Planar Drawings of Graphs on the Rectilinear Grid with Few Bends in Each Edge

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE

GILAD GRANOT
I want to thank Shimon Even for his helpful guidance. Shimon always knew to suggest a direction to work on, when I needed it. He helped me to put together my disorganized thoughts. I learned a lot from our long conversations, which were fun, also when not related to the work.

I want to thank Goos Kant and Arnold L. Rosenberg for their useful comments on previous versions of chapter 3.

I thank all my friends at the Computer Science department, who made my stay there a pleasure. I won’t mention names so that I won’t forget anyone.

Finally, I have to thank the various secretaries that handle graduate students. I want to thank them for not preventing me from finishing my studies.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>1</td>
</tr>
<tr>
<td>1 Background</td>
<td>2</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>2</td>
</tr>
<tr>
<td>1.2 definitions</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Previous Work</td>
<td>5</td>
</tr>
<tr>
<td>2 Visibility Representations</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Preliminaries</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Finding a Visibility Representation of a Nonseparable Graph</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Constrained Visibility Representations</td>
<td>10</td>
</tr>
<tr>
<td>3 Drawings with Vertices Drawn as Points</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Left Adjusted Visibility Representations</td>
<td>14</td>
</tr>
<tr>
<td>3.3 Drawing Vertices as Points</td>
<td>21</td>
</tr>
<tr>
<td>3.4 Graphs That Are Hard to Draw</td>
<td>24</td>
</tr>
<tr>
<td>4 Drawings with Vertices Drawn as Rectangular Modules</td>
<td>26</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>26</td>
</tr>
<tr>
<td>4.2 Planarity Checking</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Planar Grid-Layouts – The Case of No Self-Loops</td>
<td>30</td>
</tr>
<tr>
<td>4.4 Planar Grid-Layouts – The Case With Self-Loops</td>
<td>34</td>
</tr>
<tr>
<td>4.5 Module Descriptions that are Hard to Draw</td>
<td>36</td>
</tr>
<tr>
<td>4.6 An Application</td>
<td>37</td>
</tr>
<tr>
<td>5 Non-Planar Drawings</td>
<td>39</td>
</tr>
<tr>
<td>5.1 Vertices as Points</td>
<td>39</td>
</tr>
<tr>
<td>5.2 Vertices as Modules (Grid-Layouts)</td>
<td>39</td>
</tr>
<tr>
<td>Bibliography</td>
<td>42</td>
</tr>
</tbody>
</table>
Abstract

For a 4-planar graph, a linear-time algorithm is described to construct a planar drawing on the rectilinear grid, with at most 3 bends in each edge. The drawings found by this algorithm, for 3-planar graphs with no self loops, have at most 2 bends in each edge. The drawings use $O(n^2)$ area and edges of length $O(n)$.

We also show a 4-planar graph that has no rectilinear planar drawing with at most 2 bends in each edge, and a 3-planar graph that has no rectilinear planar drawing with at most 1 bend in each edge.

Consider an input data which specifies a block diagram of some modules and connections between them (edges), which can be drawn on the plane in such a way that lines representing the connections do not cross. The size of the modules and the placements of the terminals on them is given as part of the input.

A linear-time algorithm is described to construct a planar drawing on the rectilinear grid, with at most 6 bends in a line representing a connection between two terminals of the same module, and at most 4 bends in any other line. The external face of the drawing may be chosen. We show a planar input with no self loops that has no planar drawing with less than 4 bends in each edge and another planar input that has no planar drawing with less than 6 bends in each self loop.

Finally, we consider grid drawings of specified block diagrams, which are not necessarily planar. We present a linear-time algorithm to construct such grid drawings with at most 4 bends in any line of the block diagram. We show inputs that have no such drawing with at most 3 bends in any line.
Chapter 1

Background

1.1 Introduction

We consider drawing a graph in the plane, in a way suitable for the layout of one layered electrical circuits or VLSI components. The drawing is done on the rectilinear grid, which is a collection of parallel lines at a constant distance one from another and a copy of them rotated by 90°. Edges are drawn as paths on the grid, connecting their two end vertices. The drawing must be planar, that is the drawing of two different edges have no common point, except maybe at their ends where the two edges have a common end vertex.

A bend is a point in the path, representing an edge, in which the path has a 90° angle. We want the drawing to have few bends in each edge. This is important where the drawing realizes a circuit, and a bend may cause a delay or a signal loss. It also makes it easier for humans to follow the drawings of the edges. We call a drawing in which every edge has no more than k bends a k-bend drawing. We investigate two types of drawings which are different in the way the vertices are drawn.

In the first type of drawings we consider, the vertices are drawn as distinct points at intersections of grid lines. A planar graph is a graph that has a planar drawing. A d-planar graph is a planar graph that has no vertex of degree greater than d. By definition, drawable graphs must be 4-planar. We describe a linear-time algorithm that produces a 3-bend drawing for any 4-planar graph and a 2-bend drawing for any 3-planar graph with no self loops. The drawings have $O(n^2)$ area, a total of $O(n)$ bends, and edges of length $O(n)$. We then show a 4-planar graph that has no 2-bend drawing and a 3-planar graph that has no 1-bend drawing.

In the second type of drawings we investigate, the vertices are drawn as rectangular modules on the grid, and edges are drawn as non-intersecting paths on the grid between the perimeters of two modules. The point that an edge connects to the perimeter of a module is called a terminal. The size of the modules and the placements of the terminals on them is given as part of the input. The input assigns to each edge two specific terminals.
We try to produce drawings with few bends in each edge. We consider two versions of the problem, in one flipping of the modules is allowed, in the other it is not. Rotation of the modules is always allowed.

For both versions of the problem linear-time algorithms are known that determine if a planar drawing of the input is possible. (When we say that a planar drawing is possible, we do not mean that such a drawing has to be on the rectilinear grid.) We will show a linear-time algorithm that constructs a 6-bend drawing for any input that permits a planar drawing. In the produced drawings, edges that are not self loops have at most 4 bends. We will show planar input with no self loops that has no 3-bend drawing and planar input that has no 5-bend drawing.

1.2 definitions

We present here definitions that will be used throughout the thesis. The definitions are gathered here, so that later the explanations will be more fluent, and to make them easier to find. Some definitions brought here may also appear elsewhere, and some definitions relevant only to a specific chapter may only appear there.

When we talk about a graph \(G(V, E)\), we are not restricted to simple graphs. This means that the graph may have parallel edges and self loops. We denote \(|V| = n\) and \(|E| = m\). For definitions of basic terms in the context of graphs see [Eve79]. We use the notation \(u \rightarrow v\) for an edge \(e\) which connects vertices \(u\) and \(v\) and \(u \leftarrow v\) for a directed edge \(e\) from \(u\) to \(v\).

If a connected graph has three different vertices \(a, b, c\), such that all the paths between \(a\) and \(c\) pass through \(b\), then \(b\) is called a cut vertex or a separation vertex of the graph. A connected graph that has no cut vertex is called nonseparable. Nonseparable graphs are sometimes called bi-connected.

A planar drawing of a graph on the plane is a mapping of it to the plane. Vertices are mapped to points or to closed curves in the plane, and edges are mapped to curves in the plane. The endpoints of the curve representing an edge must belong to the drawings of the end vertices of that edge. Except for this case no point belonging to a drawing of an edge or a vertex may belong to a drawing of another edge or vertex.

A planar graph is a graph that has a planar drawing. Ad-planar graph is a planar graph that has no vertex of degree greater than \(d\).

