Grid Layouts of Block Diagrams – Bounding the Number of Bends in Each Connection (Extended Abstract)

S. Even* and G. Granot

Computer Science Department
Technion, Israel Inst. of Tech.
Haifa 32000, ISRAEL

even@cs.technion.ac.il gil@cs.technion.ac.il

Abstract. Consider an input data which specifies rectangular modules and connections between them; this is a graph. The size of the modules and the placements of the terminals on them is given as part of the input. We produce a block diagram, conforming to the input. The block diagram is on the rectilinear grid, and each edge (connection between modules) has few bends.
For planar input, a linear-time algorithm is described to construct a planar drawing with at most 6 bends in each self-loop and at most 4 bends in any other edge. The external face of the drawing may be specified by the user. We show a planar input with no self-loops which has no drawing with at most 3 bends in every edge and another planar input which has no drawing with at most 5 bends in every self-loop.
A linear-time algorithm is described to construct a nonplanar drawing of any input, with at most 4 bends in each edge. We show inputs that have no drawing with at most 3 bends in every edge.

1 Introduction

Layouts of graphs on the rectilinear grid have been studied because of their applications in VLSI planning and automatic graph and data drawing. A lot of work was done on rectilinear planar drawings with vertices drawn as points; see, for example, [Shi76, MO85, Tam87, YMS91, EG94, BK94]. Not much work has been done on drawings in which vertices are more complex structures, in spite of the fact that this approach is natural in VLSI, where the vertices can represent predefined modules.

Our interest is in drawings which have a small constant upper bound on the number of bends which occur in any edge. This is important where the drawing realizes a circuit, and a bend may cause delay or signal loss.

* Supported by the Fund for the Promotion of Research at the Technion.

Often, certain attributes of the block diagram should be preserved. Therefore, more data is specified in the input than just a graph.

The input , called *a module description*, consists of:

1. A set of rectangular modules of integral dimensions. Each module has a set of terminals which are designated points on the module's perimeter, at integral places.
2. A set of connections. Each connection has two designated terminals which are its end-points and each terminal is an end-point of exactly one connection.

The drawing of the block diagram on the rectilinear planar grid consists of:

1. Placing each module on the grid, adjusted to the grid lines. The placed modules are nonoverlapping and do not touch each other.
2. Drawing each connection as a path on the grid, so that:
   - No two such paths share a grid-edge.
   - If two paths share a point, they cross each other at that point.
   - No path shares any point with any module, except its end-points.

Such a drawing is referred to as a *grid-layout* of the module description. If no two path intersect, the grid-layout is called *planar*. (See Fig. 1).

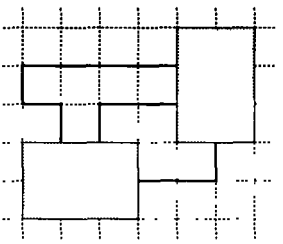

Fig. 1. A 3-bend planar grid-layout which is not 2-bend

An *edge (incidence) ordering* of a graph is a specification, for each vertex, of a cyclic order of the endpoints of its incident edges. A graph with an edge-ordering is called an *ordered graph*. A drawing of an ordered graph in the plane is *proper*, if the edge-ordering of the ordered graph corresponds to the clockwise arrangement of the endpoints around the vertices in the drawing. An ordered graph is *planar*, if it has a proper planar drawing. An ordered graph determines the faces of any proper drawing, but doesn't determine which one will be external (infinite).

Clearly, a module description corresponds to an ordered graph. Each module is represented by a vertex, and each connection is represented by an edge of the graph. Note, that the ordered graph is not necessarily simple; it may have parallel edges and self-loops. If this ordered graph is planar, we say that the module

description is *planar*. A module description has a plane grid-layout (without flipping modules) if and only if it is planar.

When referring to a module description, we will frequently use terms belonging to the corresponding ordered graph. That is, we will say edge and degree of a vertex instead of connection and number of terminals on a module. We denote the ordered graph corresponding to the input module description as $G(V, E)$.

