)).

The advantage of placing b, 1-vertices in this way is that we do not have to open up any new columns to insert such a vertex. We simply use existing columns. The only exception to this is when both of the following two situations hold: (a) We place the first b, 1-vertex ($b > 2$) v above a vertex u with at least two outgoing edges, and (b) vertex v is represented by a box. This is depicted in Figure 4(c): b, 1-vertex v opens up one new column to the right of box u. A b, 1-vertex whose insertion requires opening up one new column in the drawing is a *column-taker*. On the other hand, a b, 1-vertex that entirely reuses existing columns when it is placed in the drawing is a *column-saver*.

4.1.1. *Placing v Above Another b, 1-Vertex.* Let us assume that b, 1-vertex v is the next vertex to be inserted into the drawing. According to the discussion above, we find its predecessor vertex u above which v is going to be placed. Sometimes, it happens that vertex u is a b, 1-vertex too. From an implementation standpoint, each b, 1-vertex "remembers" (by keeping a pointer) its closest ancestor vertex with at least two outgoing edges above which it is placed. We call this vertex the *cover box* of the b, 1-vertex. Let w be u's cover box when v is about to be inserted. We distinguish two cases:

1. Vertex u is represented by a point. If vertex v is also a point, we simply insert it right above u as shown in Figure 5(a). If v is a box, we move column c_u containing u to occupy the next available column of w's outgoing edges from right to left. Any columns of w's outgoing edges holding only b, 1-vertices represented by points that are to the right of column c_u are now shifted to the left. This is depicted in Figure 5(b) (before the move), and Figure 5(c) (after the move of column c_u and insertion of v).
2. Vertex u is represented by a box. If vertex v is itself a point, we simply insert it above u as shown in Figure 6(a). Otherwise (i.e., vertex v is a box), we have two further subcases:
   (a) Vertex v does not have any incoming edges from b, 1-vertices represented by boxes, placed in columns to the left of u's column, and having the same cover box. We move the column with box u to occupy the next available column with

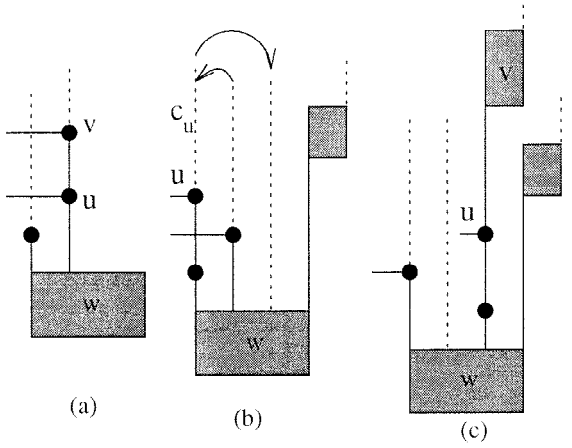

Fig. 5. Inserting v above its predecessor u when u is a point.

w's outgoing edges from right to left. Figure 6(b) shows the situation before v's insertion, and Figure 6(c) shows the result after v is inserted. Note that when we move u, the rows that box u is occupying in Figure 6(b) are removed and then re-opened to place u in Figure 6(c).

(b) Vertex v receives incoming edges from one or more b, 1-vertices represented by boxes, placed in columns to the left of u's column, and having the same cover box. Let u' be one of these vertices located at the column which is the closest to the column of vertex u. We move box u to occupy a position immediately to the right of the column holding vertex u'. We place v above u so that edges (u, v) and (u', v) do not have any bends (see Figure 6(d) for the situation after u has moved). Note that any b, 1-vertex that comes later and has to be placed above v, can do so without requiring any move of any column.

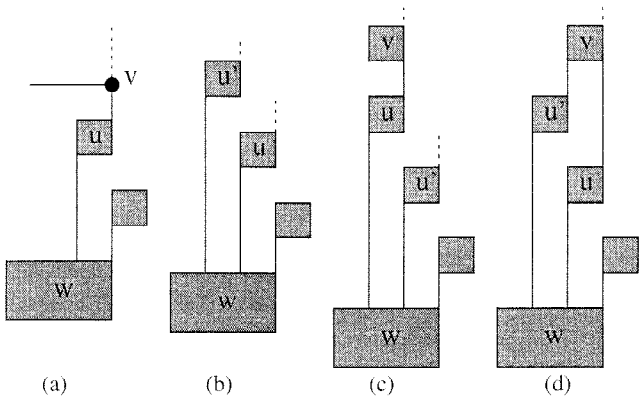

Fig. 6. Inserting v above its predecessor u when u is a box.

PROPOSITION 4.1. *Any b, 1-vertex placed directly above another b, 1-vertex u is a column-saver and has the same cover box as u.*

PROOF. Immediate from the discussion in this subsection. □

PROPOSITION 4.2. *There is at most one column-taker among all b, 1-vertices with the same cover box.*

PROOF. From Proposition 4.1, we have that any b, 1-vertex placed above another b, 1-vertex is a column-saver. In the case we have a cover box with two or more outgoing edges, a b, 1-vertex represented by a box and attached along the rightmost outgoing edge of the cover box is a column-taker (see Figure 4(c)). All other b, 1-vertices of the cover box reuse existing columns (see Figure 3(d)). □

4.2. *Vertex v Has at Least Two Incoming and Two Outgoing Edges.* We locate the median incoming edge(s) of v and place v as described in the Simple Algorithm. More specifically, we find vertex u where v's (left) median incoming edge comes from, and we open up $outdeg(v)$ new columns immediately to the right of u. However, as we explain in the rest of this subsection, this is not always possible if u happens to be a vertex with only one outgoing edge. Consider the case where all the following hold (see Figure 7(a)):

1. Vertex u has only one outgoing edge,
2. vertex u is placed directly above another box w,
3. there is at least one other vertex u' having w as a cover box, and
4. vertex u' is placed to the right of u.