We consider drawing a graph in the plane, in a way suitable for the layout of a one layered electrical circuit or a VLSI component. The drawing is done on the rectilinear grid, which is a collection of parallel lines with a constant distance between neighboring lines, and a copy of them rotated by \(90^\circ\). Edges and vertices are drawn only on the grid.
We define the following for drawings on the rectilinear grid. A *bend* is a point in the path, representing an edge, in which the path has a 90° angle. We call a drawing in which every edge has no more than $k$ bends a *$k$-bend* drawing. The *area* of a drawing is the area of the smallest rectangle, sides parallel to the grid lines, containing the drawing. The area is measured in grid squares.

When we remove from the plane points that belong to a planar drawing of a connected graph, the plane is divided into continuous areas. (Points inside a drawing of a vertex are also removed.) The boundary of each of these areas represents a cycle of the graph; an edge may appear twice in such a cycle. Theses cycles are called the *faces* of the drawing. The face which represents the boundary of the infinite area is called the *external face*.

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 end-points around the vertices in the drawing. A *planar ordered graph* is an ordered graph that has a proper planar drawing. The edge ordering of a planar ordered graph is called a *planar representation*. An ordered graph determines the faces of any proper drawing, but doesn’t determine which one is external.

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 $s \to t$.

In an st bipolar orientation of a planar ordered graph, the only source (sink) of a subgraph consisting of one face, is called the *source (sink) of the face*. As will be shown later (Lemma 2.1), in every st bipolar orientation of a planar ordered graph, for every vertex, the incoming (outgoing) edges appear consecutively. The set of consecutive incoming (outgoing) edges of a vertex is called an *edge block*.

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. A visibility representation of a graph induces directions to the edges of the graph; the edges are directed from the lower vertex to the higher vertex. We assume that the ends of the drawing of a vertex are at the endpoints of it’s leftmost and rightmost incident edges.

We will call a vertex in a visibility representation *left adjusted* if the leftmost point of its drawing is the end of the drawings of two edges. An *LAVR (Left Adjusted Visibility Representation)* of a graph is a visibility representation such that in the directed graph it induces, every vertex which isn’t left adjusted is a source or a sink. See figure 2.1 for an example.
1.3 Previous Work

Drawings on the rectilinear grid have been studied because of their applications in VLSI planning and automatic graph and data drawing [DETT93, TDB88]. Planar drawings, in which lines do not cross, have been given a special consideration. In planar drawings, following the drawings of edges is easier; and the drawing can be used as a layout of a one layered electrical circuit.

Most of the previous work dealt with drawings in which vertices are drawn as grid points. Several aspects of such drawings have been considered. The area of the drawing is the area of the smallest rectangle, sides parallel to the grid lines, containing the drawing. The area is measured in grid squares and is considered better when small. It is known that every 4-planar graph can be drawn in \( O(n^2) \) area [Shi76, TT89] and that some 4-planar graphs require \( \Omega(n^2) \) area to be drawn [Shi76]. Other properties of drawings which are of interest are the total edge-length, the maximum length of an edge [MO85], and the total number of bends [Sto84, Tam87, TT89, TTV91]. A bend is a point in the path, representing an edge, in which the path has a \( 90^\circ \) angle.

Our interest is in drawings which minimize bends, but instead of considering the total number of bends in the drawing, we want to minimize the maximum 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. It also makes it easier for humans to follow the drawings of the edges. We call a drawing in which every edge has no more than \( k \) bends a \( k \)-bend drawing. (See figure 3.1.) An algorithm in [Tam87], given \( k \) and a planar representation, finds a \( k \)-bend drawing consistent with the representation, with the minimum total number of bends, in \( O(n^2 \log n) \) time. In [TT89] a linear-time algorithm is given to find a 4-bend drawing for simple graphs. Kant [Kan92] shows that every 3-planar graph without self loops and with five vertices or more has a 1-bend drawing; however the choice of the external face is determined by the algorithm. Two relevant papers have been written after our documentation of the results of chapter 3; they are:

- Biedl [Bie93] shows a linear-time algorithm to find a 2-bend nonplanar drawing of a graph.

- Biedl and Kant [BK94] show a linear-time algorithm to find 2-bend planar drawings of most simple graphs.

So far, we have only referred to drawings in which the vertices are drawn as grid points. In chapters 4,5 we consider drawings in which vertices are drawn as rectangles on the grid. The shape of each rectangle and the placements of the edge endpoints on its perimeter are given as part of the input. Although such drawings seem natural for VLSI
planning, we do not know of algorithms for finding such planar drawings. We only know of algorithms to check if a planar drawing exists for a given input [Pin83, Ami87, Zak93].
Chapter 2

Visibility Representations

In this chapter we describe known algorithms for finding visibility representations of graphs. The algorithms are described for two reasons: The first reason is that the descriptions of the algorithms makes the thesis more self contained. And the second reason is to clarify the use of these algorithms in later chapters.

Proofs are omitted in this chapter since they can be found in the referenced work.

2.1 Preliminaries

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. A planar ordered graph is an ordered graph that has a proper planar drawing. The edge ordering of a planar ordered graph is called a planar representation. An ordered graph determines the faces of any proper drawing, but doesn’t determine which one will be external.

Graphs are usually represented in computers by adjacency lists. The order of the incident edges of a vertex in the list can be viewed as their cyclic order. Therefore, ordered graphs can be represented in a computer in the same way that regular graphs are. When dealing with planar graphs, we will usually operate on a corresponding planar ordered graph.

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 $s \to t$. Given a non-separable graph and one of its edges $s \to t$, an st bipolar orientation of the graph, with the specified $s \to t$ edge, can be found in linear time [ET76]. Even and Tarjan use an equivalent notion of an st numbering.
Lemma 2.1 [RT86, TT86] In an st bipolar orientation of a planar ordered graph, the following hold:

- For every vertex, its incoming (outgoing) edges appear consecutively in the edge ordering.
- Every face consists of two edge disjoint directed paths.

Thus if we take a subgraph, consisting of one face of an st bipolar orientation of a planar ordered graph, the subgraph has exactly one source and one sink, called the source and the sink of the face. The set of consecutive incoming (outgoing) edges of a vertex is called an edge block.

Let us consider an st bipolar orientation whose edges have non-negative integer weights. The weight of the longest path from $s$ to each vertex $v$, which we denote by $d(v)$, can be found in linear time. (See, for example, [Eve79], pp 138–139.) 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. The values of $d$ are a topological ordering.

We look at an edge $e$ of a directed graph with a planar representation. We will denote the face to the left (right) of $e$ when $e$ is drawn upwards as $L(e)$ ($R(e)$).

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. A visibility representation of a graph induces directions to the edges of the graph; the edges are directed from the lower vertex to the higher vertex. We assume that the ends of the drawing of a vertex are at the endpoints of it’s leftmost and rightmost incident edges. A visibility representation is described by the following data: For each vertex $v$, its $Y$ axis coordinate $y(v)$; and for each edge $e$, its $X$ axis coordinate $x(e)$.

We will call a vertex in a visibility representation left adjusted if the leftmost point of its drawing is the end of the drawings of two edges. An LAVR (Left Adjusted Visibility Representation) of a graph is a visibility representation such that in the directed graph it induces, every vertex which is not left adjusted is a source or a sink. See figure 2.1 for an example.