There are several natural tasks concerning grid-layout. They can be classified according to the following issues:

- Do we insist on a planar grid-layout? If we do, the corresponding ordered graph must be planar.
- Do we allow to flip modules? Of course, flipping a module reverses the edge-ordering in the corresponding vertex.

There are known linear-time algorithms to check if a given ordered graph is planar [Zak93, Pin83, Ami87]. Also, in case flipping is allowed, there is a known linear-time algorithm to check if a flipping exists for which the resulting ordered graph is planar, and if so, to specify such a flipping [Pin83]. In this extended abstract we assume that we are given a planar module description.

We allow rotation of the modules since it does not effect planarity.

A *bend* is a point of a path in a grid-layout in which the path makes a 90° turn; we say that the corresponding edge of the ordered graph has a bend. A grid-layout is called *k-bend* if no edge (path) has more than k bends.

We present two linear-time algorithms:

1. For any given planar module description, as well as a specified external face, a 6-bend planar grid-layout is constructed. Each edge which is not a self-loop has at most 4 bends. This result is the best possible in the following sense: There are flip-planar module descriptions with self-loops, for which none of their planar module descriptions (after flippings) has a 5-bend planar grid-layout. Also, there are flip-planar module descriptions without self-loops, for which none of their planar module descriptions has a 3-bend planar grid-layout.
2. For any module description a 4-bend grid-layout is constructed. (It may be nonplanar.) There is no need to rotate or flip any module. Again, this result is the best possible in the following sense: There are module descriptions for which no 3-bend grid-layout exists, even if rotation and flipping is allowed.

We use the notation $u \overset{e}{-} v$ for an edge e which connects vertices u and v and $u \overset{e}{\rightarrow} v$ for a directed edge e from u to v.

An *st bipolar orientation* of a graph G is a directed acyclic graph whose underlying undirected graph is G; it has one source s, one sink t and an edge $s \rightarrow t$. An *st* bipolar orientation of a nonseparable graph[2] with no self-loops can be found in linear time. This is described in [ET76], where an equivalent notion of *st numbering* is used.

[2] A connected graph is called nonseparable if it has no separating (cut) vertex.

A *visibility representation* of a graph is a planar drawing of the graph on the rectilinear grid, where vertices are drawn as horizontal lines or points and edges are drawn as vertical lines. We assume that the ends of the drawing of a vertex are at the endpoints of its leftmost and rightmost incident edges. Two very similar linear-time algorithms for finding visibility representations of graphs are described in [RT86, TT86].

2 Planar Grid-Layouts – The Case of No Self-Loops

We assume G is connected, for otherwise we can draw a planar grid-layout of each connected component separately. The connected components of a graph can be found in linear time. Thus, we show that any graph can be drawn in linear time by showing that connected graphs can be drawn in linear time. In this section we assume G has no self-loops.

If the graph is nonseparable, we find an st bipolar orientation of it [ET76]. Otherwise we first add edges to the ordered graph which make it nonseparable without impairing its planarity. This is done without changing the order around vertices of the edges of the input ordered graph; it can be done by adding edges inside faces in which a (cut) vertex appears more than once, or as described in [EG94]. Next, we find an st bipolar orientation of the new ordered graph. The added edges are called *auxiliary* and will not be drawn in the grid-layout. As in [ET76], one may choose any edge to be $s \rightarrow t$. We choose an edge such that when it is drawn vertically, with s at the bottom, the face we want to be external is on its left hand side. This ensures that the specified face will be external. Henceforth we consider the ordered graph to be directed, the edges having the directions of the st bipolar orientation.

Next, for each vertex, we determine the rotation of its module's drawing and divide its incident edges to two subsets; one containing edges that will be drawn to the right of the vertex and the other containing edges that will be drawn to the left of the vertex.

First, let us describe how this is done for vertices which are neither s nor t.