In this situation, we cannot open up $outdeg(v)$ new columns immediately to the right of u because that would distort (stretch) the box representing vertex w. Instead, we open up these new columns immediately to the right of the rightmost box that has w as a cover box and place v there (see Figure 7(b)). After we place v as shown in Figure 7(b), it is

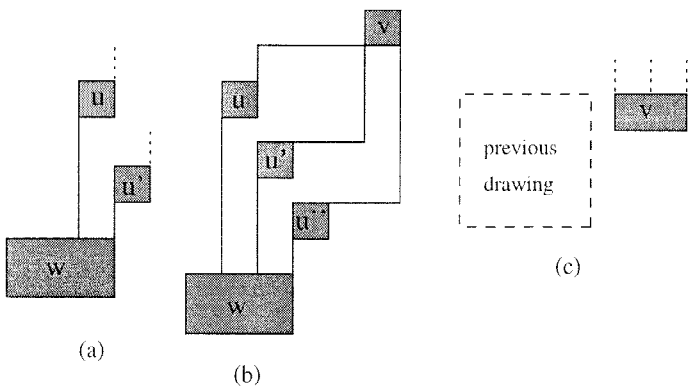

Fig. 7. (a) Box v cannot be inserted immediately to the right of its predecessor u, (b) routing some of v's incoming edges to v's bottom side, (c) placing source v.

possible that there are incoming edges of v which are supposed to attach to v's right side, but these edges are now along columns located to the left of box v. This happens because we moved box v away from the column carrying its (left) median incoming edge (u, v). If this is the case, we do the following:

1. Identify all the vertices of the current drawing which
   - have outgoing edges that enter vertex v, and
   - are located in between v's (left) median incoming edge and box v itself.
2. For each one of the vertices identified in the previous step, we draw an edge that exits the box of the vertex from the top right connector having horizontal direction and then bends once to enter box v through its bottom side.

This is all explained in Figure 7(b). Recall that u is the vertex where v's (left) median incoming edge comes from. We also assume that vertices u' and u'' are all the vertices identified in Step 1 of the above procedure. Note that in all cases, the vertices identified in this step are b, 1-vertices having the same cover box w. The number of incoming edges of these vertices is either three or an even number (see the previous section on the preprocessing phase), and therefore there is no other edge that exits the same top connector having horizontal direction. If some of these vertices are represented by points, the edge is drawn from the point to the right and then turned upward to enter v.

If an edge that enters box v from its bottom side comes from a column-taker b, 1-vertex, then we have this edge attach to one of the two corners of the bottom side of box v (see the way edge (u'', v) is routed in Figure 7(b)). On the other hand, if we have more edges entering v from its bottom side and box v's width is not sufficient to accommodate all of them, we make v as wide as necessary. The phenomenon of growing the width of the box representing a vertex in order to accommodate incoming edges entering through the bottom side of the box, is called *box inflation*. We also say that these incoming edges *cause box inflation*.

LEMMA 4.1. *Box inflation can only be caused by incoming edges that originate at column-saver b, 1-vertices.*

PROOF. Clearly, the vertices that cause box inflation are always b, 1-vertices with the same cover box. From Proposition 4.2 we know that at most one column-taker can be placed above a single cover box. The edge that comes from this column-taker can never cause box inflation when it is routed to the bottom side of another box, because it is forced to always attach to a corner of that box. Therefore, the statement of the lemma holds. □

Finally, when we attach the incoming edges of a box to a side (right or left) of a box, we try to avoid unnecessary crossings. Namely, the incoming edge (column) that is closer to the box should attach to connectors of lower y-coordinate of the side of the box.

4.3. *Vertex v Is a Source or a Sink.* The initial numbering of the vertices of the given graph yields one source and at least one sink. However, more sources/sinks may be

created after modifying the initial numbering. A source is inserted as a box of height one, and can be placed without opening up any new rows (except for vertex number 1 which is the original source). An example is depicted in Figure 7(c), where the box of source v is placed right off the end of the right margin of the current drawing.

Placing a sink follows the same rules as placing b, 1-vertices discussed above. In order to maximize row reuse, we can delay the placement of all sinks until the very end (in fact we can ensure this by assigning appropriate numbers to them). This is possible since sinks do not participate in groups/pairs. Placing sinks does not require opening up any new columns.

5. Grouping/Pairing Problematic Vertices. Another disadvantage of the Simple Algorithm is that vertices with one, two, or an odd number of three or more incoming edges contribute extra to the height of the drawings (see Theorem 2.1). We call these vertices the *problematic* vertices of the drawing phase. In Section 3 we discussed how problematic vertices can be assigned to groups/pairs in the preprocessing phase. When the time comes to insert a group/pair of vertices to the drawing, we create the boxes of the vertices in a way similar to the Simple Algorithm, and then we place all vertices involved in the group/pair in one step. In Sections 5.1–5.3 we show how different types of groups/pairs are placed in the drawing.

If there are problematic vertices that were not grouped/paired in the preprocessing phase, we try to group/pair them as we place them, that is during the time the drawing phase is going on. Thanks to the grouping/pairing of problematic vertices, we are able to produce area-efficient drawings occupying at most m columns and at most $m/2+c$ rows (c is a small constant). We assume that v is the next vertex considered by the drawing algorithm for insertion.

5.1. *Vertex v Has Only One Incoming Edge.* If the next vertex to be inserted v is a 1, a-vertex ($a > 1$), then we have the following cases, assuming that v was grouped/paired in the preprocessing phase:

- Vertex v is the first vertex of a chain of at least two 1, a-vertices. We open up a new line, and we place the boxes of all the vertices of the chain horizontally, as shown in Figure 8(a). Notice that the boxes of the chain are placed right off the end of the right or left margin of the current drawing so that one row is reused.
- Vertex v and another 1, a-vertex have the same predecessor and belong to the same pair. We open up one new row and we place the two boxes of the pair right off the end of the right and left margins of the current drawing, as shown in Figure 8(b). Hence, one row is reused.
- Vertex v and a b, 1-vertex ($b \geq 1$) belong to the same pair. Then, the b, 1-vertex is v's predecessor and v is placed along the row of the top side of the box of the b, 1-vertex so that no new row opens up. The situation is described in Figure 8(c) if b is even, and in Figure 8(d) if b is odd. Note that v is again placed right off the end of the right or left margin of the current drawing.

