
Constraints, An International Journal, 3, 87–120 (1998)

Constraints in Graph Drawing Algorithms

ROBERTO TAMASSIA rt@cs.brown.edu
Department of Computer Science, Brown University, Providence, RI 02912–1910, USA

Abstract. Graphs are widely used for information visualization purposes, since they provide a natural and intuitive representation of complex abstract structures. The automatic generation of drawings of graphs has applications a variety of fields such as software engineering, database systems, and graphical user interfaces. In this paper, we survey algorithmic techniques for graph drawing that support the expression and satisfaction of user-defined constraints.

1. Introduction

Graph drawing addresses the problem of constructing geometric representations of graphs, and is a major aspect of the emerging field of information visualization. Graph drawing has applications to key computer technologies such as software engineering, database systems, information retrieval, visual interfaces, and computer-aided-design. Research on graph drawing has been conducted within several diverse areas, including discrete mathematics (topological graph theory, geometric graph theory, order theory), algorithmics (graph algorithms, data structures, computational geometry, VLSI), and human-computer interaction (visual languages, graphical user interfaces, software visualization).

There are infinitely many drawings for a graph. A drawing *convention* dictates basic rules for mapping vertices and edges to geometric objects. Commonly used drawing conventions include:

Polyline drawing: each edge is drawn as a polygonal chain (Fig. 1.a).

Straight-line drawing: each edge is drawn as a straight line segment (Fig. 1.b).

*Orthogonal drawing***:** each edge is drawn as a polygonal chain of alternating horizontal and vertical segments (Fig. 1.c).

Grid drawing: vertices, crossings, and edge bends have integer coordinates.

Upward (downward) drawing: (for directed acyclic graphs) each edge is drawn monotonically increasing (decreasing) in the vertical direction (Fig. 1.d).

In order to effectively draw a graph, we would like to take into account a variety of properties. For example, planarity and the display of symmetries are highly desirable in visualization applications. Also, it is customary to display trees and acyclic digraphs with upward drawings. In general, to avoid wasting valuable space on a page or a computer screen, it is important to keep the area of the drawing small. Drawing a graph can thus be formalized as a multi-objective optimization problem, where the layout of the graph is

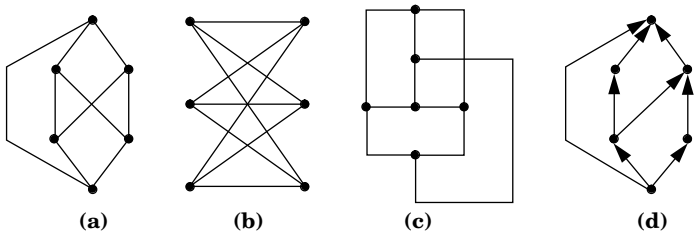

Figure 1. Examples of drawing conventions (a) polyline drawing of $K_{3,3}$; (b) straight-line drawing of $K_{3,3}$; (c) orthogonal drawing of $K_{3,3}$; (d) planar upward drawing of an acyclic digraph.

generated according to a prespecified set of *aesthetics*, i.e., general rules that are embodied in a layout algorithm. Commonly adopted graph drawing aesthetics include:

- Minimization of the total number of crossings between edges. Ideally, we would like to have a planar drawing, but not every graph admits one.
- Minimization of the area of the drawing. This aesthetic is meaningful when the drawing convention adopted prevents drawings from being arbitrarily scaled down (e.g., a grid drawing, or a prescribed minimum edge length).
- Minimization of the total number of bends along the edges. This aesthetic is especially important for orthogonal drawings, while it is trivially satisfied by straight-line drawings.
- Minimization of the aspect ratio of the drawing, which is defined as the ratio of the length of the longest side to the length of the shortest side of the the smallest rectangle with horizontal and vertical sides covering the drawing.
- Maximization of the axial and rotational symmetries displayed by the drawing.
- Maximization of the local (subgraph) isomorphisms and symmetries displayed by the drawing.

Aesthetics are naturally associated with graph optimization problems. However, most such problems are computationally hard. Thus, many heuristic graph drawing techniques have been developed that work well in practice although they do not guarantee to optimize the aesthetics.

Aesthetics are criteria of general applicability that refer to the entire graph and to the entire drawing, In addition to aesthetics, it is important to take into account visualization requirements associated with the specific graph and application domain. For example, we may want to draw a PERT diagram such that the edges of the critical path are horizontally aligned, or we may want to place on the outer boundary of the drawing the vertices of a Data-Flow diagram representing interface modules. It should be noted that in some visualization applications the shape and size of the vertices may be relevant, and overlap between vertices may be allowed.

These specific requirements can be viewed as *constraints* and should be considered as additional input to the drawing problem. Widely used graph drawing constraints include:

- Place a given vertex in the "center" of the drawing.
- Place a given vertex on the outer boundary of the drawing.
- Place a given subset of vertices "close together".
- Draw a given path horizontally aligned from left to right (or vertically aligned from top to bottom).
- Draw a given subgraph with a predefined "shape".

Constraint satisfaction also plays an important role in interactive applications, where a graph undergoes a series of updates (performed either by the user or by the application), and it is important to preserve the *mental map* the user has of the drawing by limiting the changes between the new and old layout.

In this paper, we survey *algorithmic* techniques for drawing graphs that take into account several aesthetics and drawing conventions, and support user-defined constraints. We also mention *declarative* techniques for graph drawing that are based on the specification and resolution of systems of constraints.

Sections 2–3 consider two widely used approaches to drawing general graphs. The force-directed approach, which uses a physical model where the vertices and edges of the graph are viewed as objects subject to various forces, is presented in Section 2. In Section 3, we overview the planarization approach, which takes advantage of the availability of many efficient and well-analyzed drawing algorithms for planar graphs.

More specialized approaches to the construction of planar drawings are covered in Sections 4–6. Section 4 considers visibility representations of planar graphs, where the vertices are represented by horizontal segments and the edges are represented by vertical segments. We show how alignment constraints can be imposed on chains of edge-segments, and give an applications to drawing planar digraphs such that the edges of prescribed paths are constrained to be aligned on parallel lines. In Section 6, we overview a technique for constructing planar orthogonal drawings with the minimum number of bends. This technique naturally supports a variety of constraints on the "orthogonal shape" of the drawing, such as imposing certain edges to have no bends, and specifying the angle formed by two edges incident on a vertex.

Several formalisms have been developed for the specification of constraints, and various graph drawing techniques based on the resolution of systems of constraints have been devised. We overview this work in Section 7.