We will now describe two known linear time algorithms for finding visibility representations. The difference between the two algorithms is that the second algorithm finds a visibility representation with some constraints.
2.2 Finding a Visibility Representation of a Non-separable Graph

In this section we describe an algorithm of Rosenstiehl and Tarjan [RT86], that was discovered independently by Tamassia and Tollis [TT86].

The directed dual graph $G^*(V, E^*)$ of an st bipolar orientation $G(V, E)$ with a planar representation is defined as follows:

- $V^* = \{ f \mid f \text{ is a face of } G \}$,
- $E^* = \{ L(e) \rightarrow R(e) \mid e \in (E - \{ s \rightarrow t \}) \} \cup \{ R(s \rightarrow t) \rightarrow L(s \rightarrow t) \}$.

It can be shown that $G^*$ is an st bipolar orientation, with $R(s \rightarrow t) \rightarrow L(s \rightarrow t)$ being the $s \rightarrow t$ edge.

**Theorem 2.1** [RT86, TT86] algorithm VR, described in figure 2.2, finds a visibility representation for any non-separable graph without self-loops, in linear-time.

The $s \rightarrow t$ edge is the leftmost edge in the visibility representation, and the face on its left is the external face. Therefore, the choice of $s \rightarrow t$ in the algorithm ensures that $f$ will be the external face.

The edges in the visibility representation are drawn upward according to their direction in the st bipolar orientation of $G$.

**Lemma 2.2** The visibility representations produced by algorithm VR are left adjusted.

**Proof:** Let $v \neq s, t$, and let $e_1$ be the leftmost incoming edge of $v$, and $e_2$ be the leftmost outgoing edge of $v$. Since $d^*(L(e_1)) = d^*(L(e_2))$, $x(e_1) = x(e_2)$.  

The height of the drawing is less than $|V|$, and the width of the drawing is less than the number of faces of $G$.  

---

Figure 2.1: a graph and an LAVR representation of it

tehemosh 2.1: נַעַר וַאוֹר קַוּיָּוָא מְשֶׁרָא שֵׁלָא
procedure VR(G, f);  { [RT86], very similar to [TT86] }  
{ G is a planar undirected nonseparable ordered graph, and its data structure }  
{ specifies its planar representation. The face f is specified to be external. }  
begin  
let s be a vertex on f ;  
let s⁻→t be an edge in the clockwise traversal of f ;  
direct the edges of G as an s→t bipolar orientation ;  { [ET76] }  
find G*, the directed dual graph of G ;  
set all edge weights to 1 ;  
calculate d(v) for the vertices of G and d*(v) for the vertices of G* ;  
for each vertex v of G do  
y(v) := d(v) ;  
x(s⁻→t) := -1 ;  
for each edge e' ∈ (E - {s⁻→t}) do  
x(e') := d*(L(e'))  
end;

Figure 2.2: finding an LAVR of a nonseparable graph

As will be shown later, the VR algorithm can be used to construct a visibility representation of any planar connected graph with no self-loops. If the graph is connected but separable, it can be made nonseparable by adding edges. These edges will be called auxiliary. An LAVR of the graph with the auxiliary edges can be found using the VR algorithm. After removing the drawings of the auxiliary edges from the resulting LAVR one gets a visibility representation of the graph. Notice, that this visibility representation may not be left adjusted. An algorithm for finding an LAVR of a connected graph is described in section 3.2.

2.3 Constrained Visibility Representations

In this section we are interested in producing visibility representations under certain constraints; we want certain paths of the graph to be drawn as straight vertical lines. We describe an Algorithm of Di Battista, Tamassia and Tollis [DTT92] that runs in linear-time. For consistency with the rest of the thesis, we describe this algorithm in a slightly different way than in [DTT92].

The input consists of a planar ordered graph which is an st bipolar orientation G(V, E), and some constraints. The output is a visibility representation of G that is proper, in which all edges are drawn pointing upwards. What we have described so far can be done
by the VR algorithm, if one removes the finding of the st bipolar orientation from the algorithm. Next, we describe the constraints imposed on the visibility representation, which force us to use the new algorithm.

A set $\Pi$ of mutually edge disjoint directed paths of the st bipolar orientation, is specified in the input. All the specified directed paths have to be aligned in the output visibility representation. That is, the X axis coordinate of all edges belonging to the same path must be the same. The specified paths may not cross one another in a proper drawing. That is, for any four edges $e_1, e_2, e_3, e_4$ incident to the same vertex in this order (not necessarily consecutively), if $e_1$ and $e_3$ belong to the same path, then $e_2$ and $e_4$ may not belong to the same path.

If some edge does not appear in a path of $\Pi$, it can be added as new path to $\Pi$. Therefore, we may assume that every edge appears in one path of $\Pi$. We denote by $p(e)$ the path of $\Pi$ that an edge $e \in E$ appears in.

We define the **constrained dual graph** $G^*(V, E^*)$ of $G(V, E)$ as follows:

- $V^* = \Pi \cup \{ f \mid f \text{ is a face of } G \}$,
- $E_w = \{ p(e) \to R(e) \mid e \in E \}$,
- $E_z = \{ L(e) \to p(e) \mid e \in (E - \{s \to t\}) \} \cup \{ p(s \to t) \to L(s \to t) \}$,
- $E^* = E_w \cup E_z$.

It can be shown that $G^*$ is an st bipolar orientation, with $p(s \to t) \to L(s \to t)$ being the $s \to t$ edge.

**Theorem 2.2** [DTT92] algorithm CVR that is described in figure 2.3 finds a constrained visibility representation for any st bipolar orientation of a planar ordered graph, in linear-time.

If one applies CVR($G, \Pi$), but assigns higher integer weights to the edges, the result is still a constrained visibility representation. This can be used to creating gaps in the visibility representation in specific places.
procedure CVR(G, II); { [DTT92], very similar to [TT86] } { G(V, E) is an st bipolar orientation of a planar ordered graph. } { II is the set of aligned paths. }
begin
Construct \( G^*(V^*, E_w \cup E_r) \), the constrained dual graph of \( G \);
Set edge weights as follows: \( \forall e \in E : w(e) = 1 \),
\( \forall e^* \in E_w : w^*(e^*) = 1 \), \( \forall e^* \in E_z : w^*(e^*) = 0 \);
Calculate \( d(v), d^*(v) \) for the vertices of \( G, G^* \) respectively;
for each vertex \( v \) of \( G \) do
\( y(v) := d(v) \);
for each edge \( e \in E \) do
\( x(e) := d^*(p(e)) \)
end;

Figure 2.3: finding a constrained visibility representation

תרשים 2.3: מציאת אנואר קווים מנוקלים
Chapter 3

Drawings with Vertices Drawn as Points

3.1 Introduction

In this chapter, we consider drawing a planar graph on the rectilinear grid, with no crossings, such that there are few bends in the drawing of every edge. Vertices are drawn as points at intersections of grid lines. Edges are drawn as non-intersecting paths on the grid. By definition, drawable graphs are 4-planar.

In this chapter we denote $G$ as the underlying undirected graph of the directed graph $\tilde{G}$.

Planar drawings on the rectilinear grid have been studied because of their applications in VLSI planning and automatic graph and data drawing [TDB88]. Several aspects of the drawings have been considered. The area of the drawing is the area of the smallest rectangle, sides parallel to the grid lines, containing the drawing. The area is measured in grid squares and is considered better when small. It is known that every 4-planar graph can be drawn in $O(n^2)$ area [Shi76, TT89] and that some 4-planar graphs require $\Theta(n^2)$ area to be drawn [Shi76]. Other properties of drawings which are of interest are the total edge-length, the maximum length of an edge [MO85], and the total number of bends [Sto84, Tam87, TT89, TTV91]. A bend is a point in the path, representing an edge, in which the path has a 90° angle.