In case the 1, a-vertex v was not assigned to any pair or chain in the preprocessing phase, we try to place it without opening up any new rows. We have two options depending

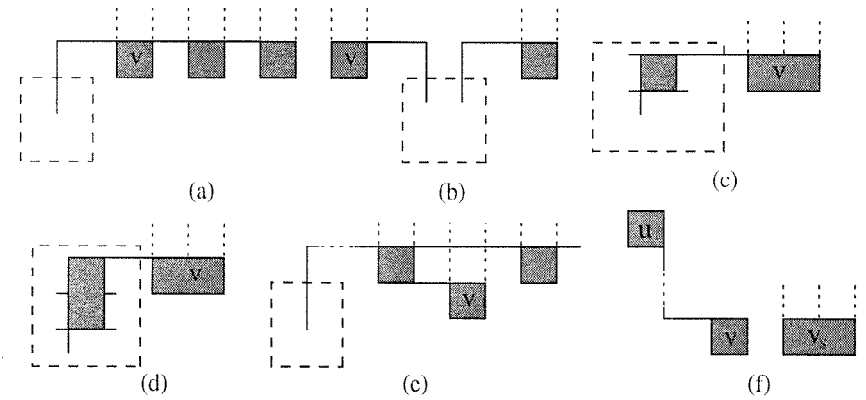

Fig. 8. Placing a group/pair containing a 1, a-vertex.

on the type of predecessor that vertex v has (see Lemma 3.1):

- If vertex v's predecessor is either a source or another 1, a-vertex belonging to some pair or chain, v is placed right next to the box of its predecessor; in this placement, two rows are reused. Figure 8(e) illustrates this when v's predecessor belongs to a chain.
- If v's predecessor, u, is a c, d-vertex where $c, d \geq 2$, then we look for the most recently placed source in the drawing, say v_s. We draw an edge from the bottom of u toward the row of v_s's top side and place v next to v_s without opening any new rows, as shown in Figure 8(f). Note that such a placement is always possible since there is no edge between vertex v and source v_s.

LEMMA 5.1. *Let $m(1, a)$ be the total number of incoming edges of a group/pair in which a 1, a-vertex participates. Inserting this group/pair to the drawing requires opening up at most $\lfloor m(1, a)/2 \rfloor$ new rows.*

PROOF. If there are only 1, a-vertices in the group/pair, then only one new row opens up in the drawing, as discussed above. Since such a group/pair has at least two 1, a-vertices, the statement of the lemma clearly holds. If a b, 1-vertex and a 1, a-vertex belong to the same pair, then we have two cases:

- If b is even then we open up $b/2$ new rows for the b, 1-vertex and no new rows for the 1, a-vertex. Clearly, the number of rows we open up for the pair is $b/2 = \lfloor (b+1)/2 \rfloor = \lfloor m(1, a)/2 \rfloor$.
- If b is odd then we open up $\lceil b/2 \rceil$ new rows for the b, 1-vertex and no new rows for the 1, a-vertex. Clearly, the number of rows we open up for the pair is $\lceil b/2 \rceil = (b+1)/2 = \lfloor (b+1)/2 \rfloor = \lfloor m(1, a)/2 \rfloor$. □

5.2. *Vertex v Has an Odd Number of Incoming Edges.* Let us assume now that the vertex to be inserted next, v, has an odd number of more than one incoming edges. Note that v also has at least two outgoing edges. If v belongs to a pair, we distinguish

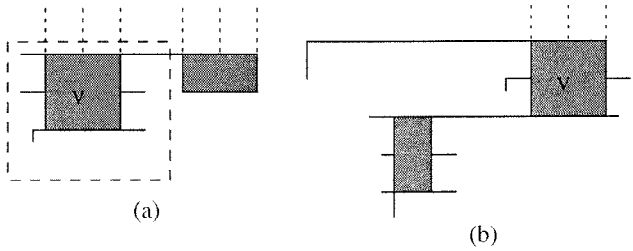

Fig. 9. Placing a vertex v with an odd number of incoming edges when it is paired with a (a) 1, a-vertex, (b) b, 1-vertex.

two cases. In both of these cases, vertex v is inserted in a way similar to the Simple Algorithm. If v's median incoming edge happens to be a b, 1-vertex, we may have to use the special placement for v described in Section 4.2. The two cases are as follows:

- Vertex v is the predecessor of a 1, a-vertex and the two vertices form a pair. First we insert vertex v. Then we attach the edge connecting the two vertices of the pair to the top right corner connector of v and place the 1, a-vertex outside the current drawing, as shown in Figure 9(a).
- Vertex v belongs to the same pair with a b, 1-vertex ($b \geq 1$) which is one of v's predecessors. We first insert the predecessor b, 1-vertex and then we place vertex v making sure that the top side of the box representing the b, 1-vertex and the bottom side of box v use the same row (see Figure 9(b)). We assume that the b, 1-vertex has been placed higher than the rest of v's predecessors; this can be achieved by delaying the placement of the b, 1-vertex until it is time to place v.

If v were not paired in the preprocessing phase, we try to pair it with the first vertex v' to be inserted after v which is not a source and has three or more outgoing edges. This pairing takes place during the drawing phase. If v' does not already belong to any group/pair and has one of the following properties, then the two vertices can be paired:

- v' and v are adjacent,
- v' and v are not adjacent and v' has an odd number of incoming edges greater than one,
- v' and v are not adjacent, v' has an even number of incoming edges, and the median incoming edge of v is not between the two median incoming edges of v'.

We place v and v' so that one row is reused. This is illustrated in Figure 10(a)–(c) for the three different cases of v' (see above). Figure 10 clearly shows that the top side of v and the bottom side of v' share the same row. This type of pairing is not always possible though. If we are unable to pair v with v', then we have two different options:

- If there were another vertex u with an odd number of incoming edges that we were unable to pair in the past, then we pair v with u. Vertex u has already been placed, so we now place vertex v so that one row is reused, as shown in Figure 11(a). More specifically, the bottom side of v and the top side of u use the same row. Notice that

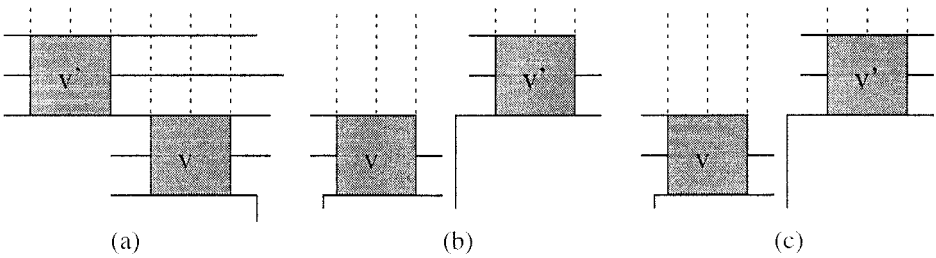

Fig. 10. Pairing vertex v with the vertex following v so that one row is reused.

v is placed at a higher position than u (we open up the appropriate number of rows there), and it is possible that some of v's incoming edges are attached to the bottom side of their corresponding boxes.

