7

Planar Orthogonal and Polyline Drawing Algorithms

Christian A. Duncan
Quinnipiac University

Michael T. Goodrich
University of California, Irvine

7.1 Introduction

One can assess the quality of a drawing of a graph in many different ways. Many important criteria deal with the aesthetics, readability, of the drawing. For example, the size of the drawing, roughly measured as the ratio between the farthest two objects of the drawings and the closest two, is a measure of how much information can be displayed at one time. The aesthetic that is of biggest concern in this chapter is that of *angular resolution*. Essentially, we are concerned with how close together edges that stem from the same vertex are to each other. The smaller the angle the more likely are the chances that the distinct edges become one. Clearly, a high-degree vertex, one with many edges extending out of it, will inevitably have a small angle between at least one pair of edges. So, the goal is to make the resolution determined to some extent by the degree of the vertex.

Optimizing angular resolution in drawings has been addressed by countless researchers. The two approaches we focus on in this chapter are to draw the graph orthogonally, that is using only vertical and horizontal line segments for the edges. Orthogonal drawings have the benefit that the smallest angle is at most $\pi/2$ and that the resulting graphs are often quite pleasing to the eye because of the few edge directions employed, but they also have the disadvantage that no vertex can have degree more than four. The study of orthogonal graphs also has the advantage of being of interest to VLSI design, because many wires are routed similarly. There are many different approaches to drawing orthogonal graphs. Early results draw the graph using few bends but sacrificed size or running-time efficiency. Improved techniques, involving computing a visibility representation, yielded orthogonal drawings in linear time with few bends and small size. By using network flows, we can draw (embedded) graphs with the guaranteed minimum number of bends possible in the smallest area allowable, but the run-time performance goes up to near quadratic time.

When using graphs containing vertices with degree more than four, one can no longer apply standard orthogonal drawing techniques. More general polyline drawing techniques, however, do exist. The goal is usually to focus directly on the sizes of angles created rather than the types of edges allowed. Thus, during the drawing, we can route edges in any orientation so long as the angle does not go below some fixed threshold. The most successful approaches all seem to work by taking a vertex and assigning exit ports, which are adequately spaced, such that edges are routed from the start vertex through distinct ports to the destination vertex. These techniques typically produce the layout by creating a canonical ordering on the vertices and adding the vertices into the drawing based on this ordering, while constantly maintaining the routing requirements of the edges. Using this approach, one can guarantee, for example, that a drawing can be made in linear time with good angular resolution, good size bounds, and using at most one bend per edge.

Before going into the details of the different approaches, we first present some basic terminology and general techniques in Section 7.2. Section 7.3 describes some standard approaches to drawing orthogonal graphs. Section 7.4 describes work done on more general polyline drawings. We conclude our chapter in Section 7.5 with a brief summary of the main results presented.

7.2 Preliminaries

We begin with a few basic definitions of some general graph terminology along with some more detailed descriptions of techniques useful for constructing drawings of graphs.

7.2.1 Definitions

Although common in nearly any book on graph algorithms, we borrow notation predominantly from [DETT99]. A *(simple) graph* $G = (V, E)$ is a finite set V of *vertices* and a finite set E of *edges*, where each edge is an unordered pair $e = (u, v)$ of vertices. A *multigraph* is a graph where the edges are multisets, that is two edges may have the same pair of vertices. For each edge $e = (u, v)$, we say that e is *incident* to u and v. We also say that u and v are *neighbors*. The *degree of a vertex* is the number of edges incident to it. The *maximum degree of a graph* is the maximum degree among all vertices in V. A *(simple) path p of G* is a sequence of distinct vertices of G, $(v_1, v_2, \ldots, v_k)$ such that for $1 \leq i < k$, $(v_i, v_{i+1}) \in E$. A *(simple) cycle c of G* is a path such that $v_1 = v_k$ with $k > 1$. A graph is *acyclic* if it has no cycles. A graph is *connected* if for every pair of vertices $u, v \in V$, there is a path from u to v. For any $k > 0$, a graph is *k-connected* if the removal of any $k - 1$ vertices from the graph still leaves the graph connected. We often refer to 2-connected graphs as *biconnected* and 3-connected graphs as *triconnected*.

We may also define many of our terms based on giving each edge a specific direction. A *directed graph* (*digraph*) is a graph where each *directed edge* $e = (u, v)$ is an ordered pair, where we consider u to be the *origin* and v to be the *destination* of the edge. In addition, $e = (u, v)$ is an *incoming edge* of v and an *outgoing edge* of u. The *indegree* of a vertex v is the number of its incoming edges, and the *outdegree* of a vertex v is the number of its outgoing edges. A *source* is a vertex with no incoming edges, i.e., with indegree 0. A *sink* is a vertex with no outgoing edges, i.e., with outdegree 0. A *directed path* of G is a path of G, $(v_1, v_2, \ldots, v_k)$, such that for $1 \leq i < k$, (v_i, v_{i+1}) is a directed edge in E. A *directed acyclic graph* (*DAG*) is a directed graph that has no cycles.

A *drawing* Γ of a graph $G = (V, E)$ is essentially a mapping of each vertex $v \in V$ to a distinct point $\Gamma(v)$ and of each edge $e = (u, v) \in E$ to a simple open Jordan curve $\Gamma(e)$, which has $\Gamma(u)$ and $\Gamma(v)$ as its endpoints. If G is directed, it is common to draw the edge with an arrow toward the destination vertex. When the drawing is understood from the context, we often leave out the Γ notation. For example, we may say that an edge e is made of horizontal and vertical segments rather than the drawing $\Gamma(e)$.

A *planar graph* is a graph G that admits a *planar drawing* Γ, a drawing with no edges intersecting, except for edges that share a common vertex v and only at that vertex. A *planar embedding*, or, simply, *embedding*, of a graph is the collection of (counter-clockwise) circular orderings of incident edges around every vertex induced by a planar drawing. A *plane graph* is a graph that has been associated with a specific planar embedding. A *maximal planar graph* is a graph where the addition of any edge $e \notin E$ causes the graph to be non-planar. Maximally planar graphs have the property that every face is a triangle, a cycle of three edges. For notation, we often refer to planar graphs with maximum degree k as *k-planar graphs*, in particular, we deal with many cases of 4-planar graphs.

A *straight-line drawing* of a graph is a drawing where every edge is a straight-line segment. A *polyline drawing* of a graph is a drawing Γ such that every edge $e = (u, v) \in E$ is represented as a connected sequence of line segments $\overline{p_1p_2}, \overline{p_2p_3}, \ldots, \overline{p_{k-1}p_k}$, where $p_1 = \Gamma(u)$ and $p_k = \Gamma(v)$ are the endpoints of the edge. We refer to $p_2, \ldots, p_{k-1}$ as *bend points* of the drawing of the edge. An *orthogonal drawing* of a graph is a polyline drawing where every edge is an alternating sequence of horizontal and vertical line segments. A *grid drawing* is a drawing of the graph where each vertex and each bend point has integer coordinate values, effectively being placed on an integer grid. The *area of a grid drawing* is the area of the smallest enclosing axis-aligned rectangle containing the drawing. For a given drawing of G, the *angular resolution of a vertex v* is the smallest angle between two distinct edges incident to v and the *angular resolution* of G is the minimum angular resolution among all vertices.

An *st-graph* is a DAG with one source and one sink. A *planar st-graph* is an *st*-graph that has a planar embedding with the source s and sink t located on the external face.

DEFINITION 7.1 Given a planar *st*-graph G, the *dual planar st-graph* $G^* = (V^*, E^*)$ is a digraph with the following properties:

- V^* is the set of faces in G with the addition that the external face $(s, \ldots, t, \ldots, s)$ is broken into two parts s^* representing the portion of the face from s to t and t^* representing the portion from t to s.
- For every edge $e \in E$, we have an edge $e^* = (f, g) \in E^*$ where f is the face to the left of e and g is the face to the right of e.

In the construction of an orthogonal drawing of a graph G discussed in Sections 7.2.3 and 7.3.1, the dual graph coupled with the following special ordering of vertices play a critical role in the creation of an intermediate visibility representation of G.