Finally, a visual approach to graph drawing, where the layout of a graph is pictorially specified "by example," is overviewed in Section 8.

To find out more about graph drawing, see Di Battista et al. (1994), Tamassia (1997) and the WWW graph drawing page maintained by the author at

`http://www.cs.brown.edu/people/rt/gd.html`.

For background material on graph algorithms, see, e.g., Even (1979), Gibbons (1980), Mehlhorn (1984), Nishizeki and Chiba (1988).

2. Force-Directed Approach

Force-directed methods construct straight-line drawings of general graphs using a physical model where the vertices and edges of the graph are viewed as physical objects subject to various forces. Starting from an initial random drawing, the graph evolves into a final drawing, which is a local minimum energy configuration of the physical system. Force-directed methods are referred in the literature also as *spring methods* and *physical simulation methods*.

2.1. Spring Embedder

In Eades' *spring embedder* (Eades, 1984), every pair of vertices is connected by a "spring":

- for adjacent vertices, the spring has unit natural length, thus attracting the vertices unless they are at the "ideal" unit distance;also, the spring has logarithmic strength, i.e., the intensity f_a of the attractive force exerted on its endpoints depends on the length ℓ according to the following formula:

  $$f_a = c_a \log \ell$$

  where c_a is a parameter;

- for nonadjacent vertices, the spring has infinite natural length, thus always repelling the vertices; also, the spring has inverse-square strength, i.e., the intensity f_r of the repulsive force exerted on its endpoints depends on the length ℓ according to the following formula:

  $$f_r = \frac{c_r}{\ell^2}$$

  where c_r is a parameter.

Rather than solving with symbolic methods a system of differential equations, the evolution of the system is usually simulated using numerical methods. The simulation consists of a sequence of steps (see Fig. 2). At each step, for each vertex v, the resulting force $\vec{f}$ on v is computed, and v is translated by $\Delta \vec{f}$, where Δ is a displacement parameter.

2.2. Other Force-Directed Methods

The force-directed approach was pioneered in Eades (1984), Kruskal and Seery (1980). Notable developments include Davidson and Harel (1996), Frick, Ludwig, and Mehldau, 1995), Fruchterman and Reingold (1991), Harel and Sardas (1995), Kamada and Kawai (1989), Sugiyama and Misue (1995). In particular:

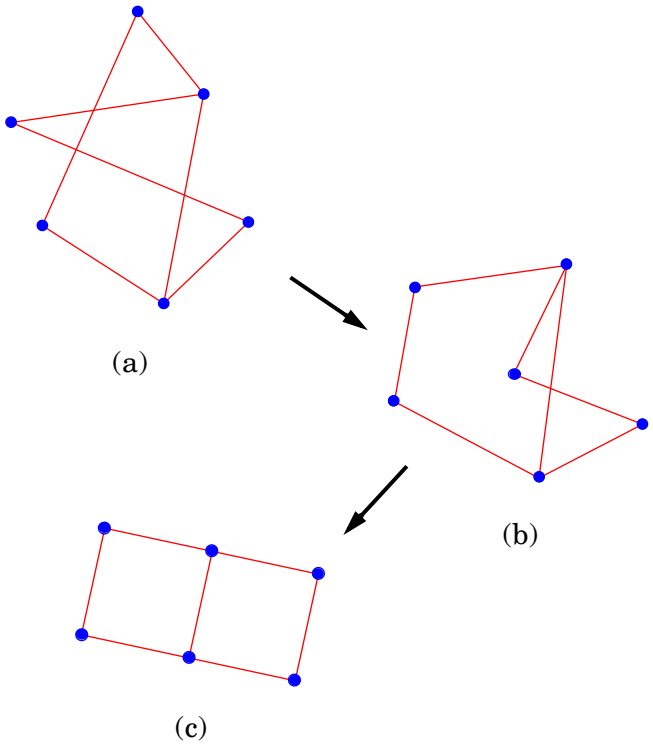

Figure 2. Simulation of the evolution of a drawing in the spring embedder algorithm: (a) initial drawing; (b) drawing at an intermediate step; (c) final drawing.

- Kamada and Kawai (1989) use forces that attempt at placing vertices such that, for a pair of vertices u and v, their Euclidean distance in the drawing is equal to their graphtheoretic distance $d(u, v)$ in the graph (the length of a path with the fewest edges connecting them); thus, in their approach, edge (u, v) has ideal length $d(u, v)$.
- Fruchterman and Reingold (1991) use a complex system of forces similar to that of subatomic particles and celestial bodies; also, they control the size of the drawing by assuming that the boundary of the prespecified drawing region acts as a "wall".
- Davidson and Harel (1996) do not use forces explicitly, but define an energy function takes into account vertex distribution, edge-lengths, and edge-crossings; the simulated annealing method is used to find a drawing with low energy.

Brandenburg, Himsolt and Roher (1996) have conducted an experimental comparison of various force-directed methods. Their results indicate that drawing algorithms based on the force-directed approach have several advantages. Namely, they are:

- relatively simple to implement;

- straightforward to parameterize;
- easy to extend by adding new forces;
- usually effective for small graphs with regular structure;
- often able to detect and display symmetries in the graph (see, e.g., Fig. 2);
- capable of preserving the user's "mental map" by providing a continuous evolution of the drawing from the initial to the final configuration.

Disadvantages include:

- the running time for large graphs is rather slow;
- only straight-line drawings are supported;
- the properties of the generated drawings (e.g., area, crossings) are difficult to analyze theoretically.

2.3. Constraints in the Force-Directed Approach

The force-directed approach has been extended to support several types of constraints in Dengler, Friedell, and Marks (1993), He and Marriott (1997), Kamps, Kleinz, and Read (1996), Ryall, Marks, and Shieber (1997), Sugiyama and Misue (1995). Simple extensions are mentioned below. For more sophisticated extensions, see Section 7.

Three types of constraints can be easily supported within the force-directed approach:

- position constraints;
- fixed-subgraph constraints; and
- constraints that can be expressed by forces or energy functions.

A position constraint assigns to a vertex a topologically connected region where the vertex should remain. Examples of prescribed regions include:

1. a single point, equivalent to "pinning down" the vertex at a specific location;
2. a line, equivalent to assigning a "layer" to the vertex;
3. a circle, which allows to place groups of vertices into distinct regions.

Position constraints are immediate to satisfy by confining to the prescribed region the translation of the vertex at each simulation step.

A fixed-subgraph constraint assigns to a subgraph a prescribed subdrawing, which may appear translated or rotated, but not otherwise deformed, in the overall drawing of the graph. It can be supported by considering the subgraph as a rigid body, which gets translated and rotated at each simulation step according to the overall force and torque applied to it as a result of the individual forces applied to its vertices.