Lemma 1. *[RT86, TT86] In an st bipolar orientation of a planar ordered graph, for every vertex, its incoming (outgoing) edges appear consecutively in the edge ordering.*

We need to know which edges are the leftmost and rightmost edges of the incoming (outgoing) edges incident to a vertex. Lemma 1 implies that there is no ambiguity in their indication for vertices other than s and t.

Thus, the perimeter of the module can be divided to two paths; one incident only to incoming edges and the other incident only to outgoing edges. The module can be rotated such that the path of outgoing edges has a section on the top of the module, and the path of incoming edges has a section at the bottom of the module. Figure 2 demonstrates how this can be done. More than one appropriate rotation may be possible.

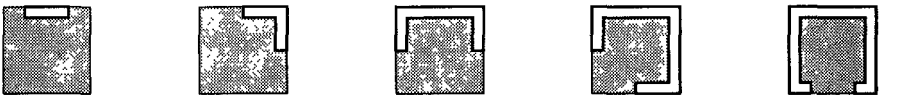

Fig. 2. Rotation of a vertex which is not a source nor a sink.
The marked path on the perimeter of the module is the path that contains only outgoing edges and should have a section on the top of the module, etc.

We choose a point on the path incident to outgoing edges which is on the top of the module, and a point on the path incident to incoming edges which is at the bottom of the module. Such points exist due to the way the rotation has been selected. These two points divide the perimeter of the module to two new paths. The edges incident to the new path which contains the right hand side of the module, are the edges that will be drawn to the right of the vertex. The edges incident to the other new path, which contains the left hand side of the module, are the edges that will be drawn to the left of the vertex. Although the auxiliary edges are marked as drawn to the right or to the left of the module, they will not be actually drawn.

Now we deal with s and t. First we consider the edge $s \rightarrow t$ to be the leftmost edge of s and t. Since s (t) has only outgoing (incoming) edges, we choose a point on the perimeter of the module, between the leftmost and rightmost incident edges, and consider it to be the path incident to incoming (outgoing) edges. The rotation of the module and the division of the incident edges is done as with the other vertices. Note, that this ensures that $s \rightarrow t$ is the leftmost edge.

Next, for every vertex we determine how much space the drawing of the vertex needs. We calculate for every vertex a quantity called the *drawn height* of the vertex. The drawn height is composed of the height of the module plus added space below and above the module. The size of the added space above (below) the module is the maximum of the number of edges incident to the top (bottom) of the module that are drawn to its left and the number of edges incident to the top (bottom) of the module that are drawn to its right.

Next, we want to construct a visibility representation so that each vertex will have the space needed to draw its module and its connections. This requires that if we stretch each vertex of the visibility representation downwards to be in the height of its drawn height, the vertices of the visibility representation will remain nonoverlapping. We also require that in the drawing of each vertex v, in the visibility representation, there will be a section having the width of the module of v, with the following properties. All edges to be drawn to the right (left) of v are incident to v at points to the right (left) of this section, therefore no edge is incident to v at this section. The $s \rightarrow t$ edge must be the leftmost edge of the visibility representation; this causes the correct face to be external.

Such a visibility representation can be found using the algorithm of [DTT92]. This algorithm finds a visibility representation, for an st bipolar orientation, with certain specified directed paths of the st bipolar orientation drawn as straight

vertical lines. That is the edges of each such path are vertically aligned. The given paths are required not to cross each other in a proper planar drawing of the ordered graph. In this algorithm, the coordinates of the visibility representation are found by two topological orderings. A *topological ordering* for a directed acyclic graph with non-negative edge weights is an assignment of a positive integer to each vertex; such that for each edge, the number assigned to its destination minus the number assigned to its origin is greater than or equal to the weight of the edge. One ordering is done on the vertices of the original graph and the other is done on the vertices of a modification of the dual graph. The weights represent the minimal space below a vertex in the first topological ordering and represent the minimal space to the right of an edge in the second topological ordering. Following, we describe how to determine the weights and how to specify the paths to be aligned.