DEFINITION 7.2 Let $G = (V, E)$ be a directed acyclic graph. A *topological ordering* $T(G)$ is an assignment of integer values $T(v)$ to each vertex $v \in V$ such that for every directed edge $(u, v) \in E$, we have that $T(u) < T(v)$. The *size of the topological ordering* $s(T)$ is $\max_{v \in V} T(v) - \min_{u \in V} T(u)$. An *optimal topological ordering* $T^*(G)$ is a topological ordering with the smallest size, $s(T^*) = \min_{T(G)} s(T)$.

```
Require: G = (V, E) be a Directed Acyclic Graph
Ensure: T(G) is an optimal topological ordering
  {Compute the indegree for every vertex}
  for all v ∈ V do
    in(v) ← 0
  end for
  for all (u, v) ∈ E do
    increment in(v)
  end for
  {Identify all sinks}
  S_0 ← ∅
  for all v ∈ V do
    if in(v) = 0 then
      S_0.add(v)
    end if
  end for
  n ← 0
  repeat
    {Mark all current sinks and remove them from G}
    S_{n+1} ← ∅
    for all v ∈ S_n do
      T(v) ← n
      for all (v, u) ∈ E do {remove v from the graph}
        decrement in(u)
        if in(u) = 0 then {u is a new sink}
          S_{n+1}.add(u)
        end if
      end for
      increment n
    end for
  until S_n is empty {No more sinks}
```

Figure 7.1 Algorithm for computing an optimal topological ordering of a DAG.

In our definition, it is possible for two vertices u and v to have the same value if there is no directed path between u and v. Note, this is basically a partial ordering where the optimal size is the length of the longest chain in the partial order. Topological orderings are discussed in most standard graph and algorithms textbooks. See, for example, [CLR90, GT02].

Computing an optimal topological ordering in linear time is fairly straightforward. We assign every sink vertex a number 0, remove these vertices and their edges from the graph, and repeat the process with a number one larger until there are no vertices left. Figure 7.1 describes the process in more detail.

This common algorithm proves useful for the construction of orthogonal graphs via a visibility representation. However, there are other more difficult, but equally useful, orderings. We next discuss one such ordering, the canonical ordering.

7.2.2 Canonical Ordering and Shifting Sets

In [dFPP90], de Fraysseix, Pach, and Pollack describe a technique for embedding a plane graph on a grid. Their technique uses an incremental approach that is built around a particular ordering of the vertices known as a canonical ordering. Initially defined for maximal plane graphs, Kant [Kan96] later extended it to triconnected plane graphs and Gutwenger and Mutzel [GM98] to biconnected plane graphs. In this section, we define and describe the canonical ordering of [dFPP90, CDGK01] as well as the shifting sets derived from this ordering, which are needed in the polyline drawing method described in Section 7.4.

DEFINITION 7.3 Let G be a maximal plane graph on m vertices. Let $\pi = (v_1, v_2, \ldots, v_n)$ be an ordering of the vertices of G. For $1 \leq k \leq n$, let G_k be the plane subgraph of G induced by the vertices of $v_1, \ldots, v_k$ and let $C_k = (v_1 = w_1, w_2, \ldots, w_m = v_2)$ be the cycle forming the external face of G_k. We call π a *canonical ordering* of G if

1. v_1, v_2, and v_n are the external vertices of G in counter-clockwise order,
2. for $2 < k < n$, G_k is 2-connected and internally maximal, i.e., every internal face is a triangle, and
3. for $2 < k < n$, v_k is a vertex of C_k and has at least one neighbor in $G - G_k$.

de Fraysseix, Pach, and Pollack [dFPP90] prove the following theorem, which was later extended to triconnected plane graphs by Kant [Kan96]:

Theorem 7.1 *Every maximal plane graph has a canonical ordering that can be found in linear time and space.*

The canonical ordering has the property that all of the neighbors of v_{k+1} in G_{k+1} lie on C_k. Intuitively, the ordering is constructed in reverse order by starting with the initial external triangular face and repeatedly removing a vertex $v_{k+1} \notin \{v_1, v_2\}$ that has at most two neighbors on C_{k+1} creating the new graph G_k and external face C_k. See Figure 7.2.

Once constructed, the canonical ordering π leads to an incremental approach for constructing a drawing of G. Here, we start with the triangle v_1, v_2, v_3 and repeatedly add the next vertex v_{k+1} to the graph of G_k by adding edges for v_{k+1} to its neighbors in C_k forming G_{k+1} and C_{k+1}. The vertices of C_k that are no longer on C_{k+1} are said to be *covered* by v_{k+1}. Since the neighbors of v_{k+1} are all continuous on the cycle C_k, we can label them as $w_l, w_{l+1}, \ldots, w_r$. We refer to the two vertices w_l and w_r as the leftmost and rightmost neighbors of v_{k+1} in C_k. Since all the neighbors of v_{k+1} except the leftmost and rightmost neighbors are covered by v_{k+1}, we know that the cycle $C_{k+1} = (v_1 = w_1, w_2, \ldots, w_l, v_{k+1}, w_r, \ldots w_m = v_2)$. See Figure 7.3.

Starting with de Fraysseix, Pach, and Pollack, several authors have used this canonical ordering (or a variant) to build a graph incrementally. However, to place the vertices effectively, onto a grid location for example, one must also repeatedly shift the vertices in G_k to create a proper location for v_{k+1}. Typically, the approach is to increase the space between the leftmost and rightmost neighbors of v_{k+1}. However, shifting these two vertices also forces other vertices to shift to avoid creating edge crossings.

To solve the problem of determining which vertices must shift together, we also define a shifting set associated with each vertex on the current external face. See Cheng et al. [CDGK01].

DEFINITION 7.4 For a given canonical ordering $\pi = (v_1, v_2, \ldots, v_n)$, we define for $3 \leq k \leq n$, the *shifting set* $M_k(w_i) \subseteq V$ for each vertex $w_i \in C_k$ on the external face of G_k as follows. $M_3(v_3) = \{v_3\}, M_3(v_2) = M_3(v_3) \cup \{v_2\}, M_3(v_1) = M_3(v_2) \cup \{v_1\}$. For $3 \leq k < n$, let w_l and w_r be the leftmost and rightmost neighbors of v_{k+1} in C_k. Then,

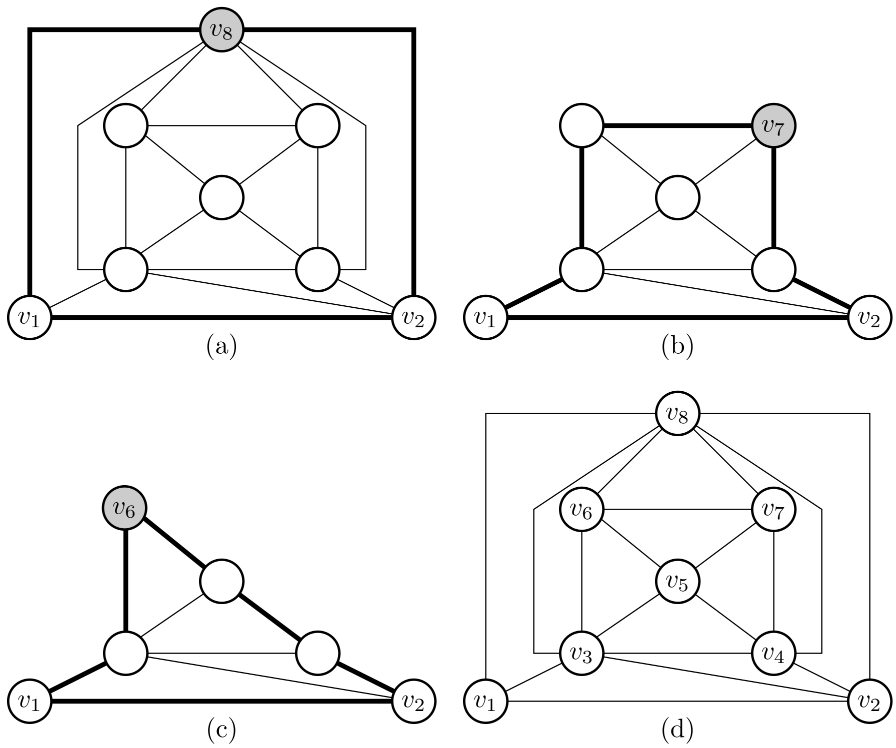


Figure 7.2 An illustration showing the creation of the canonical ordering in reverse order. (a) The first vertex v_8 is about to be removed with the external cycle highlighted. (b) Removal of the next vertex, v_7. (c) Removal of vertex v_6. (d) The final canonical ordering of the vertices.

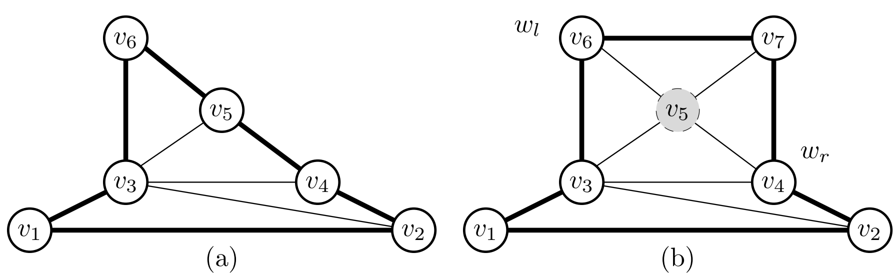


Figure 7.3 Inserting a vertex using the canonical ordering. This example does not follow the vertex placement techniques employed by the standard algorithms used to produce good area drawings. (a) The graph G_6, with its external cycle C_6 drawn in bold. (b) The graph G_7 after inserting vertex v_7. The covered vertex v_5 is lightened. The leftmost and rightmost neighbors are $w_l = v_6$ and $w_r = v_4$. The new external cycle C_7 is therefore $(v_1, v_3, v_6, v_7, v_4, v_2)$.