Our interest is in drawings which minimize bends, but instead of considering the total number of bends in the drawing, we want to minimize the maximum 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. It also makes it easier for humans to follow the drawings of the edges. We call a drawing in which every edge has no more than $k$ bends a $k$-bend drawing. (See figure 3.1.) An algorithm in [Tam87], given $k$ and a planar representation, finds a $k$-bend drawing consistent with the representation, with the minimum total number
of bends, in $O(n^2 \log n)$ time. In [TT89] a linear-time algorithm is given to find a 4-bend drawing for simple graphs. Kant [Kan92] shows that every 3-planar graph without self loops and with five vertices or more has a 1-bend drawing. However the choice of the external face is determined by the algorithm.

Figure 3.1: a 3-bend drawing which is not a 2-bend drawing

We describe an algorithm that produces 3-bend drawings for 4-planar graphs and 2-bend drawings for 3-planar graphs. The drawings have $O(n^2)$ area, a total of $O(n)$ bends, and edges of length $O(n)$. We then show a 4-planar graph that has no 2-bend drawing and a 3-planar graph that has no 1-bend drawing. The algorithm runs in linear time.

We will draw each connected component of the graph separately and thus consider from now on only connected graphs.

The algorithm starts by finding an LAVR of the graph without its self loops. Next, the drawing is converted to a drawing with vertices drawn as points.

### 3.2 Left Adjusted Visibility Representations

We start with some preliminaries.

The input to our drawing algorithm is a planar ordered graph and a specification of the face which is to be external. If no external face is specified, we can choose any face to be external. The output is a drawing congruous with the input data.

Linear-time algorithms for checking the planarity of a graph, and finding a planar representation of it are known [LEC67, ET76, BL76, CNAO85, HT74]. Therefore, if we do not have a planar representation of the graph, or do not know if the graph is planar, then we can check for planarity and find a planar representation while keeping the running time linear. We will work with planar ordered graphs, and assume that the data structure of the graph specifies its planar representation.

We assume knowledge of the discussion of st bipolar orientations and the related definitions that appear in section 2.1.

**Lemma 3.1** A graph that has an st bipolar orientation is nonseparable.

We denote the input graph, without its self loops, $G_0$. Recall that the planar representation and chosen external face of $G_0$ are derived from the input data. The drawing algorithm of this chapter uses an LAVR of $G_0$. We show an algorithm that, given $G_0$, a
connected ordered graph with no self loops which may be separable, and $f$ a face of $G_0$, finds an LAVR of $G_0$ that is proper and $f$ is its external face. This algorithm generalizes the algorithm in [RT86] (described in section 2.2) to find an LAVR of connected graphs, which are not necessarily nonseparable.

Our algorithm modifies $G_0(V, E)$, to form a planar ordered graph $\tilde{G}_1(V, E')$. $\tilde{G}_1$'s undirected underlying version, $G_1(V, E'')$ contains $G_0$ as a subgraph and is nonseparable. The edges of $\tilde{G}_1$ which correspond to edges in $E'' - E$ are called auxiliary. If we remove the auxiliary edges from $\tilde{G}_1$, the planar representation is the same as that of $G_0$. The construction of $\tilde{G}_1$ from $G_0$ is done as follows: The edges of $G_0$ are directed one nonseparable component after another. Each component is directed as an st bipolar orientation. The auxiliary edges are composed of up to one auxiliary edge for each directed component, each auxiliary edge corresponds to a different component. After directing the edges of each component, the components directed so far together with the auxiliary edges corresponding to them, constitute an st bipolar orientation; we will call this temporary graph $\tilde{G}^t$. The auxiliary edge corresponding to each component is determined at the time the component is directed, but it is only added later. The auxiliary edges are added only after directing all of the edges. When directing a component $C$ other than the first that is connected to $\tilde{G}^t$ by a cut vertex $u$, it is directed in one of two ways depending on $u$ being the sink of the face of $\tilde{G}^t$ containing $C$; see figure 3.2. The auxiliary edge corresponding to $C$ is also shown in figure 3.2. However, in case b of figure 3.2, this edge may not be eventually added to the graph if it is not needed; in this case there will be no auxiliary edge corresponding to $C$ in $\tilde{G}_1$.

![Figure 3.2](image)

Figure 3.2: a visual description of the two cases of directing a nonseparable component

Next, the auxiliary edges are added to the graph. This is done according to markings made when directing the graph. Following, an LAVR of the new graph is found by the algorithm in [RT86]. We remove the auxiliary edges from the visibility representation and
it remains an LAVR, due to the special way the auxiliary edges have been added. Our algorithm is described in figures 3.3, 3.4.

Remarks:

1. We perform DFS on $G_0$ to find the nonseparable components. We start the DFS at the st-edge $s \rightarrow t$, and choose the edges incident to a vertex in the order of their placement in the planar representation, starting with the edge through which the vertex has been reached. Later, when we do a DFS on the nonseparable components tree, we traverse the components in the order in which their first edge is traversed here. This ensures that we won’t traverse a component that is inside a face of another yet-untraversed component.

2. The first edge in the clockwise traversal of an external face of a component, starting at the entry cut-vertex of the component, can be found easily. It is the first traversed edge of the component, in the DFS used to determine the nonseparable components. (See remark 1.)

3. Observe that the edges of $f''$ are a subset of the edges of the face of $G_0$ containing $f''$, but some of the edges of this face may not be directed yet. These latter edges belong to the external faces of other components of $G_0$.

If a vertex is not a source or a sink, and if its block of incoming (outgoing) edges consists of more than one edge, then the first and last edges of the block are called extreme. The remaining visibility representation, after the removal of the auxiliary edges, is ensured to be left adjusted, by making sure that no auxiliary edge is extreme.

**Lemma 3.2** The auxiliary edges in $G_1$, as constructed by the algorithm, are not extreme.

*Proof:* After the first edge-adding stage, each of the added auxiliary edges is the only incoming edge of its end vertex ($s''$). In the edge-blocks of their start vertex ($s'$) these edges are inserted between the two edges of $f$. Therefore, after the first stage there are no extreme auxiliary edges.

After the second edge-adding stage, each of the added auxiliary edges is the only outgoing edge of its start vertex ($t''$). In the edge-blocks of their end vertices ($t'$), they are always inserted between the two edges of $f$. Therefore, the edges added in this stage are not extreme. What is left to show is that the edges that are added in the first stage do not become extreme in the second stage. This may happen if an edge is added to an edge-block that contained only one auxiliary edge. For this to happen, an incoming edge has to be added to a vertex marked as a temporary source. In the second stage, we add only incoming edges to vertices marked as sinks of faces. A vertex marked as a temporary
procedure LAVR(G₀, f');
{ G₀(V, E) is a planar undirected ordered graph. The face f' is specified as external. }
begin
let sᵣ be a vertex on f';
let sᵣ→tᵣ be an edge in the clockwise traversal of f';
orientation(G₀, sᵣ→tᵣ); { see figure 3.4 }
{ The current graph is G₀, a directed version of G₀. }
{ Every face of G₀ has two vertices marked as its source and sink. }
{ Also, vertices may be marked as temporary sources or as temporary sinks. }
for every face f of G₀ do { first adding stage }
begin
let s' be the vertex marked as the source f;
traverse f clockwise starting at s' with s'→s'a;
for every vertex s'' traversed that is marked as a temporary source and isn’t s' do
begin add a new auxiliary edge s''→s'a inside f end
{ The added edges are ordered consecutively around s' after e' in their creation order. }
the vertex marked as the sink of f is marked as the sink of all the newly created faces
end;
{ Now the faces of the current graph may be different. }
for every face f of the current graph do { second adding stage }
begin
let t' be the vertex marked as the sink of f;
traverse f clockwise starting at t' with t'→t'a;
for every vertex t'' traversed that is marked as a temporary sink and isn’t t' do
begin add a new auxiliary edge t''→t'a inside f end
{ The added edges are ordered consecutively around t' after e' in their creation order. }
end;
{ We call the resulting nonseparable graph G₁. }
find an LAVR of G₁ such that sᵣ→tᵣ is the leftmost edge; { RT86, section 2.2 }
remove the auxiliary edges from the LAVR of G₁ to produce an LAVR of G₀
end;