Constraints that can be expressed by forces include:

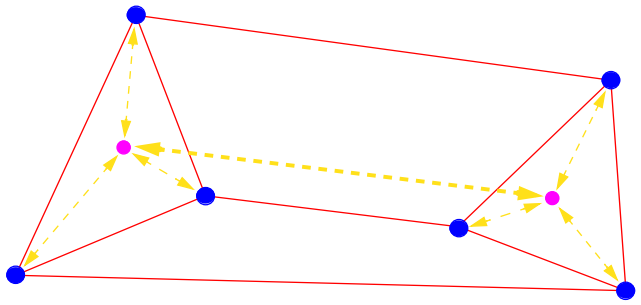

Figure 3. Example of clustering constraints realized by means of forces: the two shaded vertices represent the dummy attractors of the two clusters; the attractive forces between cluster vertices and attractors, and the repulsive force between attractors are shown with dashed double-arrows.

- clustering of vertices;
- alignment of vertices;
- orientation of directed edges.

Clustering can be achieved as follows (see Fig. 3):

1. for each cluster, add to the graph a dummy "attractor" vertex;
2. add attractive forces between an attractor and each vertex in the attractor's cluster;
3. add repulsive forces between pairs of attractors and between attractors and vertices not in any cluster.

Forces that simulate a parallel "magnetic field" can be used to orient the edges according to a general direction, e.g., upward. Namely, a torque is applied to the edges that tends to align them with the field lines. This approach was recently presented in Sugiyama and Misue (1995).

2.4. Barycentric Method

We now show that a classical drawing algorithm for planar graphs, the *barycentric method* by Tutte (1960, 1963), can be reinterpreted as a constrained force-directed method.

Let G be a triconnected graph, and let γ be a nonseparating cycle of G, i.e., a cycle whose removal does not disconnect G. The barycentric method draws first γ as a convex polygon Γ. Then, for each vertex v not in γ, it writes the following *barycentric equations*, which express the fact that vertex v is in the barycenter (centroid) of its neighbors:

$$x(v) = \frac{1}{\deg(v)} \sum_{w \text{ adj. to } v} x(w) \tag{1}$$

$$y(v) = \frac{1}{\deg(v)} \sum_{w \text{ adj. to } v} y(w) \tag{2}$$

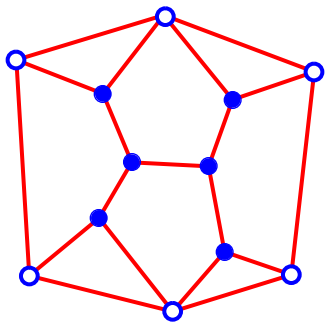

Figure 4. Example of drawing of a planar triconnected graph constructed by the barycentric method.

The barycentric equations form a linear system whose known values are the x- and y-coordinates of the vertices of cycle γ. Tutte proved the following deep result (Tutte, 1960; Tutte 1963) (see Fig. 4):

THEOREM 1 *For a planar triconnected graph G, the barycentric equations admit a unique solution that yields a planar straight-line drawing of G such that the external face is drawn as polygon Γ, and all the internal faces are drawn as convex polygons.*

Consider now the following constrained variation of the spring embedder algorithm:

- an edge is viewed as a spring of zero natural length that exerts an attractive force on its endpoints proportional to the length of the edge (thus obeying Hooke's law);
- there are no repulsive forces between nonadjacent vertices;
- position constraints pin down each vertex of γ to the corresponding vertex of convex polygon Γ.

The total energy $\mathcal{E}$ of the above system is given by:

$$\mathcal{E} = \frac{1}{2} \sum_{(v,w)\in G} (\ell(v, w))^2,$$

where $\ell(v, w)$ denotes the length of edge (v, w). Thus

$$\mathcal{E} = \frac{1}{2} \sum_{(v,w)\in G} \left(((x(v) - x(w))^2 - ((y(v) - y(w))^2\right),$$

At equilibrium, the system will be at a configuration of local minimum for energy $\mathcal{E}$. This implies that at equilibrium,

$$\nabla\mathcal{E} = 0,$$

i.e., the partial derivatives of $\mathcal{E}$ with respect to each variable $x(v)$ and $y(v)$ must be equal to zero:

$$\frac{\partial \mathcal{E}}{\partial x(v)} = \sum_{w \text{ adj. to } v} (x(v) - x(w)) = 0 \tag{3}$$

$$\frac{\partial \mathcal{E}}{\partial y(v)} = \sum_{w \text{ adj. to } v} (y(v) - y(w)) = 0 \tag{4}$$

By simple manipulations, it is easy to see that the above equations 3 and 4 are equivalent to the barycentric equations 1 and 2, respectively.

3. Planarization Approach

The planarization approach is motivated by the availability of many efficient and well-analyzed drawing algorithms for planar graphs. If the graph is nonplanar, it is transformed into a planar graph by means of a preliminary planarization step that replaces each crossing with a fictitious vertex, and then a drawing method for planar graphs is applied (see Fig. 5).

A successful drawing algorithm based on the planarization approach and a bend-minimization method (Tamassia, 1987) (see Section 6) is described in Tamassia, Di Battista, and Batini (1988). Systems based on this algorithm have been been widely used in information visualization applications.

3.1. Planarization Methods

Finding the minimum number of crossings or a maximum planar subgraph are NP-hard problems (Garey and Johnson, 1983). Hence, existing planarization algorithms use heuristics.

A simple planarization method that uses as a subroutine an algorithm for finding a planar subgraph works as follows (see Fig. 6):

1. compute a planar subgraph of the input graph, and partition the edges into "planar" and "nonplanar" accordingly.
2. construct a planar embedding of the planar subgraph;
3. add the nonplanar edges, one at at a time, minimizing each time the number of crossings introduced.

The computation in Step 3 of the above algorithm for a certain nonplanar edge (u, v) is equivalent to finding a shortest path in the dual graph of the current embedding from the faces incident to u to the faces incident to v.