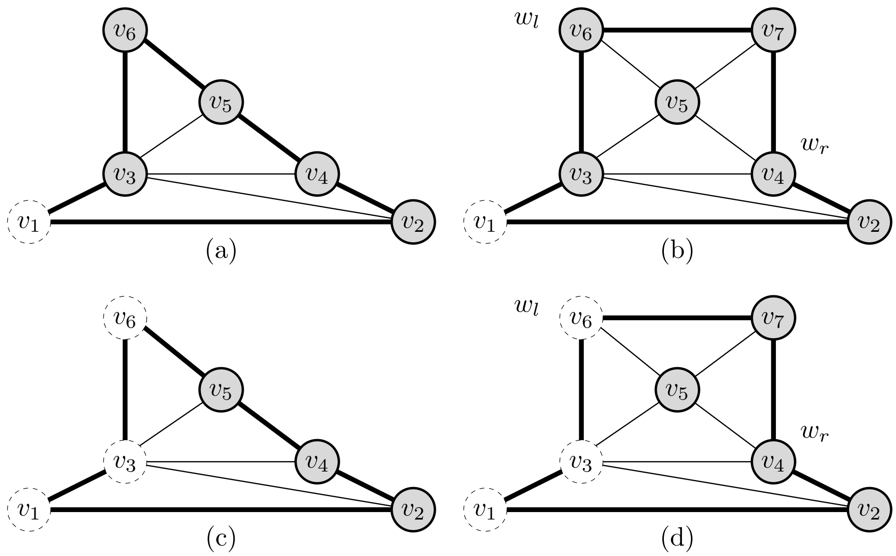


Figure 7.4 The incremental construction of a shifting set. The vertices for each set shown are highlighted. (a) The shifting set for $M_6(v_3)$. (b) After inserting v_7, the shifting set for $M_7(v_3)$. This simply merges in the new vertex. (c) The shifting set for $M_6(v_5)$. (d) After inserting v_7, the shifting set for $M_7(v_7)$. Since $w_{l+1} = v_5$, this set is the union of $M_6(v_5)$ and v_7.

- for $i \leq l$, $M_{k+1}(w_i) = M_k(w_i) \cup \{v_{k+1}\}$,
- for $j \geq r$, $M_{k+1}(w_j) = M_k(w_j)$, and
- $M_{k+1}(v_{k+1}) = M_k(w_{l+1}) \cup \{v_{k+1}\}$.

From this definition, one can show that the following properties of the shifting set hold for all $3 \leq k \leq n$ for the incremental drawing algorithms described in Section 7.4:

1. $w_j \in M_k(w_i)$ iff $j \geq i$,
2. $M_k(w_1) \supset M_k(w_2) \supset \cdots \supset M_k(w_m)$,
3. For $1 \leq i \leq m$ and a planar drawing of G_k, if we shift all vertices in $M_k(w_i)$ by distance $\delta_i \geq 0$ to the right, then the resulting drawing of G_k remains planar.

In other words, the shifting set for a vertex w_i on the external face is just the set of all vertices that need to be shifted to the right to maintain planarity if w_i is shifted to the right.

Note that $M_{k+1}(w_i)$ is undefined for $l < i < r$, since these covered vertices are no longer on the external face. See Figure 7.4.

A careful examination of the set reveals that a vertex w_i that is covered by v_{k+1} shifts by δ units if and only if v_{k+1} shifts by δ units. That is, for $k' > k$, $w_i \in M_{k'}(v)$ iff $v_{k+1} \in M_{k'}(v)$. This property of the shifting set is exploited during the incremental embedding algorithms that use a canonical ordering to ensure that shifts do not produce crossings.

7.2.3 Visibility Representations

Orthogonal drawings, and even general drawings, of planar graphs often start by computing a visibility representation of the graph. Before going into the details of using a visibility representation to compute an orthogonal drawing, presented in Section 7.3.1, we first explain the general approach of computing such a representation.

DEFINITION 7.5 Given a graph $G = (V, E)$, a *visibility representation* Γ, for G maps every vertex $v \in V$ to a horizontal *vertex segment* $\Gamma(v)$ and every edge $(u, v) \in E$ to a vertical *edge segment* $\Gamma(u, v)$ such that each vertical edge segment $\Gamma(u, v)$ has its endpoints lying on the horizontal vertex segments $\Gamma(u)$ and $\Gamma(v)$ and no other segment intersections or overlaps occur.

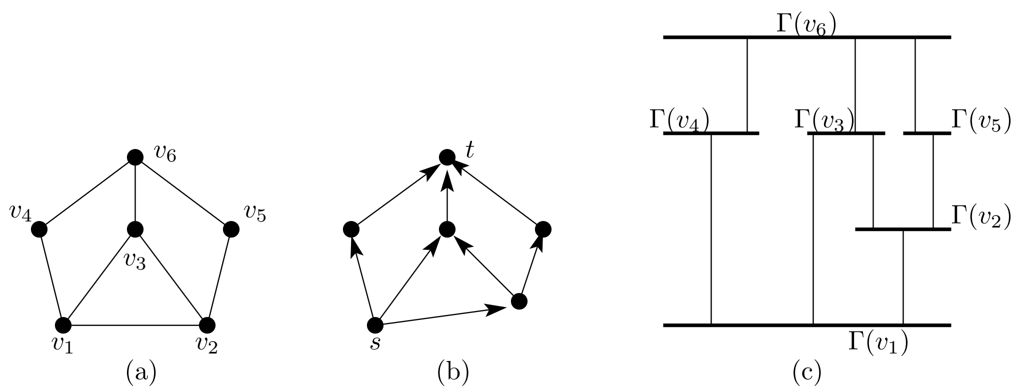


Figure 7.5 (a) A simple graph G (b) An st-ordering of G (c) A visibility representation of G.

See Figure 7.5 for one example of a visibility representation. Otten and van Wijk [OvW78] introduced the visibility representation. With varying improvements, several researchers have proved that every planar graph has such a representation, which can be found in linear time [OvW78, DHVM83, RT86, TT86]. In general, we have the following theorem about computing a visibility representation:

Theorem 7.2 *[TT86] A graph admits a visibility representation if and only if it is planar. Furthermore, a visibility representation for a planar graph can be constructed in linear time.*

Figure 7.6 describes an algorithm to compute the visibility representation of a given graph. After making the graph biconnected by adding dummy edges [FM98], we compute an st-ordering on the graph creating a planar st-graph and its dual graph G^*. The location of the vertex-segments and edge-segments are then determined by a topological ordering of the st-graph and its dual with the former serving to determine y-values and the latter to determining x-values. Figure 7.7 shows an example construction.

Require: $G = (V, E)$ be a plane graph
Ensure: Γ is a visibility representation of G on the integer grid of size $O(n^2)$
Make G biconnected by adding "dummy" edges {See [FM98]}
Select an edge (s, t) on the external face
Compute a planar *st*-graph on G {For simplicity, we refer to it as G}
Create the dual planar *st*-graph G^*
Compute the optimal topological ordering $T_x = T(G^*)$ {See Figure 7.1}
Compute the optimal topological ordering $T_y = T(G)$
for all $v \in V$ **do** {Assigning positions to the horizontal vertex segments}
  Let f_l be the face to the left of the leftmost outgoing edge of v
  Let f_r be the face to the right of the rightmost outgoing edge of v
  $\{f_l$ and f_r are vertices in the dual graph $G^*\}$
  $\Gamma(v).y \leftarrow T_y(v)$
  $\Gamma(v).\mathsf{xmin} \leftarrow T_x(f_l)$
  $\Gamma(v).\mathsf{xmax} \leftarrow T_x(f_r) - 1$
end for
for all $e = (u, v) \in E$ **do** {Assigning positions to the vertical edge segments}
  Let f_l be the face to the left of e $\{f_l$ is a vertex in $G^*\}$
  $\Gamma(e).x \leftarrow T_x(f_l)$
  $\Gamma(e).\mathsf{ymin} \leftarrow T_y(u)$
  $\Gamma(e).\mathsf{ymax} \leftarrow T_y(v)$
end for
Remove any added "dummy" edges

Figure 7.6 Algorithm for constructing a visibility representation of a plane graph.

7.2.4 Network Flows

Network flows, useful in many areas of graph theory and graph drawing, are particularly useful in finding drawings of orthogonal graphs with a minimum number of bends. We describe this use in Section 7.3.2. Beforehand, we discuss the general structure of a network flow, borrowing notation from Goodrich and Tamassia [GT02].