This type of vertex insertion would not have been possible if at least one of v's incoming edges were coming from a b, 1-vertex. This is because the outgoing edge of any b, 1-vertex cannot leave the bottom side of the box of the b, 1-vertex going down without crossing another box. However, we never have such a case here since no predecessor of v is a b, 1-vertex (if it were, v would have been paired with it).

- If there is no unpaired vertex u with an odd number of incoming edges inserted to the drawing before v, we simply place v in the drawing as described in the Simple Algorithm (see Figure 11(b)).

LEMMA 5.2. *Let m_{odd} be the total number of incoming edges of a pair that contains a vertex with an odd number of more than one incoming and at least two outgoing edges. Inserting this pair to the drawing requires opening up at most $\lfloor m_{\text{odd}}/2 \rfloor$ new rows.*

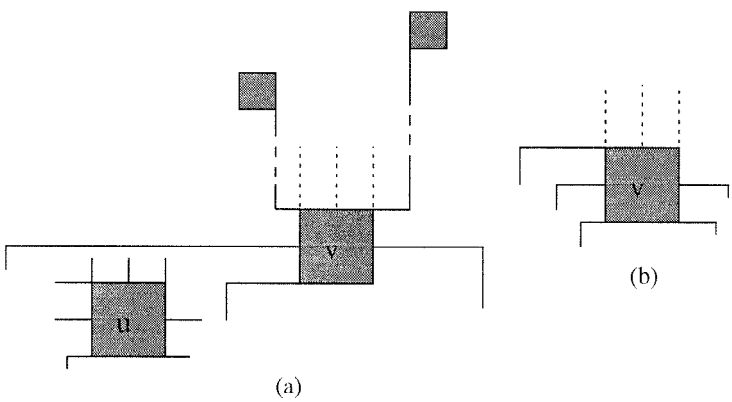

Fig. 11. (a) Pairing v with an existing unpaired vertex u, (b) placing unpaired vertex v.

PROOF. Let v be a vertex with m_v incoming edges ($m_v > 1$ is odd) and at least two outgoing edges. If v were paired in the preprocessing phase, we distinguish two cases:

- Vertex v belongs to the same pair with a 1, a-vertex. We open up $\lceil m_v/2 \rceil$ new rows for vertex v and no rows for the 1, a-vertex. Clearly, the number of rows we open up for the pair is $\lceil m_v/2 \rceil = \lfloor (m_v + 1)/2 \rfloor = \lfloor m_{\text{odd}}/2 \rfloor$.
- Vertex v belongs to the same pair with a b, 1-vertex (which has b incoming edges). First we open up at most $\lceil b/2 \rceil$ new rows for the b, 1-vertex. Then we place vertex v so that it reuses one row (see Figure 9(b)). Because of the row reuse, only $\lfloor m_v/2 \rfloor$ new rows are required in order to place the box of vertex v. Clearly, the number of rows we open up for the pair is at most $\lceil b/2 \rceil + \lfloor m_v/2 \rfloor$. If b is odd, then the total is at most

$$\left\lfloor \frac{b+1}{2} \right\rfloor + \left\lfloor \frac{m_v}{2} \right\rfloor \leq \left\lfloor \frac{b+m_v+1}{2} \right\rfloor = \left\lceil \frac{b+m_v}{2} \right\rceil = \frac{m_{\text{odd}}}{2},$$

  since $(b + m_v)$ is even. If b is even, then the total is at most

$$\frac{b}{2} + \left\lfloor \frac{m_v}{2} \right\rfloor = \left\lfloor \frac{b+m_v}{2} \right\rfloor = \left\lfloor \frac{m_{\text{odd}}}{2} \right\rfloor.$$

Let us now assume that vertex v is paired during the drawing phase. In this case, v forms a pair with the immediately following vertex v' (see Figure 10) which has $m_{v'}$ incoming edges. Since one row is reused, we open up $\lceil m_v/2 \rceil$ new rows to place v and $\lceil m_{v'}/2 \rceil - 1$ new rows to place v'. For the pair, at most $\lfloor (m_v + m_{v'})/2 \rfloor = \lfloor m_{\text{odd}}/2 \rfloor$ rows are required no matter whether $m_{v'}$ is odd or even. For the situation of Figure 11(a) where vertex v is paired with a previously inserted and unpaired vertex u, we can argue in a similar way to show that the number of rows for the pair is at most $\lfloor m_{\text{odd}}/2 \rfloor$. □

LEMMA 5.3. *There may be at most one vertex v with an odd number of more than one incoming and at least two outgoing edges which is not paired in the end of the drawing phase.*

PROOF. Let us assume that there is a vertex v with m_v incoming edges ($m_v > 1$ is odd) and at least two outgoing edges, which was inserted to the drawing by itself (i.e., not as a member of a pair). There are three possibilities for any other vertex w with an odd number of more than one incoming and at least two outgoing edges which is inserted to the drawing after vertex v:

1. Vertex w was already paired in the preprocessing phase.
2. Vertex w is paired with vertex $w' \neq v$ in the drawing phase.
3. Vertex w is paired with vertex v in the drawing phase.

In cases 1 and 2, vertex v remains unpaired after w is inserted to the drawing. In case 3 (and after w is inserted), there is no unpaired vertex with an odd number of more than one incoming and at least two outgoing edges any more. Therefore, at any time during the drawing phase as well as in the end of the drawing phase, the statement of the lemma holds. The unique unpaired vertex v opens up $\lceil m_v/2 \rceil$ new rows when it is placed in the drawing. □

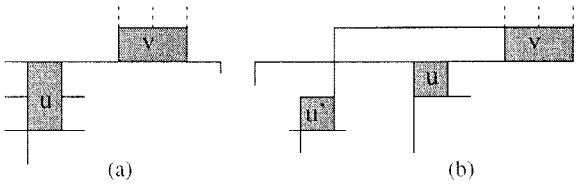

Fig. 12. Placing a 2, a-vertex when it is grouped/paired with its predecessor(s).