Figure 3.3: finding an LAVR of a connected graph

התרשים 3.3: מציאת ציר קוי מחודש בלתי ניתן לשני קישיור
procedure orientation($G_0$, $s, \overset{s\rightarrow t_r}{\leftarrow}$);
{ a subroutine of the algorithm in figure 3.3, that directs the edges of $G_0$ to produce $\widetilde{G}_0$ }
begin
find the nonseparable components of $G_0$ { see remark 1 } ;
let $c_r$ be the component containing $e$ ;
find the nonseparable components tree of $G_0$ such that the root component is $c_r$ ;
direct the edges of $c_r$ as an st bipolar orientation with $s_r \overset{s\rightarrow t_r}{\rightarrow} t_r$ being the $s\rightarrow t$ edge ; { [ET76] }
mark the source (sink) of each face $f''$ of $c$, as the source (sink)
of the face of $G_0$ containing $f''$ { see remark 3 } ;
do a DFS on the component tree and when first reaching a component $c$ other than $c_r$ do:
begin { see remark 1 }
let $s'$ be the entry cut-vertex of the component in the tree ;
let $f$ be the external face of $c$ ;
let $s' \overset{t'}{\rightarrow} t'$ be the first edge in the clockwise traversal of $f$ starting at $s'$ ; { see remark 2 }
if $s'$ is the vertex marked as the sink of the face of $G_0$ that $f$ is a part of then
begin
direct the edges of $c$ as an st bipolar orientation with $t' \overset{s\rightarrow t_r}{\rightarrow} s'$ being the $s\rightarrow t$ edge ;
mark $t'$ as a temporary source
end
else
begin
direct the edges of $c$ as an st bipolar orientation with $s' \overset{s\rightarrow t'}{\rightarrow} t'$ being the $s\rightarrow t$ edge ;
mark $t'$ as a temporary sink ;
if $s'$ is marked as a temporary sink then unmark it
end;
mark the source (sink) of each face $f''$ of $c$ other than $f$ as the source (sink)
of the face of $G_0$ containing $f''$ { see remark 3 }
end
end;

Figure 3.4: a subroutine of the algorithm in figure 3.3

תרשים 3.4: שורטט של האלגוריתם בתשוש 3.3
The dotted lines represent parts of the drawing that are removed, due to the removal of the auxiliary edges.

Figure 3.5: input and output of the LAVR algorithm.

Lemma 3.3 $G_1$, the undirected underlying version of $\tilde{G}_1$, is a nonseparable planar ordered graph.

Proof: $G_0$ is planar, and the auxiliary edges are added inside faces of the current graph without introducing edge crossings. Therefore $G_1$ is a planar ordered graph.

Next we show that $\tilde{G}_1$ is an st bipolar orientation with $s\to t_r$ being the $s\to t$ edge. We have to show that $s$ and $t$ are the only source and sink of $\tilde{G}_1$ and that $\tilde{G}_1$ is acyclic.

All the sources and sinks of $\tilde{G}_0$ other than $s,t$ are marked as temporary sources and temporary sinks respectively. Following, we show that the vertices marked as temporary sources and sinks are not sources and sinks of $\tilde{G}_1$, and therefore the only source and sink of $\tilde{G}_1$ are $s_r$ and $t_r$.

Every vertex $v$ that is marked as a temporary source appears on a face $f$ of $G_0$, for which $v$ is not marked as a source. This is the face containing the external face of the nonseparable component closest to the root on the component tree, that contains $v$. In the first adding stage, while traversing $f$, an edge entering $v$ is added. Therefore, $v$ is not a source of $\tilde{G}_1$.

Every vertex $v$ that is marked as a temporary sink appears on a face $f$ of $G_0$, for which $v$ is not marked as a sink. If $f$ is split to several faces in the first adding stage, the resulting face that contains $v$ still has the same marked source, which is not $v$. Therefore, in the second adding stage, an edge leaving $v$ is added. Consequently, $v$ is not a sink of $\tilde{G}_1$.

Now we show that $\tilde{G}_1$ has no directed cycles. First we show that the faces of $\tilde{G}_1$ are not directed cycles and then we show that this implies that $\tilde{G}_1$ has no directed cycles.

Clearly the faces of $\tilde{G}_0$ are not directed cycles. Observe that after the first adding stage there are no faces that are directed cycles; this is because the new faces that are
formed by adding auxiliary edges have a source which is the start vertex of the auxiliary edge. After the second adding stage there are still no faces that are directed cycles, since the newly formed faces have a sink which is the end vertex of the new auxiliary edge. So we see that the faces of \( \tilde{G}_1 \) are not directed cycles.

We prove that \( \tilde{G}_1 \) has no directed cycle by contradiction. Let’s assume that \( \tilde{G}_1 \) has a directed cycle. We showed that \( G_1 \) is planar. We look at a proper planar drawing of \( \tilde{G}_1 \) in which the \( s \rightarrow t \) edge is on the external face. Every cycle in \( \tilde{G}_1 \) bounds a finite area in the drawing. We take a directed cycle \( c \) that bounds the least area in the drawing. The \( s \) and \( t \) vertices are outside of the area \( c \) bounds since they can’t be on \( c \), and the \( s \rightarrow t \) edge is on the external face. This implies that there is no source or sink inside the bounded area. There must be an edge incident to a vertex on \( c \) that is drawn in the bounded area; otherwise \( c \) is a face, and we have seen that a face can’t be a directed cycle. We trace a path in the bounded area starting with this edge, going through edges in the same direction, until we reach a vertex which is on \( c \). This must happen since there is neither a source, nor a sink, nor a “smaller” directed cycle in the bounded area. This path with a part of \( c \) constitutes a “smaller” directed cycle. A contradiction.

Thus, \( \tilde{G}_1 \) has no directed cycles and has only one source and one sink connected by an edge, rendering it an st bipolar orientation. By Lemma 3.1, \( G_1 \) is nonseparable. ■

Theorem 3.1 The algorithm in figure 3.3 finds an LAVR for any connected planar ordered graph with no self loops. Also, the drawing is proper and has the given external face.

Proof: \( G_0 \) is a subgraph of \( G_1 \). The planar representation of the original edges is the same. We have shown in Lemma 3.3 that \( G_1 \) is nonseparable and planar, and therefore we can find an LAVR of it [RT86]. After deleting the auxiliary edges we get a visibility representation of the original graph which is proper.