A *(single-source single-sink) flow network* N is a connected directed graph of *arcs* and *nodes*[1] with the following properties:

- Each arc e has a positive integer *capacity* $c(e)$ and a nonnegative integer *cost* $w(e)$;
- There exists a *source* node, s, such that s has no incoming arcs;
- There exists a *sink* node, t, such that t has no outgoing arcs;
- All other *non-terminal* nodes have at least one incoming and one outgoing arc.

Figure 7.8(a) shows one particular flow network. The network is viewed as transporting some *commodity* from the source to the sink by flowing along the arcs. A *flow* f for some network N is an assignment to each arc e of some (integer) *flow value* $f(e)$ such that the following two rules apply:

- *Capacity rule*: The (positive) flow for each arc does not exceed the capacity. For each arc $e \in N$, $0 \leq f(e) \leq c(e)$.

[1]We use the terms *arc* and *node* for a flow network instead of the analogous terms edge and vertex to help differentiate between a flow network and a graph, which is to be drawn using the flow network.

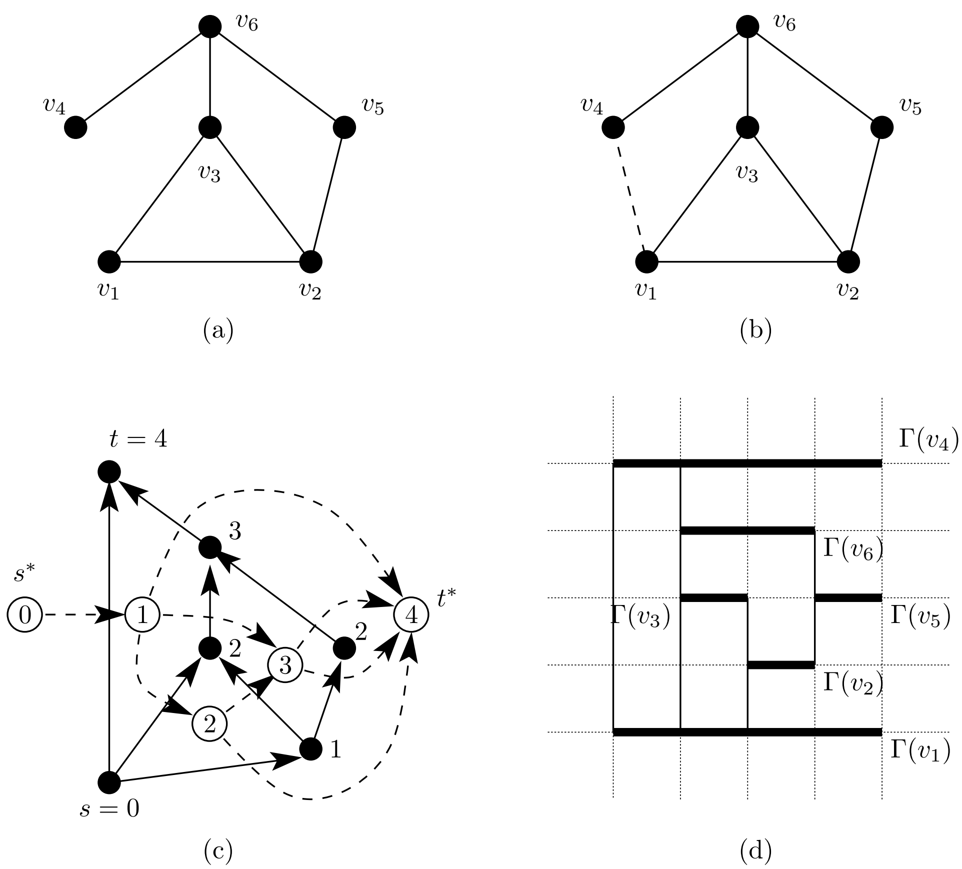


Figure 7.7 (a) A simple graph G. (b) G after augmenting to make it biconnected. (c) The st-planar graph of G (solid) and the dual graph G^* (dashed). The two topological orderings from these graphs are shown labeled by their nodes. (d) The visibility representation of G computed from these orderings.

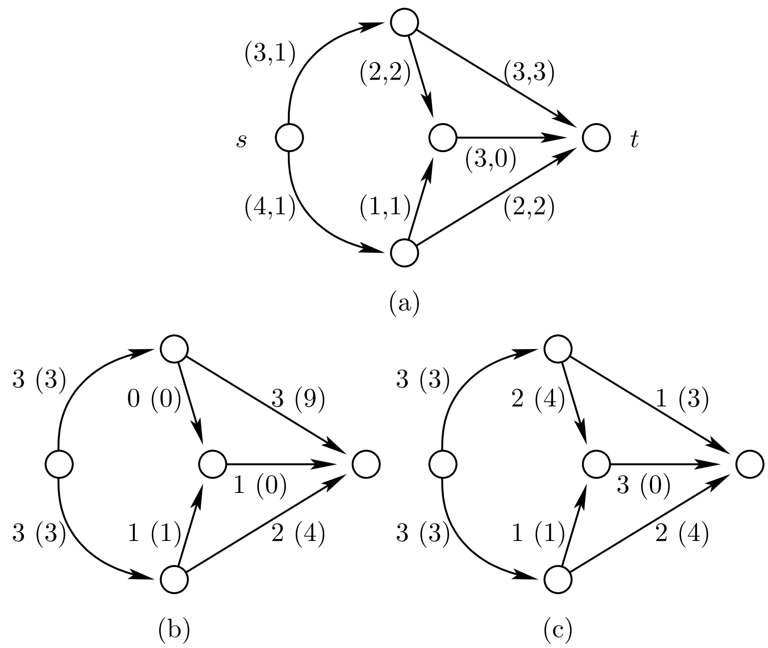


Figure 7.8 (a) A (single-source single-sink) flow network N with arcs labeled with the pair $(c(e), w(e))$ (capacity, cost). (b) A maximum flow of value 6 for flow network N. Each arc of N is labelled with its flow and, in parentheses, the cost of the flow on that arc. The total cost of this flow is 20. (c) A minimum-cost maximum flow for N. Note, the value of this flow is still 6 but the cost is now 18.

- *Conservation rule*: The flow coming *in* to a non-terminal node is the same as the flow going *out* of the node.
  For each non-terminal node $v \in N$, with $v \neq s, t$,

$$\Sigma_{e \in \mathsf{inarc}(v)} f(e) = \Sigma_{e \in \mathsf{outarc}(v)} f(e).$$

The *value* of the flow $v(f)$ is the total flow leaving the source node, which because of the conservation rule is the same as the flow entering the sink node. That is, $v(f) = \Sigma_{e \in \mathsf{outarc}(s)} f(e)$. For a given flow f, the cost of the flow on a given arc e is the cost of the arc $w(e)$ times the amount of flow on that arc $f(e)$. The *cost* of the flow $w(f)$ is the sum of the costs of each arc. That is, $w(f) = \Sigma_{e \in N} w(e) f(e)$.

The *maximum flow problem for* N is to find a flow f^* with maximum value among all possible flows of N. The *minimum-cost flow problem for* N is to find the minimum cost flow among all possible maximum flows in N. Figure 7.8 shows a maximum flow that does not have minimum cost as well as a minimum-cost maximum-flow solution.

There are several methods for solving flow networks, which are beyond the scope of this chapter. Their running times often depend on combinations of the number of nodes in the network, the capacity of the edges in the network, and the cost of the edges in the network. For details, see [CLR90, GT02].

Of particular relevance are minimum cost flow algorithms with running time that depends on the value of computed flow [CK12, GT97]. We use such an algorithm in Section 7.3.2 to compute a planar orthogonal drawing with the minimum number of bends.

7.3 Orthogonal Drawings

One highly effective way to draw graphs with good angular resolution is to use only edges that are rectilinear, or orthogonal. Such edges consist of alternating sequences of vertical and horizontal segments. In graph representations where each vertex is a point and where two edges are not allowed to overlap, a necessary condition for a graph to have an orthogonal drawing is that the maximum vertex degree be at most four. However, the introduction of rectangular regions for vertices allows for larger graph degrees.