For a vertex that has incoming (outgoing) edges drawn to the left of the vertex, we will call the rightmost incoming (outgoing) edge, which is drawn to the left of the vertex, *aligned* to the vertex. If a vertex has incoming (outgoing) edges, but they are all drawn to the right of the vertex, we duplicate the leftmost incoming (outgoing) edge. The duplicate edge left of the edge, will be considered drawn to the left of the vertex, and will be called aligned to the vertex. The added edge is auxiliary and will not appear in the grid-layout.

For every vertex which is neither the source nor the sink of the *st* bipolar orientation, its incoming aligned edge and outgoing aligned edge will be put in a path to be aligned in the visibility representation. This path may continue farther if any of these two edges are aligned to their other end vertices. The paths created do not cross each other in a proper planar drawing of the ordered graph, since by definition, there can be at most one path which passes through a vertex.

The weight of an edge in the topological ordering of the original graph (including the auxiliary edges) is the drawn height, of its target vertex, plus 1. The weight of an edge in the topological ordering of the modified dual graph is as follows. If the edge is aligned to one of its end vertices, its weight is the vertex's module width plus 2. If the edge is aligned to both of its end vertices its weight will have the greater value of the two module widths plus 2. Otherwise the edge will have a weight equal to 1.

The specification of the input to the visibility representation algorithm ensures us that the visibility representation produced will be as required: There will be a section of the line, representing the vertex, with length greater than or equal to the module width plus 2, with no incident edges, and it will be to the right of the edges aligned to the vertex.

Finally, we translate the visibility representation to the grid-layout by changing the drawing of each vertex. All changes in the drawing of a vertex are made in the rectangle below the aforementioned section. See Fig. 3.

For every edge incident to the right and left sides of the module, a horizontal line segment is drawn, connecting its terminal with the vertical line representing the edge in the visibility representation. For every edge incident to the bottom

and top of the module, two line segments are used for the connection. A vertical segment originating at the terminal, and a horizontal segment. The horizontal line segmentss of the edges incident to the top (bottom) of the module that are drawn to the right (left) of the module are stowed next to the module. See Fig. 3.

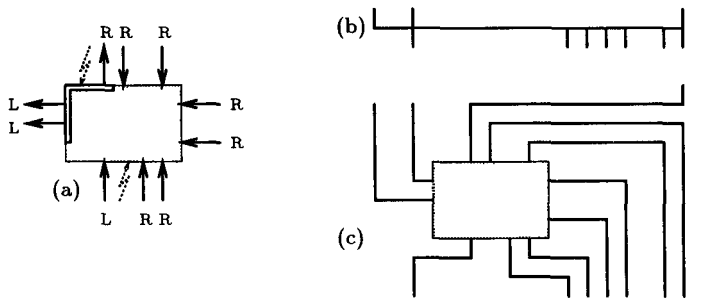

Fig. 3. The steps of drawing a module in the algorithm
(a) - After the edges are directed, the rotation of the module is determined and the incident edges are divided to edges drawn to the left and to the right of the module; this is done by twice dividing the perimeter of the module to two paths. (b) - The drawing of the vertex in the visibility representation. (c) - The final drawing of the module in the grid-layout.

This produces the grid-layout. The connection of an edge to a module introduces one bend when incident to the right or left side of the module, and two bends when incident to the top or bottom of the module. An edge has at most two bends at the top, at most two bends at the bottom, and no bends in between. Thus, an edge has no more than four bends in the grid-layout.

Theorem 2. *A linear-time algorithm exists that given a planar module description with no self-loops finds a corresponding 4-bend planar grid-layout.*

3 Planar Grid-Layouts – The Case With Self-Loops

Self loops impose a problem. Following we describe how we deal with them.

First, we put a new vertex in the middle of every self-loop; i.e. we replace every self-loop by two edges and a new vertex. This is shown in Fig. 4(b). The newly created vertices are called *dummy* vertices, and the vertices of the original graph are called *normal.* The dummy vertices will be drawn differently from normal vertices; they will not be replaced by modules. Other than this, the graph is drawn in the same way as described in the previous section.