5.3. *Vertex v Has Two Incoming Edges.* If vertex v has two incoming edges and belongs to a group/pair, we distinguish the following cases:

- Vertex v is paired with one of its predecessors which is a b, 1-vertex (b even), say u. We place vertices u and v one after the other so that the bottom side of v and the top side of u share the same row (see Figure 12(a)).
- Vertex v is grouped with both its predecessors, say u and u'. In Figure 12(b), we show how we can reuse rows and place u, u', v opening up only four new rows; in this case, both u and u' are 3,1-vertices. Note that this placement also saves one bend from edge (u, v). Similar row reuse can take place when at least one of u and u' is a 1,1-vertex.

Note that in both of the above cases, placing the box of vertex v opens up only one new row in the drawing. Before we describe how to insert a 2, a-vertex v which does not belong to any group/pair, we give some definitions. A group of 2, a-vertices which have already been placed is called *open*, if and only if all the following hold:

- The group is formed during the drawing phase.
- No member of the group belongs to any other group/pair.
- The box of each vertex of the group shares one row with the box of some other vertex of the same group.

Clearly, if there are r 2, a-vertices in such an open group, then the placement of their boxes requires a total of $r + 1$ rows in the drawing ($r \geq 1$). Figure 13(a) shows an example with an open group of three 2, a-vertices. An open group of 2, a-vertices *becomes closed* the moment when a new 2, a-vertex u is placed in the drawing so that u shares two rows with the set of boxes of the open group. A closed group of r 2, a-vertices requires r rows in the drawing. If the next vertex to be inserted into the drawing is a 2, a-vertex which was not grouped/paired in the preprocessing phase, we try to assign this vertex to a group/pair at insertion time. We have the following options:

- We consider the first vertex to be inserted after v which is not a source, say v'. If v' has an even number of more than two incoming edges, at least two outgoing edges, and the columns of v's incoming edges are not positioned between the columns of the left and right median incoming edges of v', then we pair v with v' and we place them as shown in Figure 13(b). Note that the top side of v and the bottom side of v' share the same row.
- If there exists an open group of 2, a-vertices, then we place v so that v shares one or two rows with the boxes of the open group. Note that v can always be placed so that

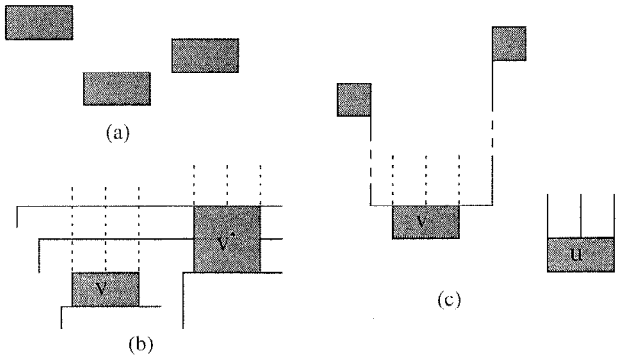

Fig. 13. (a) An open group of three 2, a-vertices placed in four rows, (b) placing v and v' reuses one row, (c) adding a 2, a-vertex v to an open group sharing one row.

one row is shared, even if this placement requires one or both incoming edges of v to attach to the bottom side of their corresponding boxes and extend in a downward direction. This is shown in Figure 13(c), where vertex u is a member of the open group. Such a placement is always possible since none of v's predecessors is a b, 1-vertex. If v shares one row with the rest of the open group, then the group remains open; if it shares two rows, then the group is now a closed group.

- If vertex v can neither be paired with the next incoming vertex nor join an open group, then we simply place v in a way similar to the Simple Algorithm. Note that v now starts a new open group, and opens up two new rows in the drawing.

PROPOSITION 5.1. *At any time during the drawing phase, there can be at most one open group of* 2, a*-vertices.*

PROOF. In order to have two or more open groups, we must have a situation where an open group already exists in the current drawing and a newcomer 2, a-vertex starts a new open group. However, this is a contradiction, since the newcomer 2, a-vertex has to join the existing open group (if no other grouping/pairing for this vertex is possible). □

LEMMA 5.4. *Let $V_{2,a}$ be the set of all* 2, a*-vertices placed by Algorithm BOX_ORTHOGONAL. The total number of new rows created by the insertion of all vertices in $V_{2,a}$ in the drawing is at most $|V_{2,a}| + 1$.*

PROOF. We have the following cases with respect to the grouping of the vertices of $V_{2,a}$ in the end of the drawing phase:

1. Each vertex v in $V_{2,a}$ was grouped/paired either in the preprocessing or the drawing phase with vertices not in $V_{2,a}$ (i.e., no open or closed groups of 2, a-vertices were formed). In either case, the insertion of vertex v reuses one row and opens up only one new row. So, all the vertices in $V_{2,a}$ collectively open up at most $|V_{2,a}|$ new rows.

2. There is at least one closed group of 2, a-vertices, but there is no open group of 2, a-vertices. We have seen that a closed group requires as many rows as the number of 2, a-vertices it contains. If we combine this with case 1 above, we have that all the vertices in $V_{2,a}$ collectively open up at most $|V_{2,a}|$ new rows.
3. There is one open group of 2, a-vertices. From Proposition 5.1 above, we have that there cannot be any other open group. We have also seen that the number of rows that the members of an open group require is at most one more than the number of 2, a-vertices it contains.

By combining the above cases, we obtain that all the vertices in $V_{2,a}$ collectively open up at most $|V_{2,a}| + 1$ new rows. □

6. The Drawing Algorithm and Its Analysis. After the preprocessing phase is complete, Algorithm BOX_ORTHOGONAL considers the vertices in the order of the final numbering, places them in the plane following the special instructions discussed above for groups, pairs, and individual vertices, and draws the edges. The vertices are represented as described above (point or box), depending on the number of their incoming edges. Here is an outline of our algorithm:

Algorithm BOX_ORTHOGONAL

Input: A graph G.
Output: An orthogonal drawing of G.