7.3.1 Orthogonal Drawings from Visibility Representations

Given a 4-planar graph $G = (V, E)$, one can construct a good *orthogonal drawing* using the *visibility representation* discussed in Section 7.2.3. The following theorem is due to Tamassia and Tollis [TT89]:

Theorem 7.3 *Let G be a 4-plane graph. If G is biconnected, there exists an orthogonal grid drawing of G using $O(n^2)$ area with at most $2n + 4$ bends and where only two edges have more than two bends. If G is connected, the number of bends is $2.4n + 2$ and no edge has more than four bends.*

The version of the algorithm used to prove this theorem uses a *constrained visibility representation*. The additional constraint is that each (horizontal) vertex segment other than the source and sink have two (vertical) edge segments incident to its leftmost endpoint, with one being above and the other below the vertex segment. We describe the simpler, but slightly less effective, algorithm that uses a regular visibility representation. First, we compute a visibility representation $\Gamma(G)$. For each vertex $v \in V$, place the vertex at a single point on the horizontal vertex segment $\Gamma(v)$, determined below. The routing of the edges incident to v and the location of v on the vertex segment are based on various cases. Since each vertex has at most 4 incident edges and accounting for symmetry and subcases with smaller vertex degrees, Figure 7.9 shows the six possible cases along with the resulting edge routings and vertex placements. A careful study of the cases shows that no edge has more than two bends per endpoint, resulting in no more than four bends total. This creates an orthogonal shape, discussed in the next section, for G. To help improve the size and number of bends one can do a few heuristics to straighten out various edges. Finally, using the compaction technique described in the next section or similar more efficient techniques, one can convert the orthogonal shape into an orthogonal drawing using the smallest area. Figure 7.10 shows an example of an orthogonal drawing constructed from a visibility representation.

7.3.2 Network Flow Algorithms

Tamassia [Tam87] showed that by using a network flow algorithm one could construct orthogonal drawings of embedded 4-planar graphs with a minimum number of bends.

The fact that the graph is given with its embedding is significant. Formann et al. [FHH$^+$93] and Garg and Tamassia [GT01] showed that the problem of determining whether a drawing with no bends exists is NP-hard for 4-planar graphs. The strategy in their proof deals with the difficulty of assigning an order of the edges around vertices of degree 4. It is interesting to note that the problem is polynomial when the maximum degree is 3 [DLV98].

Tamassia's algorithm originally ran in $O(n^2 \log n)$ time. However, an improvement for certain types of planar flow networks (see Section 7.2.4) presented by Garg and Tamas-

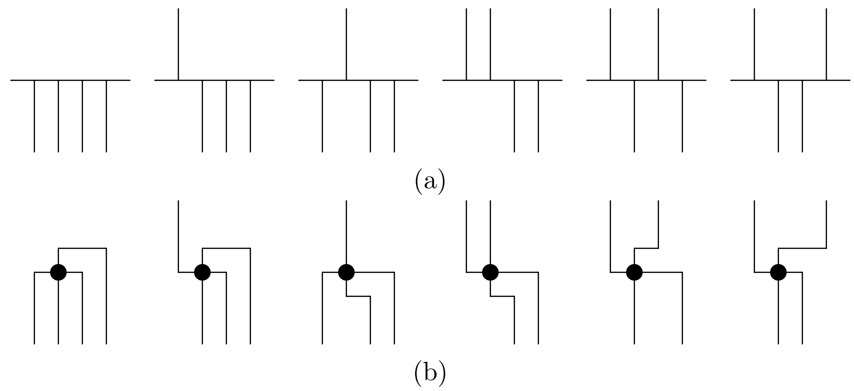


Figure 7.9 (a) The six possible cases for horizontal vertex segments intersecting with its 4 incident vertical edge segments in a visibility representation, accounting for symmetry. (b) The vertex placement and edge routings for each of the cases.

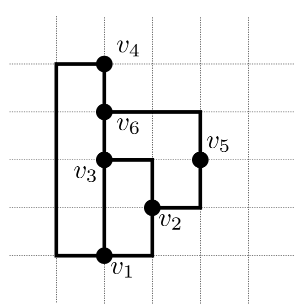


Figure 7.10 An orthogonal drawing from the visibility representation of Figure 7.7.

sia [GT97] reduced the running time to $O(n^{7/4}\sqrt{\log n})$. Recently, Cornelsen and Karrenbauer obtained a running time of $O(n^{3/2} \log n)$ [CK12].

Let $G = (V, E)$ be an embedded planar graph having maximum degree 4. We can compute a drawing of G with the minimum number of bends in two phases. First, we compute an *orthogonal shape* for G. Here we only define the bends of the edges and angles between adjacent edges at a vertex of G. In the second phase, we assign integer lengths to the edge segments of the orthogonal shape.

By transforming the first phase into a network flow problem, we are able to compute the required drawing's orthogonal shape. In this network, the commodities are the angles between adjacent edges. Each unit of flow in the network is associated with a right angle in the orthogonal shape, originating from the vertices, flowing across the faces by the edge bends, and ultimately sinking at the faces. Since this interpretation leads to a multi-source, multi-sink flow, we actually create a dummy source and sink that connect to the respective nodes. For simplicity, we allow certain arcs to have a *lower bound* in addition to a capacity. This is easily incorporated into the algorithms for the original flow network.

We want each vertex v to supply 4 units of flow and to have the faces consume these units. Here, 4 "units" correspond to a 2π angle. Let $d(f)$, the *degree of a face* f in the graph G, be the length of the cycle bounding face f. If the graph is not biconnected, an edge may be counted twice on the same face. The *consumption rate* of each face is designated by $\sigma(f)$ with

$$\sigma(f) = \begin{cases} 2d(f) - 4 & \text{if } f \text{ is an internal face} \\ 2d(f) + 4 & \text{if } f \text{ is the external face} \end{cases}$$

From Euler's formula, we know that $\Sigma_f \sigma(f) = 4n$, which is the total number of units supplied by the vertices. Our network N has three types of nodes and four types of arcs with the following described attributes:

- Non-terminal nodes correspond to the vertices and faces of G;
- A source node s and sink node t serve to supply and consume the commodity;
- For every vertex v, arcs of type (s, v) with a capacity of 4, cost 1, and lower bound 4 act to supply the vertex v with its commodity;
- For every face f, arcs of type (f, t) with a capacity of $\sigma(f)$ and cost 1 act to consume the commodity from the face vertices;
- From every face f and every vertex v on the cycle of f, we use an arc of type (v, f) with a capacity of 4, cost 1, and lower bound 1. This arc flow represents the angle at vertex v in face f;
- For every pair of faces f and g sharing an edge, we designate an arc of type (f, g) having a capacity of $+\infty$, cost 1, and lower bound 0. This arc flow represents the number of bends along edge e with the right angle *inside* of the face f.

Figure 7.11 shows a detailed example of a 4-planar graph, its network model, and the minimum cost solution. We now take a closer look at an interpretation of the network from the source side. At every vertex v the network supplies the vertex with 4 units, all of which must, by the conservation rule, flow across the (v, f) arcs. Since each unit corresponds to $\pi/2$ radians, this guarantees that the sum of the angles around a vertex, which is equivalent to the sum of the flow leaving v along these arcs, is 2π.

From the sink side, by the conservation rule, we know that the sum of the units at the vertices and the bends of a face is equal to $2d(f) - 4$ units for an internal face and $2d(f) + 4$