A simple heuristic for Step 1, computes a "maximal" planar subgraph S of the input graph G as follows:

1. start with subgraph S consisting only of the vertices of G, but no edges;

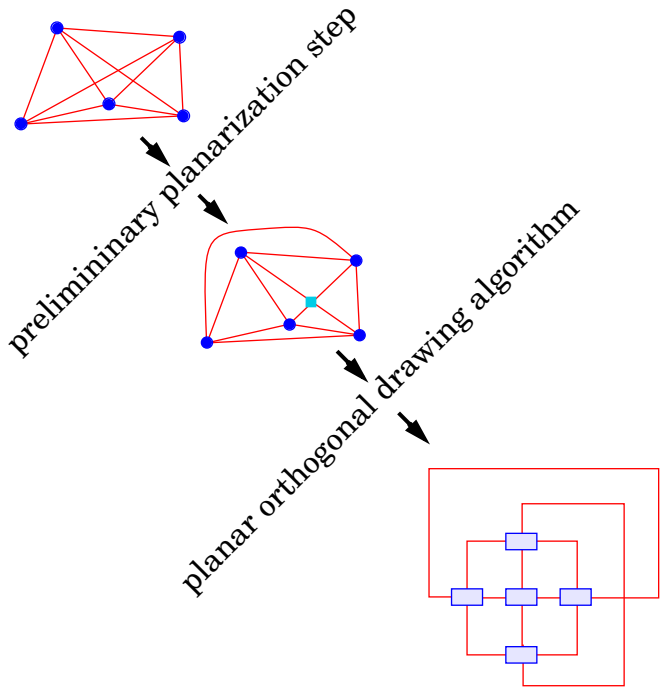

Figure 5. Schematic illustration of the planarization approach.

2. for each edge e of G, if the graph obtained by adding e to S is planar, then add e to S and classify e as "planar", else reject e and classify it as "nonplanar".

The best available heuristic for the maximum planar subgraph problem is described in Jünger and Mutzel (1996). This method has a solid theoretical foundation in polyhedral combinatorics, and achieves good results in practice.

3.2. Constraints in the Planarization Approach

Several constraints of topological nature can be supported within the planarization approach, including:

- preventing crossings on edges, and
- placing vertices on the external boundary.

In order to prevent the edges of a given subset E^* from having crossings, we can use the following variation of the simple planarization heuristic of Section 3.1:

- In Step 1, we try adding first the edges of E^* so that a maximal subset of them will be in the planar subgraph computed by this step.

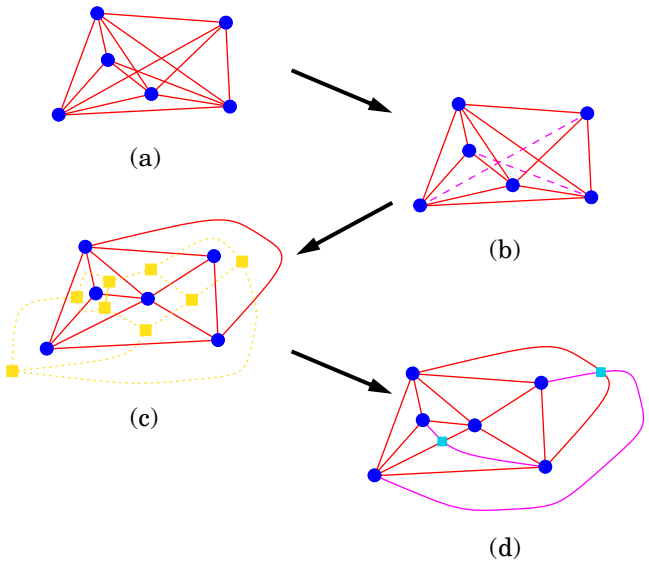

Figure 6. Simple planarization method: (a) initial graph; (b) partition of the edges into planar (solid) and nonplanar (dashed); (c) dual graph (dotted) used to route the nonplanar edges; (d) final planarized graph.

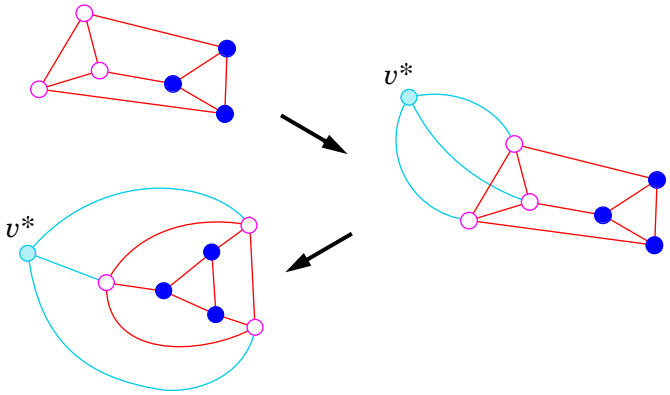

Figure 7. Constraining vertices to be on the external face.

- In Step 3, we place a large "crossing cost" on the edges of E^*.

In order to constrain a given subset of vertices V^* to be on the external boundary, we can add to the graph a fictitious vertex v^* and fictitious edges connecting v^* with all the vertices in V^*. We then impose the aforementioned constraint of preventing crossings on the edges incident on v^*. See Fig. 7.

4. Visibility Representations and Planar Polyline Drawings

We start by recalling some definitions on numberings and orientations of digraphs, which are used in the drawing techniques discussed in this section.

4.1. Properties of Planar Directed Graphs

Let G be a directed graph with n vertices. A *topological numbering* of G is an assignment of numbers to the vertices of G such that, for every edge (u, v) of G, the number assigned to v is greater than the one assigned to u (i.e., $number(v) > number(u)$). A *topological sorting* is a topological numbering of G such that every vertex is assigned a distinct integer between 1 and n. A topological sorting is not unique unless G has a directed path that visits every vertex. It is easy to show that the following statements are equivalent:

- G is acyclic;
- G admits a topological numbering;
- G admits a topological sorting.

If the edges of G have nonnegative weights associated with them, a *weighted topological numbering* is a topological numbering of G such that, for every edge (u, v) of G, the number assigned to v is greater than or equal to the number assigned to u plus the weight of (u, v) (i.e., $number(v) \geq number(u) + weight(u, v)$. The numbering is *optimal* if the range of numbers assigned to the vertices is minimized (i.e., $\max_v number(v) - \min_u number(u)$ is minimum). An example of an optimal weighted topological numbering is shown in Fig. 8.

There are simple linear-time algorithms for computing a (weighted) topological numbering or sorting. For example, an optimal weighted topological numbering can be obtained by assigning to each vertex a number equal to the length of a longest directed path terminating at that vertex. Note that all source vertices of G are assigned number 0.

An acyclic digraph with a single source s and a single sink t is called an *st-graph*. A *planar st-graph* is an st-graph that is planar and embedded with vertices s and t on the boundary of the external face. It is easy to visualize a planar st-graph as drawn upward in the plane (with s at the bottom and t at the top), as shown in Fig. 8.a. Since a planar st-graph is acyclic, it admits a topological ordering (numbering).