1. Compute an initial numbering of G having the properties specified in Section 3. If G is biconnected, the initial numbering is an st-numbering.
2. Modify the initial numbering to obtain the final numbering, as discussed in Section 3.
3. Apply the grouping/pairing procedure on the vertices of G.
4. Place vertex v_1 using a box of height one and width $outdeg(v_1) - 1$.
5. REPEAT
   (a) Consider the next vertex v_i according to G's final numbering.
   (b) If v_i has already been placed, then continue with the next vertex.
   (c) If v_i belongs to a group/pair, then place all members of this group/pair as described in the previous section for the specific kind of group/pair.
   (d) If v_i is a problematic vertex which does not belong to any group/pair, then try to group/pair it in the drawing phase and place it as discussed in the previous section.
   (e) If v_i is none of the above, then place it as described in the Simple Algorithm, while taking into account the special situations discussed in Sections 4.1–4.3.
6. UNTIL there are no vertices left.
7. End.

THEOREM 6.1. *Algorithm BOX_ORTHOGONAL produces an orthogonal drawing of*

any graph G with m edges in $O(m)$ time. Each vertex v with degree $deg(v) > 3$ is represented by a box whose perimeter is less than $2 \times deg(v)$. The produced orthogonal drawing has the following properties:

- *the width is at most $m - 1$,*
- *the height is at most $m/2 + 2$,*
- *the total number of bends is at most $m - 2$, and*
- *each edge has at most one bend.*

PROOF. The key to computing the running time of the algorithm is to find out how much time we spend when a vertex is inserted. In order to balance its incoming edges between the left and the right side of its box, we have to compute its (left and right) median incoming edge(s) first. In other words, we need to know the relative order among all the columns holding the incoming edges of the vertex to be inserted. This is not a trivial task since new columns can be inserted arbitrarily anywhere in the current drawing.

If we use the data structure proposed by Dietz and Sleator [7], we can answer column order queries for two columns in $O(1)$ time. This means that if we use a linear time median finding algorithm (e.g., see [4]), the (left and right) median incoming edge(s) can be computed in $O(indeg(v))$ time, where v is the vertex that is being inserted. Clearly, the total running time of the algorithm is $O(m)$. However, if we wish to attach v's incoming edges to the left and right side of v in a way that avoids as many crossings as possible, we need to know the total order of all the incoming edges of v. This order can be computed in the end collectively for all edges in $O(m)$ time using bucketsort. Then a simple scan of all the columns from left to right decides the correct connector where each edge will be attached (see Figure 2(a)).

As we saw in the previous sections, if a vertex v is represented by a box, then the height of the box is always at most $\lceil indeg(v)/2 \rceil$. The width of any box representing a vertex v is at most $outdeg(v)$, except in the case where the box is widened due to box inflation. In this case, the width of the box can grow up to $\lfloor indeg(v)/2 \rfloor$. In other words, the width of any box representing a vertex v is $\max\{outdeg(v), indeg(v)/2\}$. Clearly, the perimeter of the box is less than $2 \times deg(v)$, at all times.

In order to compute the total area taken by an orthogonal drawing produced by the algorithm, we have to count the total number of rows and columns in the drawing that are occupied by edges and/or boxes. Let us start with the rows. As we have seen, for each box we balance its incoming edges. This means that we attach half of them to the left and the other half to the right side of the box.

With the only exception of the problematic vertices, the number of rows that the algorithm opens up in the drawing in order to insert the box of any other vertex, is never more than half the number of the incoming edges of the vertex (see Simple Algorithm and Theorem 2.1). Recall that problematic vertices are 1, a-vertices, 2, a-vertices, and vertices with an odd number of incoming and at least two outgoing edges. These vertices are grouped/paired with other vertices. The result of this is to introduce row reuse in order to minimize the number of *extra* rows that the boxes of these vertices require. The following facts show how many extra rows of the final drawing must be attributed

to problematic vertices. Note that the word "extra" here means "in addition to half the number of the incoming edges."

1. From Lemma 5.1 and Section 5.1 we have that the insertion of 1, a-vertices to the drawing contributes no extra rows no matter whether these vertices are grouped/paired or not.
2. From Lemma 5.3 we have that at most one vertex with an odd number of more than one incoming and at least two outgoing edges contributes one extra row to the drawing. Any other such vertex participates in a pair which requires no extra rows (see Lemma 5.2).
3. From Lemma 5.4 we have that at most one unit has to be added to the total number of rows, and this unit comes from the 2, a-vertex which started the unique open group (if there is one in the end of the drawing phase).

Also keep in mind that the following hold:

1. Unpaired 1,1-vertices and sinks with one or three incoming edges can always be placed so that they share rows with other vertices of the same type or with other boxes. Also, 3,1-vertices that do not belong to any group/pair may be reduced to a point (they are boxes originally) so that each one of them occupies only one row.
2. Vertex v_1 (i.e., the first vertex to be inserted to the drawing) occupies two rows but it can always share one row with one 1, a-vertex that has v_1 as its predecessor. No other source opens up any new rows (see Section 4.3).

Hence, we have that the total number of rows in the final drawing is at most $m/2+1+1+1$ and the height is at most $m/2 + 3 - 1 = m/2 + 2$.

Let us now count the total number of columns in the final drawing. From Section 4.3, we have that sinks do not open up any new columns when they are inserted. Any other vertex v with $outdeg(v) \geq 1$ outgoing edges requires opening up at most $outdeg(v)$ new columns at v's insertion time. From this, we have that the total number of columns the algorithm opens up simply to insert vertices is at most m. However, recall the following facts:

1. From Section 4.1 we have that most b, 1-vertices can be placed so that they reuse already existing columns. Such b, 1-vertices do not open up any new columns and are known as column-savers. Let C_s be the total number of column-savers.
2. Some vertices may experience vertex inflation (see Section 4.2). Let C_i be the total number of columns in the final drawing that are attributed to vertex inflation.

Because of these two observations, the total number of columns in the final drawing is adjusted to $m + C_i - C_s$. From Lemma 4.1, we have that $C_i \leq C_s$, and therefore the width of the drawing is at most $m + C_i - C_s - 1 \leq m - 1$.