The side of the \( s \rightarrow t \) edge which is in the given external face of \( G_0 \) is drawn in the external face of the LAVR of \( \tilde{G}_1 \). After the deletion of the auxiliary edges, it stays in the external face of the drawing. Therefore, the given external face, which contains this side of the \( s \rightarrow t \) edge, is the external face of the drawing of \( G_0 \).

What is left to be shown is that the visibility representation of \( G_0 \) that is produced is left adjusted. The edges that are removed (the auxiliary edges) are not extreme as shown in Lemma 3.2. Therefore, the leftmost edge of an edge-block is not removed unless the edge-block contains only this edge, or the vertex of the edge-block is a source or a sink of \( \tilde{G}_0 \). This means that a vertex that has both outgoing and incoming edges in \( \tilde{G}_1 \) will either keep both its leftmost incoming and outgoing edges, or it will become a source or a sink of \( \tilde{G}_0 \). Therefore, the drawing remains an LAVR after the deletion of the auxiliary edges. ■
Clearly, the algorithm for finding an LAVR runs in linear time, including the algorithm used from [RT86] which runs in linear time (see section 2.2). As we will see in the following section, changing the LAVR to the wanted drawing can also be done in linear time. Thus, the whole procedure runs in linear time.

Let us consider the area of the determined LAVR. We get an LAVR for \( G_1 \) as in [RT86]. We denote the number of faces, edges, and vertices of \( G_1 \) by \( f', m', n' \), respectively. According to the algorithm of [RT86], (see section 2.2) the height of the LAVR is at most \( n' - 1 = n - 1 \), and its width is at most \( f' - 1 \). The original graph is 4-planar, so we have \( m \leq 2n \). We add at most \( n - 1 \) auxiliary edges, so we have \( m' \leq 3n - 1 \). By Euler’s formula we have \( f' - 1 = m' - n' + 2 - 1 \leq 3n - 1 - n + 1 = 2n \). Therefore, the width of the LAVR is at most \( 2n \), and its height is at most \( n - 1 \).

### 3.3 Drawing Vertices as Points

In this section, we consider the construction of a planar drawing with vertices drawn as points.

A self loop adjacent to a vertex of degree four, whose endpoints do not appear consecutively around the vertex, can not be drawn with less than 4 bends. These incidences can be searched for and corrected, by changing the planar representation at such vertices, in linear time. It is the only exception in which the planar representation has to be changed.

If the graph has only one vertex and two self loops, then we will draw it in the obvious way. A problem arises if a face containing only one of the self loops is supposed to be the external face. There is no 3-bend drawing of this case; in fact, a 5-bend drawing is required. This is the only exception to the free choice of the external face.

If the connected graph has more than one vertex, then we will use a procedure similar to the one used in [TT89] to create the desired drawing. We find an LAVR of the graph without its self loops using the algorithm in figure 3.3. Next, we change the drawing so that vertices are drawn as points. This is done by applying transformations to the vertices. A transformation on a vertex in the LAVR transforms the shape of the vertex and the connections of its incident edges. It also adds a self loop to the vertex if one existed in the original graph. The shapes of edges change just near their ends; the rest remains a vertical line. The transformations are described in figure 3.6. Some transformations require new grid lines to be added to the drawing. These are inserted to the drawing so that all the lines in the new shape of the vertex can be drawn on grid lines. This increases the area of the drawing: The height and width of the drawing are increased with each transformation by at most two. Therefore the height and width of the drawing increase by no more than \( 2n \) each, and the area of the drawing remains \( O(n^2) \). The horizontal position of the edges,
after the transformations, stays the same, except for the compensations for the added grid lines. The final drawing can be found in linear time.

Another approach to the production of the final drawing, is to modify the transformations to record only the shape of the drawing, but omit the length of the lines. This procedure produces an orthogonal representation [TTT89]. The orthogonal representation can be converted to the final drawing in linear time [Tam87]. This approach may lead to a smaller multiplicative constant in the bound on the total number of bends in the drawing, but it also leads to a bigger constant in the bound on the area.

The operation of the whole drawing procedure is illustrated in figure 3.7. The graph drawn is taken from [TTT91].

**Theorem 3.2** After performing the transformations, every edge has no more than 3 bends.

**Proof:** In the proof, we will look at the edges of $G_0$ as directed. The directions induced by the visibility representation of $G_0$ are the same as in $\overline{G_0}$. All the transformations except for $T_{15}, T_{21}$ present no more than two bends at the bottom of an edge and no more than one bend at the top of an edge. Therefore, every edge that isn’t the leftmost incoming edge of a vertex on which $T_{15}$ or $T_{21}$ has been performed has no more than three bends.

What is left to prove is that those edges which have two bends at the top have no more than one bend at the bottom. Such an edge $v' \rightarrow v''$ is the $s \rightarrow t$ edge of the st bipolar orientation of its component $c$, since it is the leftmost incoming edge of the sink of $c$. This implies that $e$ is the leftmost outgoing edge of $v'$ among those which belong to $c$. If $e$ is
not the rightmost outgoing edge of \( v' \), then it has no more than one bend in the bottom. Otherwise \( e \) is the only outgoing edge of \( v' \) belonging to \( c \), so \( e \) is composed only of \( e \). This means that there are two components connected at \( v'' \), and they are both in the external face of the other. This is possible only if \( v'' \) is \( t_r \) which implies that \( v' \rightarrow v'' \) is \( s_r \rightarrow t_r \). So \( e \) is the leftmost outgoing edge of \( v \), which is a source, and therefore \( e \) has no more than one bend at the bottom.

It is important that we use an LAVR since the transformations work only on left adjusted vertices. Otherwise, we could have had vertices with shapes for which suitable transformations don’t exist.\(^1\)

The transformations of vertices without self loops present at most four bends at each vertex. Therefore, the total number of bends is at most \( 4n \). In case of a graph with self loops, two of the transformations present six bends, thus increasing the upper bound on the total number of bends. The edges remain of length \( O(n) \).

When the algorithm is applied to a 3-planar graph, with no self loops, the transformations present at most one bend at each end of every edge. Thus, a 2-bend drawing is produced for 3-planar graphs. For 3-planar graphs one can transform the LAVR into a rectilinear drawing by simply choosing a point to represent each vertex. Obviously the area stays the same as in the LAVR.

---

\(^1\) Here is an example of a vertex with a shape that can’t appear in an LAVR, for which there is no transformation that presents no more than two bends at the bottom of an edge and no more than one bend at the top of an edge.
3.4 Graphs That Are Hard to Draw

We show here graphs that are hard to draw. These graphs are not ordered graphs. Following, we prove that there isn’t a 2-bend (1-bend) drawing for every 4-planar (3-planar) graph.

Lemma 3.4 The planar and triangulated simple graph in which every vertex is of degree four (octahedron, see figure 3.8) has no 2-bend drawing.

Proof: This graph is triconnected and, therefore, has only one planar representation, which uniquely determines the faces. We choose any face to be the external face and show that it can’t be drawn with no more than two bends in each edge, in a way that the rest of the graph can be drawn inside.

The external face, as all the others, has three edges, and its three vertices are of degree four. In the drawing of the external face the drawing of each vertex must have a 90° angle and a 270° angle, where the 270° angle must be inside; otherwise the two edges that are incident to the vertex and are not in the external face, can’t be connected to the vertex. We show that such a drawing of the external face is not possible with at most two bends in each edge.