Let G be a planar st-graph and F be its set of faces (recall that G is embedded). We conventionally assume that F contains two representatives for the external face: the "left external face" s^*, which is incident with the edges on the left boundary of G, and the "right external face" t^*, which is incident with the edges on the right boundary of G. For each edge e, we define $orig(e)$ and $dest(e)$ as the tail and head vertices of e, respectively. Also, we define $left(e)$ (resp. $right(e)$) to be the face to the left (resp. right) of e.

Let G^* be the digraph with vertex set F and edge set $\{(f, g) \mid f = left(e)$ and $g = right(e)$, for every edge $e \neq (s, t)$ of $G\}$. Note that G^* is the dual graph of G, except that the external face of G^* is duplicated such that the left (resp. right) external face inherits the outgoing (resp. incoming) edges. Its edges are oriented from left to right. It is easy to see that digraph G^* is a planar st-graph (see Fig. 9).

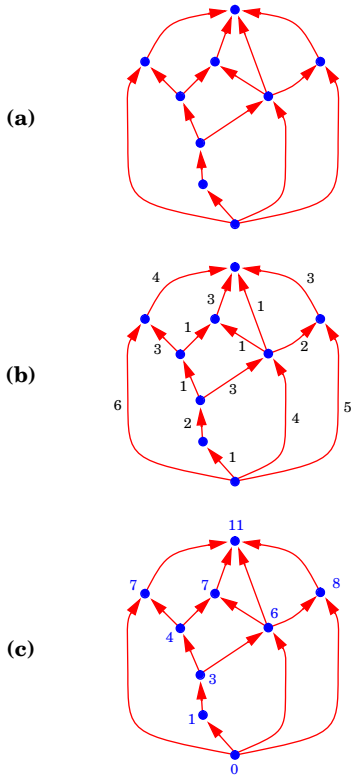

Figure 8. (a) Example of a planar st-graph G; (b) weights on the edges of G and (c) optimal weighted topological numbering of G (the path through vertices numbered 0, 1, 3, 6, 8, 11 proves the optimality of the numbering).

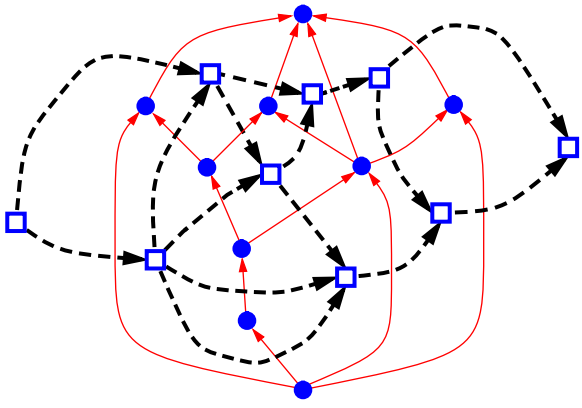

Figure 9. Digraph G^* associated with the planar st-graph G of Fig. 8.a (represented with dashed lines).

4.2. Visibility Representations

Let G be a planar st-graph. A *visibility representation* Γ of G draws each vertex v as a horizontal segment, called *vertex-segment* $\Gamma(v)$, and each edge (u, v) as a vertical segment, called *edge-segment* $\Gamma(u, v)$, such that:

- the vertex-segments do not overlap;
- the edge-segments do not overlap;
- edge-segment $\Gamma(u, v)$ has its bottom endpoint on $\Gamma(u)$, its top endpoint on $\Gamma(v)$, and does not intersect any other vertex-segment.

The study of visibility representations was originally motivated by VLSI layout and compaction problems. Algorithms that construct visibility representations in linear time are given in Di Battista and Tamassia (1988), Rosenstiehl and Tarjan (1986), Tamassia and Tollis (1986).

Algorithm 4.2 (*Visibility*) (Di Battista and Tamassia, 1988; Rosenstiehl and Tarjan, 1986; Tamassia and Tollis, 1986) provides a direct construction of a visibility representation for a planar st-graph G. For the sake of simplicity, the same notation is used for a vertex-segment of the visibility representation and its corresponding vertex in the graph. The same is done for an edge-segment and its corresponding edge.

An example of the construction obtained by Algorithm *Visibility* is shown in Fig. 10.

THEOREM 2 *Let G be a planar st-graph with n vertices. Algorithm* Visibility *computes a visibility representation of G with integer coordinates and $O(n^2)$ area in $O(n)$ time.*

4.3. Constrained Visibility Representations

In this section we show how to construct a visibility representation such that the edges of given paths are constrained to be vertically aligned (Di Battista, Tamassia, and Tollis, 1992). Such *constrained visibility representation* is interesting in itself and can be used as a starting point for obtaining constrained planar polyline drawings.

Let G be a planar st-graph with n vertices. Two paths π_1 and π_2 of G are said to be *nonintersecting* if they are edge disjoint and do not *cross* at common vertices, i.e., there is no vertex v of G with edges e_1, e_2, e_3 and e_4 incident in this clockwise order around v, such that e_1 and e_3 are in π_1 and e_2 and e_4 are in π_2. Observe that any two vertex disjoint paths are also nonintersecting.

Given a collection Π of nonintersecting paths of G, we consider the problem of constructing a visibility representation Γ of G such that for every path π of Π, the edges of π are vertically aligned. More formally, for any two edges e' and e'' of π the edge-segments $\Gamma(e')$ and $\Gamma(e'')$ have the same x-coordinate. Algorithm 4.3 (*Constrained-Visibility*) takes as input G and Π, and constructs a constrained visibility representation Γ of G. In order to simplify the description of the algorithm, without loss of generality, we assume that the set Π of nonintersecting paths covers the edges of G. Otherwise, each edge originally not in Π, is inserted in Π as an independent path.