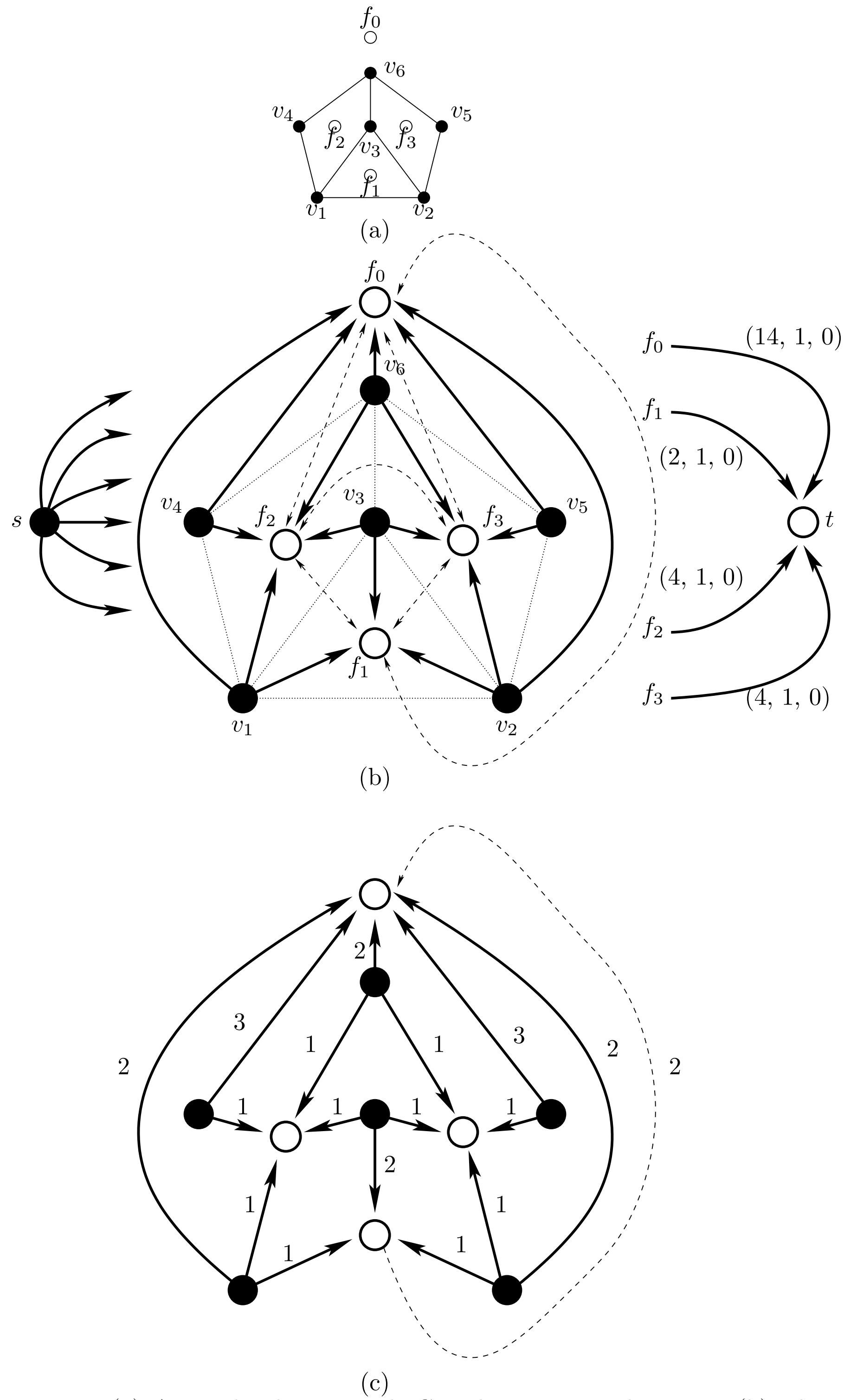


Figure 7.11 (a) A simple planar graph G with maximum degree 4. (b) The network N associated with G. The arcs from the source s to the vertex nodes have label $(4, 1, 4)$, i.e., capacity 4, cost 1, and a lower bound of 4. The vertex to face arcs are drawn as solid lines with label $(4, 1, 1)$. The face to face arcs are drawn bi-directional with both directions having label $(+\infty, 1, 0)$. (c) A minimum cost max flow with the arc labels reflecting the flow. Some vertices are omitted and some edges are partially drawn for better readability.

for the external face. Again, since each unit corresponds to $\pi/2$ radians, we know the sum of these angles is equal to $\pi(d(f) - 2)$ for an internal face and $\pi(d(f) + 2)$ for the external face. Thus, each face is properly closed, and we can see that any valid flow ϕ on the network corresponds to a proper orthogonal shape for G.

We now interpret the cost associated with a specific flow. For arcs of type (s, v) the cost is 1 and the flow is fixed. So, for this case, the total cost is exactly $4n$. Similarly, all the arcs of type (f, t) have cost that sum to exactly $4n$. Since all the arcs of type (v, f) have to release the commodity sent from the source s, we know that the sum of these arcs is also $4n$. Finally, the arcs of type (f, g) represent the number of bends for the given edge with each bend costing one unit. Therefore, the total cost of the flow is $12n + B$, where B is the total number of bends in the orthogonal shape represented by the flow. Since $12n$ is fixed for all flows along the same network, minimizing the cost of the flow corresponds to minimizing the number of bends in the orthogonal shape.

In the second phase, we take this orthogonal shape and determine a compact drawing for the actual graph. Since each bend for an edge switches between horizontal and vertical lines, our strategy is to determine the (integer) lengths of these line segments. We do this by computing the lengths of the horizontal segments independently of the vertical segments. We shall explore the vertical computation as the horizontal one is analogous.

We can compute the length of each vertical segment by, once again, using a network flow model. However, this flow model assumes that the faces are all rectangular. Therefore, we first split the faces into rectangular faces by converting bend points into dummy vertices and inserting dummy edges where necessary. This process is described in detail in Chapter 5 of [DETT99]. We therefore explain the solution for when we have an orthogonal representation where each face is a rectangle, referring to this modified graph as G'. In this case, our model has three types of nodes and three types of arcs.

- A source node s and sink node t serve to supply and consume the commodity and also represent the “left” and “right” regions of the external face;
- Non-terminal nodes correspond to the faces of G';
- For every pair of faces f and g sharing a *vertical* edge segment, with f to the left of g, we designate an arc of type (f, g), with capacity $+\infty$, cost 1, and lower bound 1. The arc flow represents the length of this vertical segment.

Figure 7.12 illustrates an example of computing a compact orthogonal drawing using this network flow approach. Since the source node s (and similarly sink node t) represents the entire left vertical border of the final drawing and the flow leaving s corresponds to the height of this border, the flow value is exactly the height of the drawing. In addition, the cost of the flow is equal to the total length of all vertical segments in the drawing. Similarly, the horizontal flow model computes the width of the drawing and the total length of all horizontal segments. By solving the minimum-cost minimum-flow problem for both vertical and horizontal networks, we can create an orthogonal drawing of G with the minimum height, width, area, and total edge length. Observe that the flow here is the smallest flow that meets the lower bound requirements for each arc.

Using their improved network flow algorithm, Cornelsen and Karrenbauer proved the following result, which improves the running time of the original algorithm by Tamassia [Tam87]:

Theorem 7.4 *[CK12] Let G be an embedded 4-planar graph with n vertices. A planar orthogonal drawing of G with the minimum number of bends can be computed in $O(n^{3/2} \log n)$ time.*

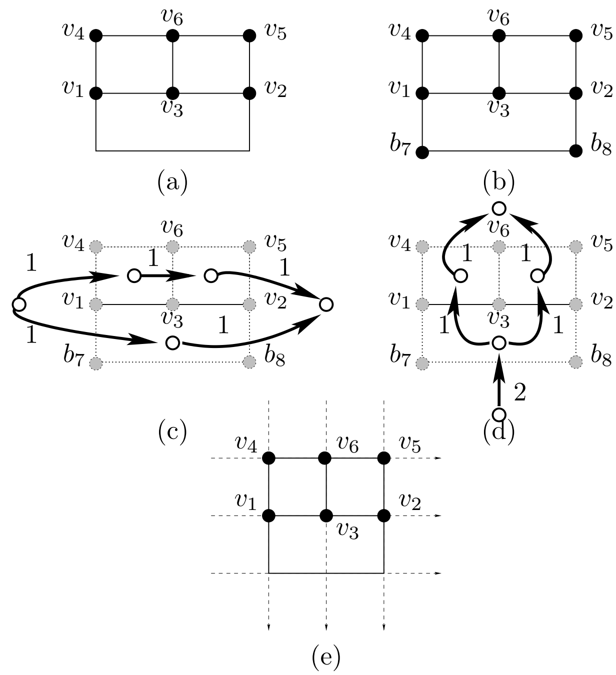


Figure 7.12 (a) An orthogonal drawing with the orthogonal representation described by Figure 7.11c. (b) The same drawing with the two bend points temporarily converted to vertices so that each face is rectangular. (c) The network flow for computing the vertical segments along with the solution. (d) The network flow for computing the horizontal segments along with the solution. (e) The final compact solution with the horizontal and vertical segments determined from the two flows and the inserted dummy vertices removed.

7.4 Polyline Drawings

When one wishes to draw planar graphs having maximum degree more than 4 with good angular resolution and with vertices as single points, clearly orthogonal drawings do not suffice. There have been various other approaches to creating planar polyline drawings with good angular resolution, many of these results extend the work of Kant [Kan96], including work by Goodrich and Wagner [GW00], Gutwenger and Mutzel [GM98], Cheng et al. [CDGK01], and Duncan and Kobourov [DK03]. The general approach is to use an incremental insertion method to add vertices one at a time using a canonical ordering and continually maintain the proper angular resolution qualities and other specific restrictions.

7.4.1 Mixed-Model Algorithm

The approach of Gutwenger and Mutzel [GM98] is similar to the approaches taken by [GW00, CDGK01, DK03], which are discussed in the next subsection. However, unlike those approaches which rely on the graph being either maximal, tri-connected, or having artificial edges added to make them maximal, the approach by Gutwenger and Mutzel uses an ordering that is defined for biconnected graphs. The benefits are significant in the sense that such artificial edges, once removed, often create unexpected artifacts. In their *mixed-model algorithm*, they take a given biconnected plane graph $G = (V, E)$, and using this new ordering, assign for each edge $e \in E$, an *inpoint* $e_{\text{in}} = (x_{\text{in}}, y_{\text{in}})$ and an *outpoint* $e_{\text{out}} = (x_{\text{out}}, y_{\text{out}})$. Then each edge $e = (v, w)$ is drawn as a polyline edge. Route the edge from v to w in the following manner:

- from v to e_{out},
- from e_{out} vertically to point $b = (x_{\text{out}}, y_{\text{in}})$,
- from b horizontally to e_{in},
- and finally to w.