When we go around the drawing of the face, which is a closed curve, at the drawing of each vertex we turn 90° towards the outside of the curve. To close the curve, we must turn 360° towards the inside of the curve, and to compensate for the turns at the vertices we must turn 630° inside. Each bend on an edge can turn at most 90° inside, so we will need at least \( \frac{630}{90} = 7 \) bends to draw the face. With at most two bends in each of the three edges there can’t be seven bends in the external face.

Lemma 3.5 The planar and triangulated simple graph in which every vertex is of degree three (tetrahedron, see figure 3.8) has no 1-bend drawing.
Proof: The proof is similar to the previous lemma. Again we show we can’t draw the external face in a way that the rest of the graph can be drawn. Here at every vertex, at best we turn 0°, so we have to turn 360° inside with the bends. This requires four bends, but with at most one bend in each of the three edges there can’t be four bends in the external face.

Lemmas 3.4 and 3.5 imply that for \( k = 2,3,4 \), there is a \( k \)-planar graph for which a \((k - 2)\)-bend drawing does not exist. The case for \( k = 2 \) is trivial.
Chapter 4

Drawings with Vertices Drawn as Rectangular Modules

4.1 Introduction

In this chapter, we consider drawings in which the vertices are drawn as rectangular modules on the grid, that is block diagrams. 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.

As before, 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.

We want to control certain qualities of the block diagram that we draw (find a layout of). Therefore, more data is specified in the input than just a graph.

The input for drawing a block diagram, 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 task of drawing 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 (grid) point, then they cross each other at that point.
- No path shares any point with any module, except its designated end-points.

Such a drawing is referred to as a grid-layout of the module description. If no two paths intersect, the grid-layout is called planar. (See figure 4.1).

![Figure 4.1: a 3-bend planar grid-layout which is not 2-bend](image)

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 planar grid-layout (without flippings) 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. 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. Applying these flippings to the original module description yields a planar module description. In such a case, the module description is called flip-planar. We allow rotation of the modules since it does not effect planarity.

Here, bends have the same meaning as before: A bend is a point of a path in a grid-layout in which the path makes a $90^\circ$ 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.

In this chapter, 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 \( u \rightarrow v \) for an edge \( e \) which connects vertices \( u \) and \( v \) and \( u \leftarrow v \) for a directed edge \( e \) from \( u \) to \( v \).

### 4.2 Planarity Checking

The planar grid-layout problem is divided into two independent parts. The first part is checking if the input is planar. This is different when flippings are allowed or not. Both cases have known linear-time solutions. The second part is finding a corresponding grid-layout.

In case flippings are not allowed, we need to know if the input module description is planar. One may use a linear-time algorithm [Zak93, Pin83, Ami87] to check if the module description is planar.

In case flippings are allowed, we need to check if by flipping some of the modules, we can get a planar module description. In this case we say that the module description is flip-planar. Pinter [Pin83] shows a linear-time algorithm to determine if a given module description is flip-planar. For the drawing algorithm we also need the actual planar module description, which we get by flipping some of the modules. Following is a description of Pinter’s algorithm.

Create a new non-ordered graph \( G' \), a modification of the input ordered graph \( G \), in which every vertex of degree three or more is replaced by a wheel, as shown in Figure 4.2. \( G' \) is called the wheel graph corresponding to \( G \).

The following lemma is a well known corollary of a theorem of Whitney [Whi32].
Lemma 4.1 A planar triconnected graph with no self loops has only one planar representation up to flipping (the whole graph).

Proof: In [Whi32] it is shown that a planar triconnected graph with no self loops has the same faces in any planar drawing. The meaning of “the same face” there is the same set of edges belonging to the face. Since the graph is nonseparable and has no self loops, a vertex may not appear more than once in a face. Every face is a simple cycle. Therefore there are exactly two possibilities for the clockwise order of the edges in each face, when one is the reverse of the other.

We have seen that there are two possibilities to order the edges in each face. Now, we will look at the relation between the “selections” of the order of each face. Let us look at some planar drawing of the graph on the sphere, and at an arbitrary face $f$ of the drawing. Each edge appears on two different faces, and when walking clockwise along the two faces, we pass the edge in opposite directions. This means, that the selection the order of the face at one side of the edge, determines the order of the face on the other side. We look at the order of $f$ in the drawing; since the dual graph is connected, this order determines the order of all other faces, without having to refer to the drawing. The clockwise order of the edges in each face gives us a planar representation. We have seen that the clockwise order of the edges in $f$ determines the edge ordering. Since there are two (reverse), not necessarily different, possibilities for this order, there are at most two planar representations of the graph. What is left to show is that the other edge ordering, if different, is a planar representation, and that one of these planar representations is the flipped version of the other. This can be seen by taking a planar drawing on the sphere, and turning the sphere inside out.

Lemma 4.2 A module description is flip-planar if and only if its corresponding wheel graph is planar.

Proof: If the wheel graph is planar, there is a planar drawing of it. Since each wheel is triconnected it has only one planar representation up to flipping by lemma 4.1. The order of the edges around the center of the wheel is either identical to the edge order around
the original vertex or is its reversal. By removing the edges and vertices of the rim of the wheel we get a planar drawing of the original graph.

If there is a planar drawing of the graph that corresponds to a flip-planar module description, then by adding a small circle around each vertex in the drawing we get a planar drawing of the wheel graph. Therefore the wheel graph is planar.

To check if a drawing is possible, we check if \( G \) is planar [LEC67, ET76, BL76, HT74]. If \( G \) is planar we make it a planar ordered graph [CNAO85], and extract from it an edge ordering that makes \( G \) a planar ordered graph. The clockwise cyclic order of the edges incident to a vertex is the order of the corresponding radial edges in the wheel around the center. This shows what modules are flipped, yielding a planar module description. To summarize:

**Theorem 4.1** There is a linear-time algorithm that checks if a module description is flip-planar, and if it is, produces a corresponding planar module description and a specification of the flipped modules.

All the algorithms used in this section run in linear time. The remaining task is to construct a planar grid-layout for the planar module description at hand.

### 4.3 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.

We draw \( G \) in several stages. Following we give an overview of the algorithm and later discuss the details. First, some edges may be added and the edges of the ordered graph are directed to become an st bipolar orientation. Next, the rotation of the drawing of each vertex is determined, that is, for every vertex we choose one of the four possible rotations for its module. Following, the edges incident to each vertex are divided to edges that will be drawn to the right or to the left of the module. Then, we find a visibility representation of the ordered graph with enough space in it to accommodate the drawings of the modules. And finally, we translate the visibility representation to the grid-layout by drawing the modules. Now, let us present a more detailed description.

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 the previous chapter. Next, we find an st bipolar orientation of the new ordered graph. The added edges are called auxiliary and are not drawn in the grid-layout. As in [ET76], one may choose any edge to be $s \to t$. We choose an edge such that when it is draw 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 want to 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 $t$. We need to know which edges are the leftmost and rightmost edges of the incoming (outgoing) edges incident to a vertex. Lemma 2.1 implies that there is no ambiguity in their indication for vertices other than $s$ and $t$.

Lemma 2.1 implies that 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 4.3 demonstrates how this can be done. More than one appropriate rotation may be possible.

![Figure 4.3: rotation of a vertex which is not a source nor a sink](image)

Figure 4.3: 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 paths have 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 $s \rightarrow t$ will be the leftmost edge of the resulting visibility representation, and the correct face will be external.

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 the section, therefore no edge is incident to $v$ at this section.

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* at 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 will be considered drawn to the left of the vertex, and will be called aligned at 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 at 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 corresponding edge is aligned at one of its end vertices, its weight is the vertex’s module width plus 2. If the edge is aligned at 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 at the vertex.