An important advantage of this algorithm is that it introduces very few bends. More specifically, every edge of the drawing has at most one bend, no matter whether it enters a box through its right, left, or bottom side. Also, all boxes/points representing sinks or vertices with only one outgoing edge have one incoming edge with no bends (see Figures 3 and 4). Clearly, there is at least one sink and at least one b, 1-vertex in the final drawing of any given graph with more than two vertices. Hence, the total number of bends of any drawing under Algorithm BOX_ORTHOGONAL is at most $m - 2$. □

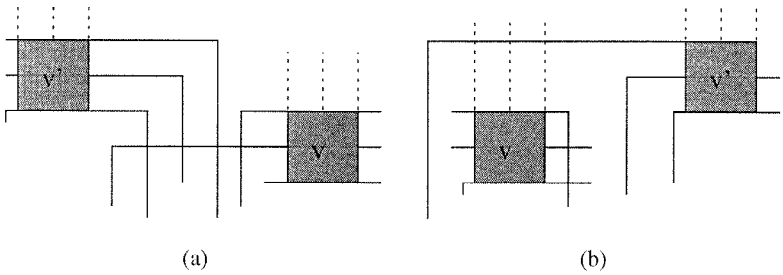

Fig. 14. Pairing two vertices in the drawing phase so that one row is shared.

7. Additional Pairing in the Drawing Phase. The pairing step that we describe here is not required in the drawing phase in order to prove the area and bends upper bounds for Algorithm BOX_ORTHOGONAL (see the previous section). However, if it can be done, it can improve the area of the resulting drawing. Let us consider a vertex v with an even number of more than two incoming edges and at least two outgoing edges, which was not grouped/paired in the preprocessing phase. Let us also assume that v is the next vertex to be inserted into the drawing by Algorithm BOX_ORTHOGONAL. We try to pair v with the first vertex to be inserted after v which is not a source, say v'. If v' already belongs to a group/pair, then we do not pair v and we simply insert it in a way similar to the Simple Algorithm. If, on the other hand, v' does not participate in any group/pair, we check whether

- the column of the (right) median incoming edge of v' is positioned to the left of the column of the left median incoming edge of v, or
- the column of the (left) median incoming edge of v' is positioned to the right of the column of the right median incoming edge of v.

In either of the above two cases we pair v with v' and we place the two vertices in the drawing so that one row is reused. This is illustrated in Figure 14(a) which depicts the first case, and in Figure 14(b) which depicts the second case. Observe that the top side of v and the bottom side of v' share the same row. If we have none of the above two cases, then no pairing is possible; in this case, we simply insert v and continue with the next vertex.

8. Further Optimizations. In practice, we expect the orthogonal drawings produced by Algorithm BOX_ORTHOGONAL to exhibit better bounds in terms of height, width, and total number of bends. More specifically, if more row reuse can take place when a new vertex is inserted, we take it. This may be done either during the drawing phase, or at a *refining phase* following the conclusion of the algorithm. For example, additional row reuse (and therefore smaller height for the drawing) results from pairing that we do in the drawing phase for vertices with an even number of more than two incoming and at least two outgoing edges (see the previous section).

As far as the width of the drawing is concerned, we expect it to be smaller than the upper bound for three reasons: First, if there are vertices for which some outgoing edge exits the vertex horizontally (and not vertically), the width of the vertex may be decreased

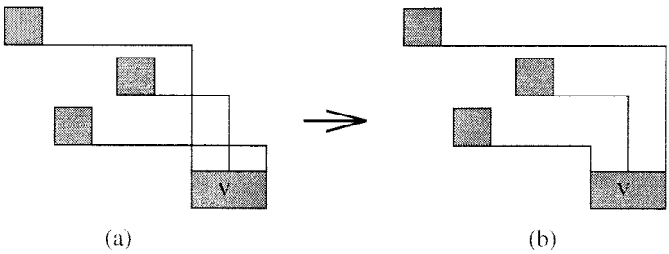

Fig. 15. Reshuffling the order of v's outgoing edges to eliminate crossings.

by one unit, as shown in Figures 8(a),(e), 9(a), and 10(a) (recall that vertex v originally occupies *outdeg*(v) columns). Second, when some of the incoming edges of a vertex v enter its bottom side (situation discussed in Section 4.2), we do not always have to widen the vertex by the number of these incoming edges, since there are usually available connectors in the bottom side of v. Third, we may shift some already placed boxes horizontally in order to reuse columns. This can be done at a final refining/compaction phase after the end of the algorithm.

The number of edges that have no bends in the drawing produced by Algorithm BOX_ORTHOGONAL may be more than just the number of vertices with zero or one outgoing edge. This is due to three reasons: First, for vertices that are adjacent and belong to the same group/pair, we draw the edge connecting the boxes of the two vertices horizontally without bends (see Figsures 8–10 and 12). Second, shifting boxes horizontally to reuse columns (see previous paragraph) may bring a box right above one of its predecessors; if this happens, the edge connecting the two boxes now has no bends. Third, in the refining phase, we can move 1,1-vertices on the bend of their outgoing edge (if there is such a bend), thus saving a bend for each 1,1-vertex.

Finally, another area where we can improve the quality of the drawing during the refining phase is the number of crossings. Let us consider the following scenario: Vertex v has three outgoing edges which enter three different vertices positioned on the same side (right or left) of v. This situation is depicted in Figure 15(a), where the three vertices are located to the left of v, and each one of v's outgoing edges crosses the other two. We can reshuffle the relative positions of the three outgoing edges so that there are no crossings (see Figure 15(b)).

In Figure 16 we show an example of a 12-vertex and 31-edge nonplanar graph G drawn by Algorithm BOX_ORTHOGONAL. Notice that G has one vertex with degree eight (vertex 9) and five vertices with degree six each (vertices 1, 2, 3, 7, and 11). In the drawing of Figure 16 we have already applied the refining phase which provided column reuse, thus saving more bends. The width of the final drawing is 17, the height is 14, and we have a total of 18 bends. Finally, the aspect ratio of the drawing is $17/14 = 1.21$ which is close to $4/3 = 1.33$, the standard screen aspect ratio.