This approach results in quite aesthetically pleasing graphs that combine a mixture of good angular resolution via general direction edges and orthogonal edges. However, the results require, in general, three bends per edge. The next section describes a technique that achieves similar results but with only one bend per edge.

7.4.2 One Bend Algorithm

Building off previous work by Kant [Kan96], Goodrich and Wagner [GW00], and Cheng et al. [CDGK01], Duncan and Kobourov [DK03] use an incremental insertion approach to create a planar polyline drawing with the following key properties:

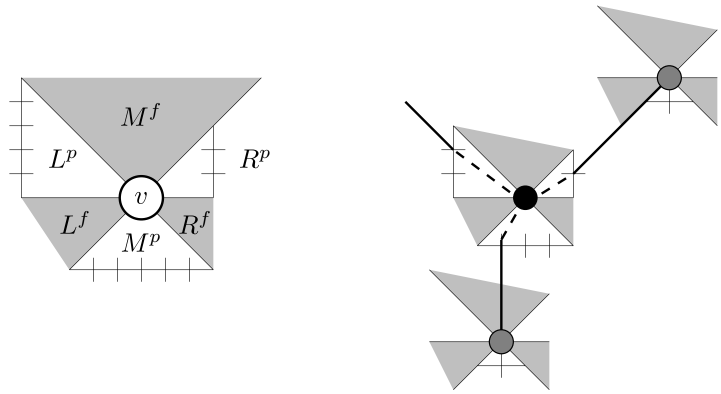
- each edge is drawn with at most one bend;
- each vertex v has angular resolution $\Theta(1/d(v))$;
- all vertices and bend points lie on an $O(n) \times O(n)$ grid.

The incremental approach uses the canonical ordering and the shifting set described in Section 7.2.2.

7.4.3 Vertex Regions

In [dFPP90], de Fraysseix, Pach, and Pollack present an algorithm to draw an n-vertex plane graph with straight-line edges on an $O(n) \times O(n)$ integer grid. Chrobak and Payne [CP95] show how to implement the algorithm in linear time. In this algorithm, each new vertex v_{k+1} is inserted above its neighbors $w_l, \ldots, w_r$, and after proper shifting, edges are drawn as straight-line segments from the location of v_{k+1} to each neighbor of v_{k+1}. In the approach used in [GW00, CDGK01], each vertex is associated with a diamond-shaped region where edges are routed through ports along the boundary of the region before connecting to the vertices. This creates bends in the edges but allows better control over the angles that are formed by the edges around vertices. To reduce the overall grid size, Duncan and Kobourov [DK03] use slightly altered vertex regions. Each vertex is surrounded by six vertex regions of two types, free regions and port regions, which alternate around the vertex. The regions are bounded by rays extending from v in various directions, with $0°$ indicating a positive vertical direction. See Figure 7.13.

DEFINITION 7.6 Let $v \in V$ have degree $d = d(v)$. The *vertex regions* associated with v are of two types, *free regions* and *port regions*. Free regions have the property that only one edge extends from v to another vertex through that region. Port regions are bounded on one side by a horizontal or vertical line segment with a number of (integer coordinate) ports, and each edge going through a port region of v from v to any other vertex passes through a unique port. Moreover, every edge is drawn as two line segments. The first, starting at one endpoint v, connects to a port in the port region of v, and the second connects from



(a) (b)

Figure 7.13 (a) The vertex regions around a particular vertex v. Notice that each port region can have a different number of ports. (b) Edges extending from a (darkened) vertex. The port edge segment is drawn dashed and the free edge segment is drawn solid.

that port to the other vertex w passing through one of w's free regions. The six regions associated with v are defined as follows:

- Free region M^f lies between $-45°$ and $45°$;
- Free region R^f lies between $90°$ and $135°$;
- Free region L^f lies between $-135°$ and $-90°$;
- Port region M^p lies between L^f and R^f;
- Port region L^p lies between L^f and M^f; and
- Port region R^p lies between R^f and M^f.

The algorithm proceeds similar to the standard embeddings that use the canonical ordering. In particular, one starts with an initial face v_1, v_2, v_3 and then repeatedly inserts the next vertex v_{k+1} by finding its leftmost and rightmost neighbors, w_l and w_r, on the current external face shifting the space between these vertices so that the lines connecting v_{k+1} to w_l and w_r intersect at a grid location. To ensure good angular resolution, one must introduce some bends, which requires a slight alteration in the approach.

Except for the initial horizontal edge (v_1, v_2), we route each edge (v_i, v_j) through a port of one of the two vertices. In the process, each edge consists of two edge segments. One segment, the *port segment*, extends from v_i to one of v_i's ports, lying entirely in one of v_i's port regions. The other, *free segment*, extends from this port to v_j passing through one of v_j's free regions. See Figure 7.13(b).

The ports are arranged in such a way that the angle between successive ports and v is $O(1/d(v))$. By Definition 7.6, since for every vertex v each free segment associated with v lies inside a free region boundary, each free region has exactly one free segment passing through it, each port segment associated with v lies inside a port region and passes through a unique port, the resulting angular resolution at v is $O(1/d(v))$. For compactness, port segments, which are essentially bend points, can also coincide with the destination vertex, effectively creating a free edge segment of zero length. That is, if we have an edge (u, v) that goes through u's port p, we may have a situation where p coincides with v. This is not necessary but allows for smaller grid size in the end.

The embedding is constructed in incremental stages, with each stage corresponding to the insertion of a new vertex v_{k+1}. At each stage, we maintain that each vertex except those on the current external face has exactly three free edge segments. The remaining edge segments connect to a vertex v through port segments. We can divide the current degree of v into three parts: $d_l(v)$, $d_r(v)$, and $d_m(v)$. The degree $d_l(v)$ corresponds to the current number of port edge segments using the L^p region. The degrees $d_r(v)$ and $d_m(v)$ are defined similarly for the R^p and M^p regions. At each insertion, we route port edge segments involving the new vertex v_{k+1} through maximal left and right ports.

DEFINITION 7.7 Let a vertex v have coordinates (v_x, v_y). Then, the *maximal left port of* v, $L^p_{\max}(v)$, has coordinates $(v_x - d_l(v) + 1, v_y + d_l(v))$ if $d_l(v) > 0$ and (v_x, v_y) otherwise. Define the *maximal right port of* v, $R^p_{\max}(v)$, similarly.

7.4.4 The Embedding

Initially, the first three vertices have integer coordinates $v_1 = (0, 0)$, $v_2 = (4, 0)$, and $v_3 = (2, 1)$. In subsequent stages, we insert the next vertex v_{k+1} maintaining the following invariants:

- All vertices and ports lie on the integer grid.
- Let $C_k = (w_1 = v_1, w_2, \ldots, w_m = v_2)$ be the exterior face of G_k with $w_i(x)$ corresponding to the x-coordinate of w_i. Then $w_1(x) < w_2(x) < \ldots < w_m(x)$. In other words, the vertices of the exterior face are strictly x-monotonic.
- Let $e = (w_i, w_{i+1})$ be an edge on the external face. The free edge segment of e has a slope of ± 1. The port edge segment of e passes through a maximal port.
- Every vertex v has at most one free edge segment crossing each free region, and each port segment goes to a unique port.

When we insert a new vertex v_{k+1}, we must create enough space so that the two neighbors w_l and w_r can "see" the new vertex through their maximal right and left ports, which are typically already used. Thus, we must shift these vertices over to create space and also to ensure that the intersection of these ports lies on a grid location, for the new vertex. Of course, we cannot simply shift these vertices, we must shift other vertices to be sure that we do not produce any crossings. Therefore, to shift a vertex w, we shift all vertices in its shifting set, defined in Section 7.2.2, and also most of the ports. See Figure 7.14.

DEFINITION 7.8 For $\delta \geq 0$ and a vertex $w_i \in C_k$, define a *regular-shift by* δ *units of* w_i as shifting all vertices in $M_k(w_i)$ by δ units to the right, including all associated ports. Define the *right-shift by* δ *units on* w_i as a regular-shift of w_i *except* that the ports in the L^p region of w_i are *not* shifted. Similarly, define the *left-shift by* δ *units on* w_i as a regular shift of w_{i+1} *and* additionally shifting the ports in the R^p region of w_i.

Notice that left-shifting a vertex w_i is nearly identical to right-shifting its neighbor w_{i+1} except for the ports that are moved.