Algorithm *Visibility*
Input: planar st-graph G with n vertices
Output: visibility representation Γ of G with integer coordinates and area $O(n^2)$

1. Assign unit weights to the edges of G and compute an optimal weighted topological numbering Y of G.

2. Assign unit weights to the edges of G^* and compute an optimal weighted topological numbering X of G^*.

3. For each vertex v, draw the vertex-segment $\Gamma(v)$ at y-coordinate $Y(v)$ and between x-coordinates $X(left(v))$ and $X(right(v) - 1)$. In other words,

   for each vertex v **do**
   draw $\Gamma(v)$ as the horizontal segment with
   $y(\Gamma(v)) = Y(v)$;
   $x_L(\Gamma(v)) = Y(left(v))$;
   $x_R(\Gamma(v)) = Y(right(v)) - 1$;
   endfor

4. For each edge e, draw the edge-segment $\Gamma(e)$ at x-coordinate $X(left(e))$ between y-coordinates $Y(orig(e))$ and $Y(dest(e))$. In other words,

   for each edge e **do**
   draw $\Gamma(e)$ as the vertical segment with
   $x(\Gamma(e)) = X(left(e))$;
   $y_B(\Gamma(e)) = Y(orig(e))$;
   $y_T(\Gamma(e)) = Y(dest(e))$;
   endfor

□

To help the intuition of the reader, we observe that the computations performed by the algorithm are equivalent to the following construction. First, it modifies G by duplicating each path π in Π thus forming a new face for each path. This is equivalent to having a vertex of G_Π (defined in Step 1) correspond to each path in Π. Second, it constructs a visibility representation for the modified graph such that the edge-segments of the left side of the boundary of each face are vertically aligned and two copies of an original vertex are horizontally aligned. Finally, it removes the right copy of every duplicated edge and joins the copies of the duplicated vertices. Fig. 11 shows a running example of Algorithm *Constraint Visibility*.

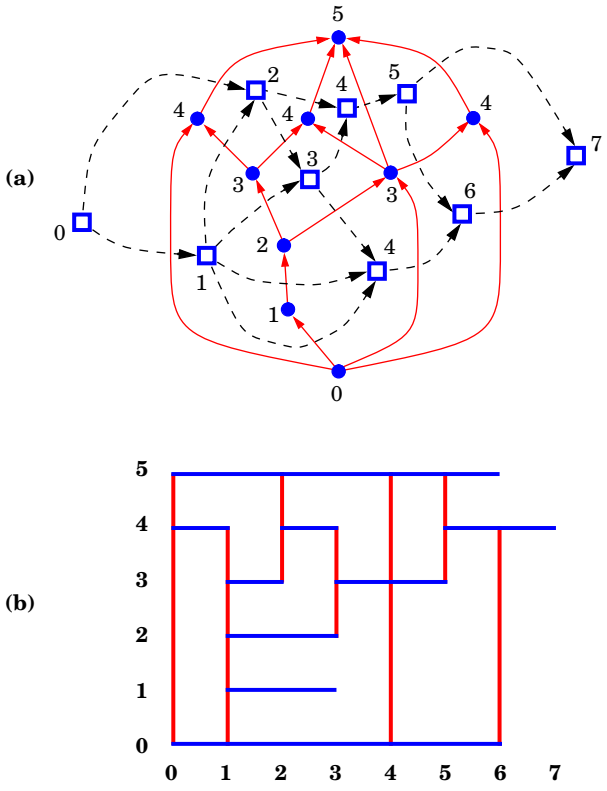

Figure 10. Example of a visibility representation constructed by Algorithm *Visibility*: (a) a planar st-graph G (drawn with solid lines), its dual G^* (drawn with dashed lines), and the numberings of G and G^* (shown as labels); (b) a visibility representation for G, where the vertices are represeneted by horizontal segments and the edges are represented by vertical segments.

THEOREM 3 *Let G be a planar st-graph with n vertices, and Π be a set of nonintersecting paths covering the edges of G. Algorithm* Constrained Visibility *computes in $O(n)$ time a visibility representation of G with integer coordinates and $O(n^2)$ area, such that the edges of every path π in Π are vertically aligned.*

5. Polyline Drawings from Visibility Representations

We can construct a planar upward polyline drawing of a planar st-graph G starting from a visibility representation of G as follows (Di Battista and Tamassia, 1988): we draw each vertex of G at an arbitrary point of its vertex-segment, and each edge (u, v) of G as a three-segment polygonal chain whose middle segment is a subset of the edge-segment of (u, v). E.g., a possible choice for the placement of $P(v)$ is the middle point of vertex-segment

Algorithm *Constrained-Visibility*
Input: planar st-graph G with n vertices; set Π of nonintersecting paths covering the edges of G
Output: constrained visibility representation Γ of G with integer coordinates and area $O(n^2)$

1. Construct the graph G_Π with vertex set $F \bigcup \Pi$ (recall that F is the set of faces of G) and edge set $\{(f, \pi) \mid f = \mathit{left}(e)$ for some edge e of path $\pi\} \bigcup \{(\pi, g) \mid g = \mathit{right}(e)$ for some edge e of path $\pi\}$.

   Note that graph G_Π is a planar st-graph.

2. Assign unit weights to the the edges of G and compute an optimal weighted topological numbering Y of G such that $Y(s) = 0$.

3. Assign half-unit weights to the edges of G_Π and compute an optimal weighted topological numbering X of G_Π such that $X(s^*) = -1/2$.

4. **for each** path π in Π **do**
   for each edge e in π **do**
   draw $\Gamma(e)$ as the vertical segment with
   $x(\Gamma(e)) = X(\pi)$;
   $y_B(\Gamma(e)) = Y(\mathit{orig}(e))$;
   $y_T(\Gamma(e)) = Y(\mathit{dest}(e))$;
   endfor endfor

5. **for each** vertex v **do**
   draw $\Gamma(v)$ as the horizontal segment with
   $y(\Gamma(v)) = Y(v)$;
   $x_L(\Gamma(v)) = \min_{v \in \pi} X(\pi)$;
   $x_R(\Gamma(v)) = \max_{v \in \pi} X(\pi)$;
   endfor

□

$\Gamma(v)$. Examples of polyline drawings obtained from the visibility representation of Fig. 10 are shown in Fig. 12.

The above technique can be extended to constrained visibility representations. Let Γ be a constrained visibility representation for a planar st-graph G and a set of *vertex disjoint* paths Π. Algorithm (*Constrained-Polyline*) (Di Battista, Tamassia, and Tollis, 1992) derives from Γ a planar upward polyline drawing of G such that all the internal vertices in a path of Π are vertically aligned.