9. Contributions and Future Work. We introduced an algorithm for drawing graphs with degree higher than four, based on grouping/pairing vertices of the graph for row

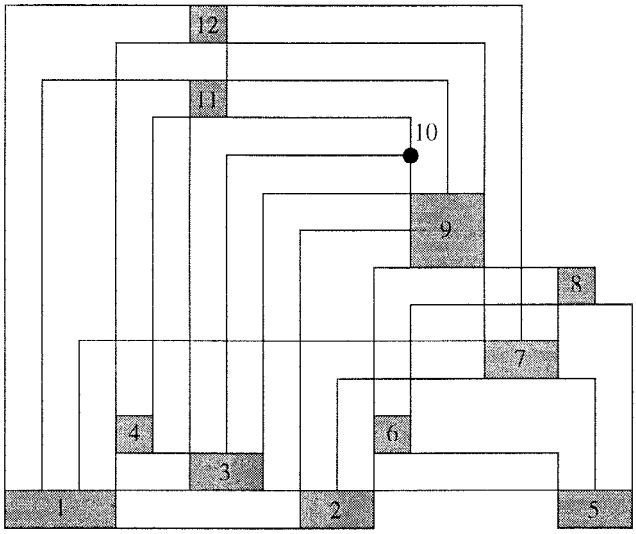

Fig. 16. An example drawing using Algorithm BOX_ORTHOGONAL.

reuse. It is interesting that vertices can form groups/pairs even at the time they are inserted. The algorithm is very efficient in terms of its running time. Also, the performance of the algorithm regarding the area and the number of bends (both total and per edge) of the drawing is better than any other known algorithm to address the same problem. The main advantages of our algorithm are the following:

- Vertices are represented by boxes of perimeter less than twice the degree of the vertex.
- It exhibits low area when compared with other algorithms.
- It provides the first upper bound on the area of any drawing of any graph.
- The width is larger than the height of the resulting orthogonal drawing (by approximately $m/2$) for better aspect ratio.
- The edges have either one bend or no bends; this increases the clarity of the drawing.
- Our algorithm is geared toward high degree vertices; the performance in terms of area increases when the graph has many vertices of high degree.
- The input graph does not have to be biconnected; our algorithm avoids the extra overhead of dealing with non-biconnected graphs.
- If the graph is not biconnected, the various components can be clearly identified in the drawing since all the vertices of the same component are drawn together.

It is open to see if we can find other representation schemes for graph vertices when their degree is higher than four. For the technique that we proposed, we would like to improve the upper bounds in the area and the number of bends. These, together with extending high degree orthogonal drawing to support interaction with the user, seem to be natural directions for future research.

Acknowledgment. We would like to thank an anonymous referee for suggestions that substantially improved the presentation of our ideas.

References

[1] T. Biedl, Drawing High-Degree Graphs with Small Grid-Size, Tech. Report RRR 37-96, RUTCOR, Rutgers University, November 1996.
[2] T. Biedl, Personal communication, December 1996.
[3] T. Biedl and G. Kant, A Better Heuristic for Orthogonal Graph Drawings, *Proc. 2nd Ann. European Symposium on Algorithms* (*ESA* '94), Lecture Notes in Computer Science, vol. 855, Springer-Verlag, Berlin, 1994, pp. 24–35.
[4] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
[5] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis, Algorithms for drawing graphs: an annotated bibliography, *Comput. Geom. Theory Appl.*, 4(5) (1994), 235–282. Also available via anonymous `ftp` from `ftp.cs.brown.edu`, `gdbiblio.tex.Z` and `gdbiblio.ps.Z` in `/pub/papers/compgeo`.
[6] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu, An Experimental Comparison of Three Graph Drawing Algorithms, *Proc. ACM Symp. on Computational Geometry*, 1995, pp. 306–315. The version of the paper with the four algorithms can be obtained from `http://www.cs.brown/people/rt`.
[7] P. F. Dietz and D. D. Sleator, Two algorithms for maintaining order in a list, *Proc. 19th Ann. ACM Symp. on Theory of Computing*, 1987, pp. 365–372.
[8] S. Even and G. Granot, Rectilinear Planar Drawings with Few Bends in Each Edge, Tech. Report 797, Comput. Science Dept., Technion, Israel Institute of Technology, 1994.
[9] S. Even and G. Granot, Grid Layouts of Block Diagrams - Bounding the Number of Bends in Each Connection, *Proc. DIMACS Workshop GD* '94, Lecture Notes in Computer Science, vol. 894, Springer-Verlag, Berlin, 1994, pp. 64–75.
[10] S. Even and R. E. Tarjan, Computing an st-numbering, *Theoret. Comput. Sci.*, 2 (1976), 339–344.
[11] U. Fößmeier and M. Kaufmann, Drawing High Degree Graphs with Low Bend Numbers, *Proc. Workshop GD* '95, Lecture Notes in Computer Science, vol. 1027, Springer-Verlag, Berlin, 1995, pp. 254–266.
[12] G. Kant, Drawing planar graphs using the canonical ordering, *Algorithmica*, 16(1) (1996), 4–32.
[13] A. Papakostas, Information Visualization: Orthogonal Drawings of Graphs, Ph.D. Thesis, Computer Science Dept., The University of Texas at Dallas, November 1996.
[14] A. Papakostas and I. G. Tollis, Algorithms for area-efficient orthogonal drawings, *Comput. Geom. Theory Appl.*, 9 (1998), 83–110.
[15] A. Papakostas and I. G. Tollis, A Pairing Technique for Area-Efficient Orthogonal Drawings, *Proc. Workshop GD* '96, Lecture Notes in Computer Science, vol. 1190, Springer-Verlag, Berlin, 1996, pp. 355–370.
[16] A. Papakostas and I. G. Tollis, Interactive orthogonal graph drawing, *IEEE Trans. Comput.*, 47(11) (1998), 1297–1309.
[17] P. Rosenstiehl and R. E. Tarjan, Rectilinear planar layouts and bipolar orientations of planar graphs, *Discrete Comput. Geom.*, 1 (1986), 343–353.
[18] M. Schäffter, Drawing graphs on rectangular grids, *Discrete Appl. Math.*, 63 (1995), 75–89.
[19] J. Storer, On minimal node-cost planar embeddings, *Networks*, 14 (1984), 181–212.
[20] R. Tamassia, On embedding a graph in the grid with the minimum number of bends, *SIAM J. Comput.*, 16 (1987), 421–444.
[21] R. Tamassia and I. G. Tollis, A unified approach to visibility representations of planar graphs, *Discrete Comput. Geom.*, 1 (1986), 321–341.
[22] R. Tamassia and I. Tollis, Planar grid embeddings in linear time, *IEEE Trans. Circuits and Systems*, 36 (1989), 1230–1234.