Assume that G_k has been embedded and that the invariants hold. We now look at the specific insertion of a new vertex v_{k+1} to create G_{k+1} while maintaining the invariants. For a vertex $w \in C_k$, recall that the current number of port edge segments using R^p is $d_r(w)$ and for L^p is $d_l(w)$. If $d_r(w_l) = 0$, we perform a left-shift of 2 units on w_l; otherwise, we perform a left-shift of 1 unit on w_l. This frees a space for a new maximal port in the

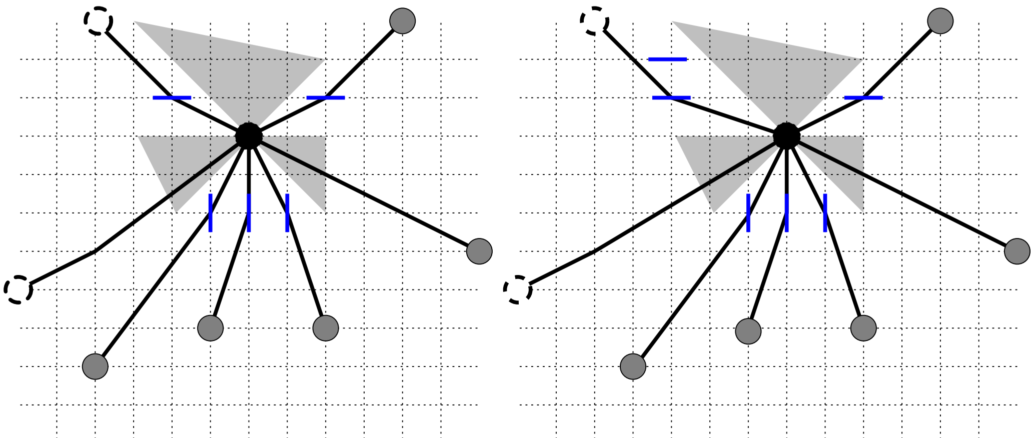


Figure 7.14 (a) A (darkened) vertex and its neighbors before a right shift of one unit. (b) And after a right shift of one unit. The other vertices that are part of the shifting set are highlighted, while those that are not are drawn dashed. Notice that the left port region remains in place creating a location for one more port.

R_p region of w_l. Similarly, if $d_l(w_r) = 0$, we perform a right-shift of 2 units on w_r, and otherwise, we perform a right-shift of 1 unit.

Let l be the line of slope $+1$ passing through w_l's newly created maximal right port. Let r be the line of slope -1 passing through w_r's newly created maximal left port. We place v_{k+1} at the intersection of lines l and r. If l and r intersect at a non-grid location, we simply perform a regular-shift of 1 unit on w_r. Observe that we therefore perform at most 5 shifts per insertion.

We now route the edges as follows. The edge from w_l to v_{k+1} goes from w_l to $R^p_{\max}(w_l)$ and then to v_{k+1} through its free region L^f. The edge from w_r to v_{k+1} goes from w_r to $L^p_{\max}(w_r)$ and then to v_{k+1} through its free region R^f. The remaining edges are from v_{k+1} to w_i for $l < i < r$. These edges are routed from v_{k+1} to nearly consecutive ports on the M^p region of v_{k+1} and then to w_i through its free region M^f. We locate the horizontal line segment containing the ports of M^p exactly $\lceil (r-l)/2 \rceil$ units below v_{k+1}. Duncan and Kobourov [DK03] prove that this guarantees that each port is above each neighbor vertex w_i. In the case that $r - l$ is even, there is exactly one port per edge routed, and the ports are mapped consecutively. In the case of an odd value, we must skip one port in the region, which is easy to identify [DK03]. Figure 7.15 shows the insertion of five vertices of a planar graph using this algorithm.

Duncan and Kobourov prove that this algorithm properly maintains the previous invariants leading to the following theorem:

Theorem 7.5 *[DK03] For a given plane graph $G = (V, E)$, there is a linear-time algorithm that constructs a planar polyline drawing of G with grid size $5n \times 5n/2$ using at most one bend per edge and with an angular resolution no less than $1/2d(v)$ for every vertex $v \in V$.*

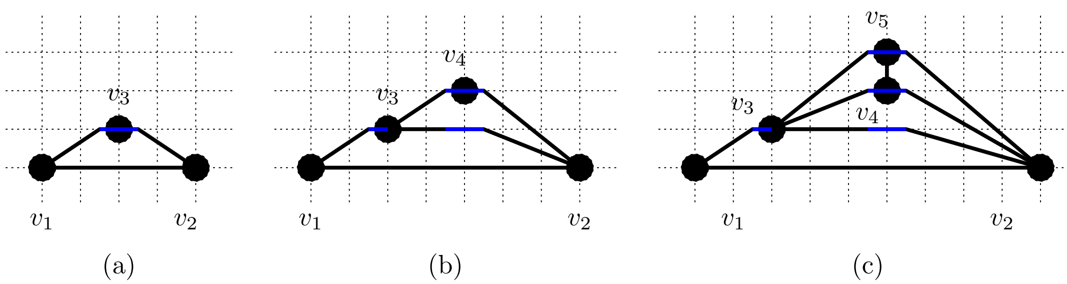


Figure 7.15 The insertion of the first five vertices of a particular planar graph. (a) The initial configuration with 3 vertices. Note that the port edge segment connecting v_1 to v_3 connects to v_1's port which is at the same location as v_3. For clarity, we illustrate the port slightly outside this location. (b) Insertion of v_4. This requires a left-shift of 1 unit for v_3 and a right-shift of 1 unit for v_2 before placing v_4. (c) Insertion of v_5. This requires a left-shift of 1 unit for v_3 and a right-shift of 1 unit for v_5 before placing v_5, which also connects to the *covered* vertex v_4.

7.5 Conclusion

When angular resolution is a desired criterion in drawing a graph, many techniques exist to accommodate it. If the graph is known to be 4-planar or if one is willing to use rectangular regions instead of points for vertices, one can efficiently construct aesthetically pleasing orthogonal drawings [Tam87, TT89, GT97, CK12]. This body of work uses network flows to compute an orthogonal shape with the minimum number of bends and to compact the representation into an orthgonal drawing with minimal height and width.

In addition, several polyline drawing strategies exist that allow one to create good drawings with relatively high angular resolution, a small number of bends, and good area bounds even when the maximum degree of the graph is greater than four [Kan96, GM98, GW00, CDGK01, DK03]. These all extend the incremental insertion algorithm using a canonical ordering initially employed by de Fraysseix, Pach, and Pollack [dFPP90]. The mixed-model approach, employed by Kant [Kan96] and Gutwenger and Mutzel [GM98], uses primarily orthogonal edges but must still connect vertices using some segments whose slopes depend on the degree of the vertex. The works of Cheng et al. [CDGK01] and Duncan and Kobourov [DK03] use an optimal one bend per edge but with one of the two segments of each edge having arbitrary slope. Unlike the purely orthogonal representations, the set of slopes determined by the edges in these polyline drawings is possibly large.

References

[CDGK01] C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. *Discrete and Computational Geometry*, 25(3):405–418, 2001.

[CK12] Sabine Cornelsen and Andreas Karrenbauer. Accelerated bend minimization. *Journal of Graph Algortihms and Applications*, 16(3):635–650, 2012.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[CP95] M. Chrobak and T. Payne. A linear-time algorithm for drawing planar graphs. *Inform. Process. Lett.*, 54:241–246, 1995.

[DETT99] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[dFPP90] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[DHVM83] P. Duchet, Y. Hamidoune, M. Las Vergnas, and H. Meyniel. Representing a planar graph by vertical lines joining different levels. *Discrete Math.*, 46:319–321, 1983.

[DK03] C. A. Duncan and S. G. Kobourov. Polar coordinate drawing of planar graphs with good angular resolution. *Journal of Graph Algorithms and Applications*, 7(4):311–332, 2003.

[DLV98] G. Di Battista, G. Liotta, and F. Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998.

[FHH+93] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Simvonis, Emo Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. *SIAM J. Comput.*, 22:1035–1052, 1993.

[FM98] S. Fialko and P. Mutzel. A new approximation algorithm for the planar augmentation problem. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '98)*, pages 260–269. ACM Press, 1998.

[GM98] C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 167–182. Springer-Verlag, 1998.

[GT97] A. Garg and R. Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In S. C. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 201–216. Springer-Verlag, 1997.

[GT01] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Computing*, 31(2):601–625, 2001.

[GT02] Michael T. Goodrich and Roberto Tamassia. *Algorithm design: foundations, analysis, and Internet examples*. John Wiley and Sons, Inc., New York, NY, 2002.

[GW00] M. T. Goodrich and C. G. Wagner. A framework for drawing planar graphs with curves and polylines. *Journal of Algorithms*, 37(2):399–421, 2000.

[Kan96] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996. Special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia.

[OvW78] R. H. J. M. Otten and J. G. van Wijk. Graph representations in interactive layout design. In *Proc. IEEE Internat. Sympos. on Circuits and Systems*, pages 914–918, 1978.

[RT86] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.*, 1(4):343–353, 1986.

[Tam87] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[TT86] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.*, 1(4):321–341, 1986.

[TT89] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.