Notice that if there is an edge (u, v) such that $y(v) - y(u) = 2$, then the two middle points

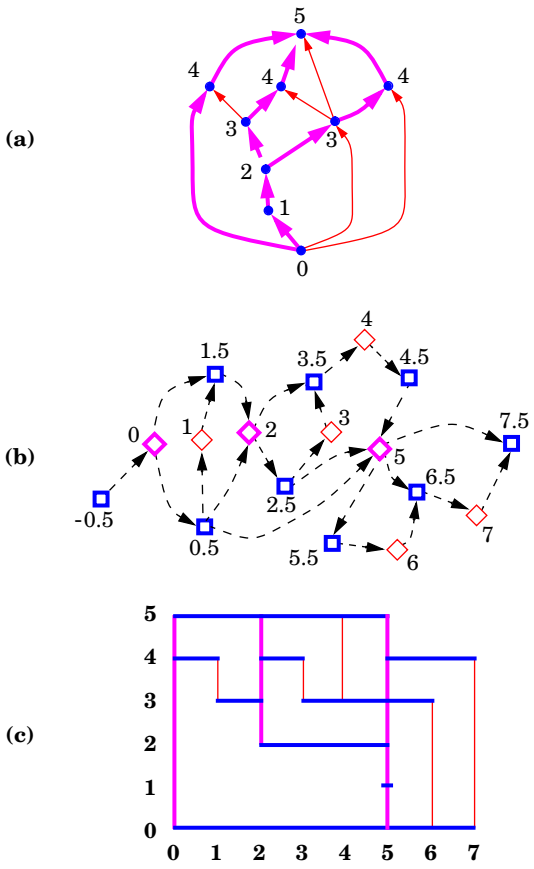

Figure 11. Example of constrained visibility representation computed by Algorithm *Constrained-Visibility*: (a) planar st-graph G, topological numbering of G, and set Π of paths that cover the edges of G, where the paths with at least two edges are drawn with thick lines; (b) graph G_Π and its topological numbering, where the square vertices represent faces of G and the diamond vertices represent paths of Π; (c) constrained visibility representation of G.

of the polygonal chain associated with (u, v) are coincident. Fig. 13 shows the polyline drawing obtained from the constrained visibility representation shown in Fig. 11.

THEOREM 4 *Let G be a planar st-graph with n vertices, and Π be a set of vertex disjoint paths of G. A planar upward polyline drawing Γ for G with the following properties can be computed in $O(n)$ time:*

- *vertices and bends have integers coordinates;*
- *Γ has $O(n^2)$ area;*
- *for every path π in Π, the edges of π are vertically aligned; and*
- *Γ has at most $4n - 10$ bends.*

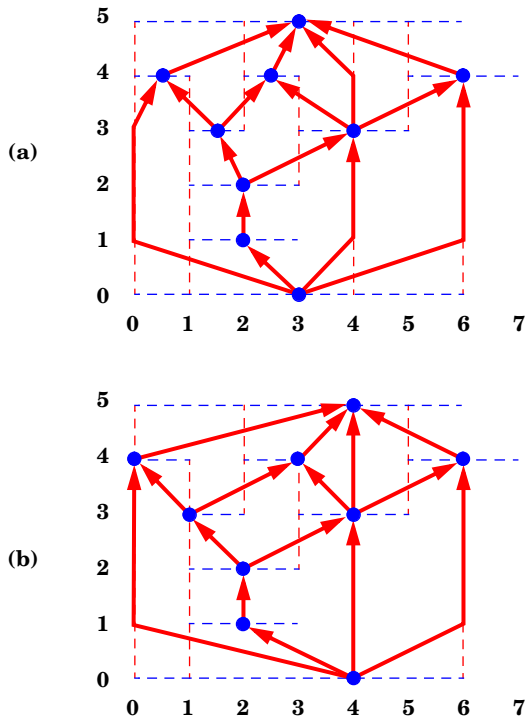

Figure 12. Polyline drawings constructed by Algorithm *Polyline* from the visibility representation of Fig. 10: (a) median positioning; (b) "long-edge" positioning with integer coordinates.

Theorem 4 allows one to effectively visualize specific paths, e.g. *critical paths* in timed PERT diagrams. A *PERT diagram* is a directed acyclic graph whose edges are associated with the tasks of a given project and whose vertices are associated with designated events in the evolution of the project, i.e., the start and completion of the various tasks. A PERT diagram has a unique source vertex s, denoting the start of the project, and a unique sink vertex t, denoting the termination of the project. Each edge has a weight which represents the expected duration of the task. The tasks are partially ordered due to technical constraints. Hence, all tasks associated with edges outgoing from a vertex v can start only if all the tasks associated with the edges incoming in v are completed. The minimum time to complete the entire project is the length of a longest path from s to t, and such a path is called a critical path, since delaying any task along that path causes a corresponding delay of the entire project. Therefore, identifying critical paths is very important in planning and monitoring the execution of the project.

Assume that the project starts at time 0. A timed PERT diagram is drawn in such a way that the y direction denotes the flow of time. For example, a vertex v is drawn at a y-coordinate equal to the earliest starting time of the corresponding event, i.e., $y(v)$ is the

Algorithm *Constrained-Polyline*
Input: A constrained visibility representation Γ of a planar st-graph G with respect to a set of vertex disjoint paths Π constructed by Algorithm *Constrained-Visibility*
Output: a planar upward polyline drawing of G such that, for every path π of Π, all the internal vertices of π are vertically aligned

1. **for each** vertex v **do**
   replace the vertex-segment $\Gamma(v)$ with a point $P(v) = (x(v), y(v))$ on $\Gamma(v)$ as follows:
   if v belongs to a path π of Π **then**
   $x(v) = X(\pi)$; $y(v) = Y(v)$;
   else
   choose any point on $\Gamma(v)$
   endfor

2. **for each** edge (u, v) **do**
   if $y(v) - y(u) = 1$ **then** { *short* edge }
   replace the edge-segment $\Gamma(u, v)$ with the segment with endpoint $P(u)$ and $P(v)$
   else { *long* edge }
   replace the edge-segment $\Gamma(u, v)$ with the polygonal line from $P(u)$ to $P(v)$ through $(x(\Gamma(u, v)), y(u) + 1)$ and $(x(\Gamma(u, v)), y(v) - 1)$
   endfor

□

length of a longest path from s to v. Our technique allows to draw a timed PERT diagram in such a way that critical paths are effectively displayed along straight lines.

6. Flow and Planar Orthogonal Drawings

An important aesthetic for planar orthogonal drawings is the minimization of the number of bends (see Fig. 14). In this section, we present a graph drawing method based on network flow techniques that constructs a planar orthogonal drawing of an embedded planar graph with minimum number of bends (Tamassia, 1987). We show that minimizing bends in planar orthognal drawings can be modeled as a minimum cost flow problem on a flow network derived from the graph and its embedding. In this flow network, each unit of flow corresponds to a $\pi/2$ angle, the vertices are producers of four units of flow, the faces

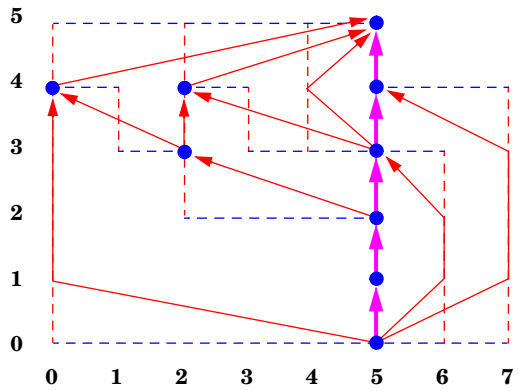

Figure 13. Polyline drawing obtained from the constrained visibility representation of Fig. 11. Note that we have aligned only one path.