After introducing the dummy vertices, the resulting graph is separable, and we use the techniques described in the previous section to make it nonseparable. For each normal vertex, we determine the rotation of the module and the assignment to the right and to the left of the incident edges, as in Sect. 2.

Next, we find a visibility representation of the graph with enough space in it to accommodate the drawings of the vertices. The weights of the edges in the topological ordering of the original graph are determined as before, when the drawn height of dummy vertices is zero. The weights of the edges in the topological ordering of the modified dual graph are determined as before, when no edge is considered aligned to a dummy vertex.

When translating the visibility representation, just the drawing of the normal vertices is changed. The horizontal line segment, representing a dummy vertex, together with the drawing of the two edges incident to it, completes the drawing of the self-loop in the grid-layout.

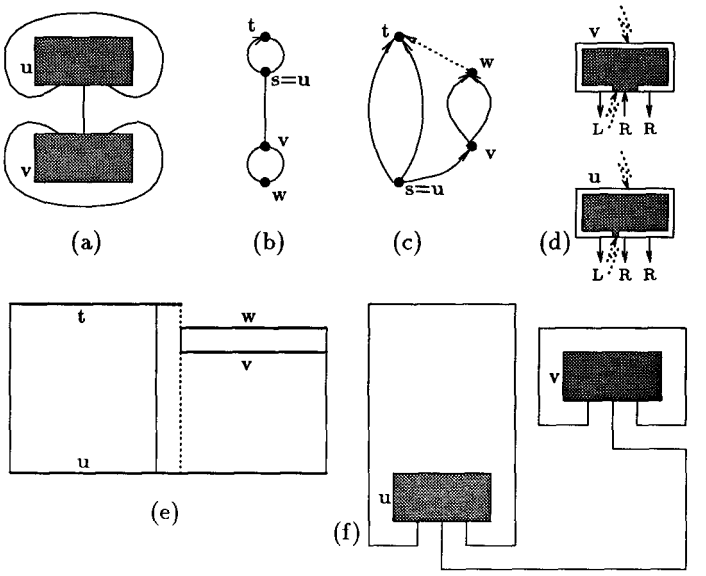

Fig. 4. Drawing a graph that has self-loops.
The steps of finding the drawing are shown: (a) The input. (b) The input graph after changing the self-loops, the choice of $s \rightarrow t$ and of the external face. (c) The *st* bipolar orientation, the edges are drawn upward according to their direction. (d) The rotation of the modules and the division of edges to be drawn to the left and to the right of the module. (e) The visibility representation. (f) The final grid-layout.

This yields a grid-layout of a module description with self-loops. A demonstration run of the algorithm is described in Fig. 4. Edges which are not self-loops have, as before, at most four bends. Self loops are composed of two edges in the visibility representation. Each has at most two bends near the vertex the self-loop is adjacent to, and exactly two bends at the drawing of the dummy vertex in the visibility representation. Thus, self-loops have a total of at most six bends.

Theorem 3. *A linear-time algorithm exists that given a planar module descrip-*

tion, finds a corresponding 6-bend planar grid-layout in which every edge that is not a self-loop has at most 4 bends.

The coordinates of the drawing are determined by the two topological orderings. Therefore, the maximum coordinate is smaller than the sum of the edge weights. We denote the sum of the widths of the modules, in their chosen rotation, by W, and the sum of the heights of the modules by H. Thus, the width and height of the produced drawing are $W+O(|E|)$ and $H+O(|E|)$, respectively.

We present examples of planar module descriptions which show that our bounds are tight. (See Fig. 5). These examples do not depend on module flippings or on the choice of the external face.