Finally, we translate the visibility representation to the grid-layout changing the drawing of each vertex. We make all the changes to the drawing of the vertex in an area of the size of a rectangle, whose top side is on the line of the vertex in the visibility representation. The height of the rectangle is the drawn height of the vertex plus 1. The module itself is drawn below the section of the vertex in the visibility representation that has no incident edges, such that it will have the correct amount of space above and below it, as calculated in the drawn height. See figure 4.4.

For every edge incident to the right and left sides of the module, a horizontal line 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 lines are used for the connection. A vertical line originating at the terminal, and a horizontal line. The horizontal lines 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 figure 4.4.

![Figure 4.4](image)

(a) - after the edges are directed, the rotation of the module is determined and the incident edges 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 4.2 A linear-time algorithm exists that given a planar module description with no self-loops finds a corresponding 4-bend planar grid-layout.

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

Self loops impose a problem. A visibility representation can not represent self loops. Also, some module descriptions with self loops have no 5-bend drawings, as shown in lemma 4.4. Following we describe how we deal with module descriptions that may have self loops.

Before drawing a module description, we put a new vertex in the middle of every self-loop, that is, replace every self-loop by two edges and a new vertex. This is shown in figure 4.5(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 drawn as 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 placement to the right or to the left of the incident edges, as in section 4.3.

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 at a dummy vertex.

When translating the visibility representation, we do not change the drawing of the dummy vertices, we just change the drawing of the normal vertices as before. The horizontal line representing a dummy vertex, together with the drawing of the two edges incident to it, constitute the drawing of the self-loop in the grid-layout.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure45.png}
\caption{drawing a graph that has self-loops}
\end{figure}

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 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 or 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 figure 4.5. Edges which are not self-loops have, as before, at most four bends. Self-loops are composed of two edges and a dummy vertex in the visibility representation. Each of these two edges has at most two bends near the normal vertex the self-loop is incident 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 4.3** A linear-time algorithm exists that given a planar module description, finds a corresponding 6-bend planar grid-layout in which every edge that is not a self-loop has at most \( \frac{1}{3} \) bends.

The coordinates of the drawing are determined by the two topological orderings, therefore the maximal coordinate is smaller than the sum of the edge weights. We denote the sum of the widths of the modules in their rotation in the grid-layout by \( W \), and the sum of the heights of the modules by \( H \). So the width and height of the produced drawing are \( W + O(|E|) \), \( H + O(|E|) \) respectively.

### 4.5 Module Descriptions that are Hard to Draw

There are linear-time algorithms to check if a module description is planar or flip-planar. We have shown a linear-time algorithm that given a planar module description, finds a corresponding 6-bend planar grid-layout in which every edge that is not a self-loop has at most 4 bends. Thus the whole process of finding a planar grid-layout for a given module description, if one exists, can be done in linear time. As far as the upper bounds on the number of bends in each edge are concerned, we give examples of planar module descriptions which demonstrate that our bounds are tight. (See figure 4.6). These examples do not depend on module flippings or on the choice of the external face.

![Figure 4.6: planar module descriptions that do not draw well](image)

(a) requires at least 4 bends for some edge (b) requires at least 6 bends for some self-loop

**Lemma 4.3** The module description in figure 4.6(a) has no 3-bend grid-layout.
Proof: We show that any grid-layout of the module description has an edge that must have at least 4 bends. Notice, that flipping only one of the modules would make the module description non-planar. First, we look at the rotation of the two modules (only their relative angle is important). For any planar grid-layout, there is a side \( A \) of the first module and a side \( B \) of the second module, such that:

- \( A \) faces the same direction as the opposite side of \( B \)
- an edge connects \( A \) and \( B \)
- an edge connects the opposite sides of \( A \) and \( B \)

One of these edges needs at least 4 bends.

Lemma 4.4 The module description in figure 4.6(b) has no 5-bend grid-layout.

Proof: Each of the self loops needs at least 6 bend to be drawn, if the face composed only of that self loop is not the external face. Since only one of these faces can be external, at least one of the self loops has at least 6 bends.

4.6 An Application

The drawing algorithm we presented, can draw the grid-layout with any specified external face. This is important since it enables us to produce a planar grid-layout, including external connections.

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

We consider constructing a planar grid-layout intended to be the internal layout of a big module. Ideally we would get the 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 cause the input to become 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. These vertices 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 figure 4.7(a). Also, the special modules will be drawn in the same rotation as they have in figure 4.7(a). This is possible since these rotations comply with the rules of selecting a rotation, and one 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 figure 4.7(a).

If the grid-layout 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 figure 4.7(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 figure 4.7(c).

Figure 4.7: 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
Chapter 5

Non-Planar Drawings

In this chapter we consider the non-planar versions of the problems we dealt with before. Namely, there is no requirement that the input or the output be planar. We allow edges to cross one another in the drawings. In this chapter, the fact that a drawing is called k-bend does not imply that the drawing is planar.

5.1 Vertices as Points

We call a graph with no self loops which has no vertex of degree greater than d a d-graph. Biedl [Bie93] describes a linear-time algorithm that finds a 2-bend drawing for every 4-graph.

Techniques similar to Biedl’s can be used to find in linear-time a 1-bend drawing for any simple 3-Graph.

It is trivial to find a 3-graph with no 1-bend drawing, and a 2-graph with no 0-bend drawing.

5.2 Vertices as Modules (Grid-Layouts)

In this section we consider the nonplanar versions of the problems we dealt with in chapter 4. 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 angle in which they are drawn may be determined arbitrarily. The input to the algorithm is a module description, and it includes the rotation in which each module is to be drawn. A linear-time drawing algorithm is described in figure 5.1. The algorithm is demonstrated in figure 5.2. The algorithm works by first placing the modules diagonally, so that if we continue the line of any terminal, it does
not hit any module. Next, we complete the drawing of the edges.

\begin{procedure}
\textbf{NPGL}(M);
\{ \text{M is a module description, with a rotation specified for each module}\}
\begin{algorithmic}
\Procedure{}{}
\EndProcedure
\begin{algorithmic}
\Procedure{NPGL}{M}
\EndProcedure
\end{algorithmic}
\end{procedure}

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{figure5.1}
\caption{construction of a 4-bend (nonplanar) grid-layout}
\end{figure}

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{figure5.2}
\caption{output of the nonplanar grid-layout algorithm}
\end{figure}

the edges are numbered in the order in which they are added by the algorithm.

\textbf{Theorem 5.1} Algorithm NPGL (see figure 5.1) 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 consists of one module which has one self-loop. The terminals of
the self-loop are on opposite sides of this module. Clearly, the self loop needs at least 4 bends. 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.

**Lemma 5.1** A module description that has two modules and sixteen edges, such that every side of one module is connected by one edge to every side of the other module, has no 3-bend grid-layout.

**Proof:** We look at any grid-layout of the module description and show an edge that must have at least 4 bends in this grid-layout. First, we look at the placement and the rotation of the two modules. The modules are both drawn non-overlapping on the grid, therefore the following holds: There is a side \( A \) of the first module and a side \( B \) of the second module, such that \( A \) and \( B \) are parallel, and the terminals exit \( A \) and \( B \) in opposite directions. Every side of the first module is connected by one edge to every side of the second module. Therefore, one edge connects \( A \) and \( B \), and it must have 4 bends in the grid-layout. \( \blacksquare \)
Bibliography