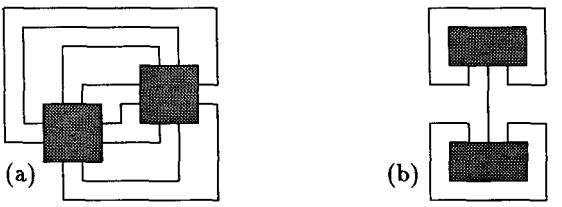

Fig. 5. Planar module descriptions that do not draw well.
(a) Requires at least 4 bends for some edge.
(b) Requires at least 6 bends for some self-loop.

4 An Application

Our drawing algorithm can draw the grid-layout with any specified external face. This is important since it enables us to produce a planar grid-layouts which include external connections.

This is demonstrated in the following example which is an application of the drawing algorithm.

Let us consider the construction of a planar grid-layout intended to be the internal layout of a big module. The input is a description of the big module, consisting of its size, the placement of its terminals, and the module description of its inside. That is, in addition to the (inner) module description, its external face is specified as well as the following constraints on external terminals: For each external terminal, a connection to a terminal on the perimeter of an (inner) module is specified. Notice that the additional specification of the terminals of the big module may render the input nonplanar. Unfortunately the problem of finding if such a planar grid-layout exists is hard. Consider an input with no edges. The problem becomes the 2D bin-packing problem which is NP-hard. In certain cases the following approach solves the problem: Find a grid-layout of the inside of the module, and if it is not too big, connect it to the perimeter of the big module. This can be done as follows.

Add four special vertices to the inner module description, which represent the sides of the big module's perimeter. The special modules are connected to the original module description by edges representing the terminals of the big module, and the special modules are also connected to each other, see Fig. 6(a). Also, the special modules will be drawn in the same rotation as they have in Fig. 6(a). This is possible since these rotations comply with the rules of selecting a rotation, and we may select any such compliant rotation. A grid-layout of the new module description is constructed using our algorithm. Also, the $s \rightarrow t$ is chosen as in Fig. 6(a).

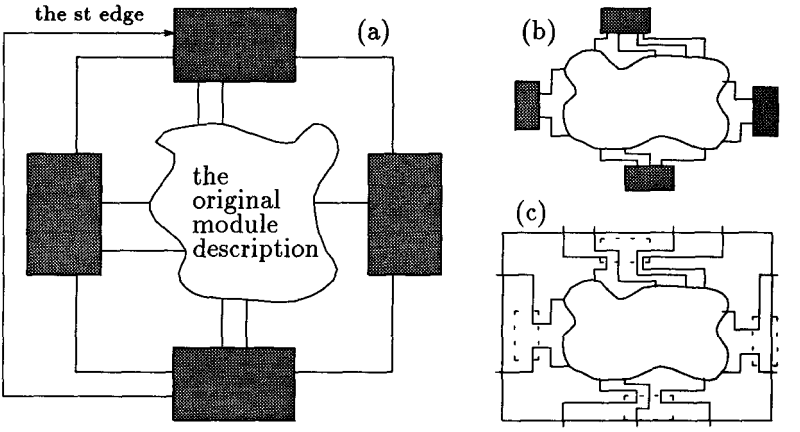

Fig. 6. An example of an application - planar layout of a big module.
(a) The new module description. (b) The grid-layout produced, after removing edges between the special modules. (c) The final layout of the big module.

If the drawing produced is bigger than the size given for the big module, we fail. If not, erase from the grid-layout the edges that connect the special modules, see Fig. 6(b). Remove the drawings of the special modules, and change the drawing of the remaining edges connected to them, so they connect to the terminals of the big module on its perimeter. This is demonstrated in Fig. 6(c).

5 Nonplanar Grid-Layouts

In this section we consider the nonplanar versions of the problems we have dealt with. We allow edges to cross one another in the grid-layout. The input module description is not assumed to be planar, and even if it is, the output grid-layout is not required to be planar.

We show how to find 4-bend (nonplanar) grid-layout of any module description. The flippings of the modules and the rotation in which they are drawn may be determined arbitrarily. The input to the algorithm is a module description,

and the rotation in which each module is to be drawn. A linear-time drawing algorithm is described in Fig. 7. The algorithm is demonstrated in Fig. 8. The algorithm works by first placing the modules diagonally, so that if we continue the line of any terminal, it will not hit any module. Next, we complete the drawing of the edges.

```
procedure NPGL(M);
{ M is a module description, with a rotation specified for each module. }
begin
position the modules in the drawings diagonally, such that
    the drawings of two modules do not share a row or a column ;
R := the smallest rectangle on the grid which contains all modules ;
for every edge e of M do
    begin
    continue the lines of the two terminals of e till they exit from R,
        and connect them by at most 3 lines around the boundary of R ;
    expand R to contain the drawing of e
    end
end;
```

Fig. 7. Construction of a 4-bend (nonplanar) grid-layout.

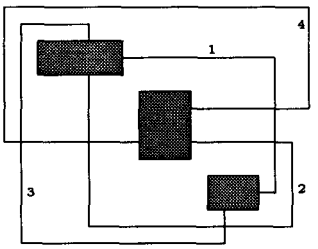

Fig. 8. Output of the nonplanar grid-layout algorithm.

Theorem 4. *Algorithm NPGL (see Fig. 7) finds in linear-time a 4-bend (nonplanar) grid-layout for any module description, with any specified rotations of the modules.*

The area of the grid-layouts has the same upper bound as for the planar grid-layouts.

Finally we mention two module descriptions which have no 3-bend grid-layout. The first module description has one module which has one self-loop. The terminals of the self-loop are on opposite sides of this module. The second module description has two modules and sixteen edges. Every side of one module is connected by one edge to every side of the other module.

References

[Ami87] A. Amihood. A direct linear-time planarity test for unflippable modules. *Intern. J. Computer Math.*, vol. 21, pp. 277–290, 1987.

[BK94] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. In *Proc. of the Second Annual European Symposium (ESA '94), Lecture Notes in Computer Science, Vol. 855*, pp. 24–35. Springer-Verlag, 1994.

[DTT92] G. Di Battista, R. Tamassia, and I. G. Tollis. Constrained visibility representations of graphs. *Information Processing Letters*, vol. 41, pp. 1–7, 1992.

[EG94] S. Even and G. Granot. Rectilinear planar drawings with few bends in each edge. Technical Report 797, Computer Science Department, Technion, Israel Inst. of Tech., 1994. can be retrieved by anonymous ftp from ftp.technion.ac.il at directory /pub/supported/cs/Tech_Reports/1994 as file TR797.ps.Z.

[ET76] S. Even and R. E. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, vol. 2, pp. 339–344, 1976.

[MO85] Z. Miller and J. B. Orlin. NP-completeness for minimizing maximum edge length in grid embeddings. *Journal of Algorithms*, vol. 6, pp. 10–16, 1985.

[Pin83] R. Y. Pinter. River routing: Methodology and analysis. In *third CALTECH conference on Very Large Scale Integration*, pp. 141–163. Computer Science Press, 1983.

[RT86] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete and Computational Geometry*, vol. 1 no. 4, pp. 343–353, 1986.

[Shi76] Y. Shiloach. *Linear and Planar Arrangements of Graphs.* PhD thesis, Department of Applied Mathematics, Weizmann Institute of Science, Rehovot Israel, 1976.

[Tam87] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Computing*, vol. 16 no. 3, pp. 421–444, 1987.

[TT86] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete and Computational Geometry*, vol. 1 no. 4, pp. 322–341, 1986.

[YMS91] L. Yanpei, A. Morgana, and B. Simeone. General theoretical results on rectilinear embedability of graphs. *Acta Mathematicae Applicatae Sinica Journal*, vol. 7 no. 2, pp. 187–192, 1991.

[Zak93] S. Zaks. An easy planarity test for unflippable modules. Technical Report 771, Computer Science Department, Technion, Israel Inst. of Tech., 1993.