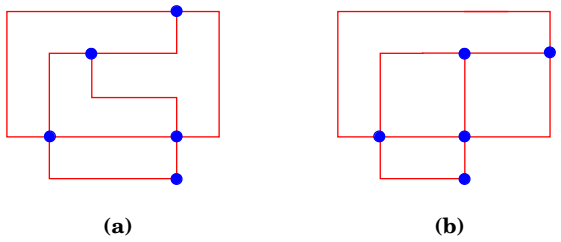

Figure 14. Two planar orthogonal drawings of the same embedded planar graph: (a) drawing with 9 bends; (b) drawing with 6 bends. The drawing in part (b) has the minimum number of bends.

consume an amount of flow proportional to the number of angles in their interior, and each bend transfers a unit of flow across its incident faces. By giving unit cost to the flow associated with bends, we have that a drawing with the minimum number of bends corresponds to a flow of minimum cost. This yields a quadratic-time algorithm for bend minimization. This technique was first presented in Tamassia (1987), with variations and refinements given in Tamassia (1985), Tamassia, Di Battista, and Batini (1988).

Recall that only graphs with vertices of degree at most four admit a planar orthogonal drawing. Hence, throughout this section, unless otherwise specified, we consider only graphs with vertices of degree at most four.

In order to describe the algorithm, we introduce the concept of *orthogonal representation*, which defines the "shape" of an orthogonal drawing in terms of angles, without considering the actual lengths of the edges.

6.1. Orthogonal Representation

In this section, we introduce the concept of orthogonal representation, which captures the notion of "orthogonal shape" of a planar orthogonal drawing by taking into account angles but disregarding edge lengths.

Let Γ be a planar orthogonal drawing of an embedded planar graph G. There are two types of angles in Γ:

- angles formed by two edges incident on a common vertex, called *vertex-angles*; and
- angles formed by bends, called *bend-angles*.

Let G be an embedded planar graph with vertices of degree at most four. We denote with $a(f)$ the total number of vertex-angles inside face f of G. If G is biconnected, $a(f)$ is equal to the number of vertices (edges) of f. For each (undirected) edge e of G with endpoints u and v, we call *darts* the two possible orientations (u, v) and (v, u) of edge e. A dart is said to be *counterclockwise* with respect to face f if f is on the left hand side when traversing the dart from tail to head. We denote with $D(v)$ the set of darts with vertex v as tail, and with $D(f)$ the set of counterclockwise darts on face f.

An *orthogonal representation* of G is an assignment of integer values $\alpha(u, v)$ and $\beta(u, v)$ to each dart (u, v) of G such that:

- $1 \leq \alpha(u, v) \leq 4$;
- $\beta(u, v) \geq 0$;
- for each vertex u, the sum of $\alpha(u, v)$ over all the darts with tail v is equal to four, i.e.,

$$\sum_{(u,v)\in D(u)} \alpha(u, v) = 4;$$

- for each internal face f, the sum of $\alpha(u, v) + \beta(v, u) - \beta(u, v)$ over all the counterclockwise darts (u, v) on face f is equal to $2a(f) - 4$, i.e.,

$$\sum_{(u,v)\in D(f)} \alpha(u, v) + \beta(v, u) - \beta(u, v) = 2a(f) - 4;$$

- for the external face h, the above sum is equal to $2a(h) + 4$, i.e.,

$$\sum_{(u,v)\in D(h)} \alpha(u, v) + \beta(v, u) - \beta(u, v) = 2a(h) + 4.$$

An orthogonal representation describes an equivalence class of orthogonal drawings with "similar shape". More formally, an orthogonal drawing Γ of G yields the following assignment of values α and β to the darts of G (see Fig 15.a):

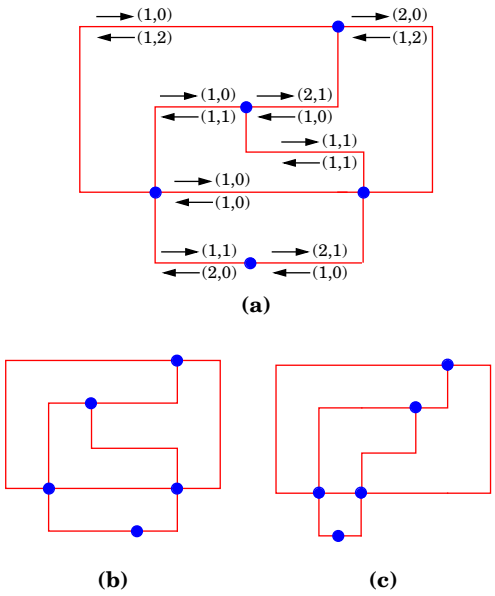

Figure 15. Three planar orthogonal drawings with the same orthogonal representation. Each dart (u, v) of the drawing in part (a) is labeled with the pair $[\alpha(u, v), \beta(u, v)]$.

- $\alpha(u, v) \cdot \pi/2$ is the angle at vertex u formed by the first segments of dart (u, v) and the next dart counterclockwise around u (see Fig. 15.a);
- $\beta(u, v)$ is the number of bends along dart (u, v) with the $\pi/2$ angle on the left hand side (see Fig. 15.a).

The above assignment is an orthogonal representation, which is said to be the orthogonal representation of drawing Γ. We show in Fig. 15 three planar orthogonal drawings of a graph with the same orthogonal representation. Note that the orthogonal representation describes the "shape" of an orthogonal drawing, up to permuting the order of the bends along each edge. In particular, two orthogonal drawings with the same orthogonal representation have the same number of bends (equal to the sum of the β values over all the darts). Conversely, given an orthogonal representation H, there exists a planar orthogonal drawing with orthogonal representation H that can be constructed in linear time.

6.2. The Network Flow Model

In this section, we present a network flow model for the problem of constructing planar orthogonal drawings of embedded planar graphs. This model views angles as a "commodity" that is "produced" by the vertices, "transported" between faces by the edges through their bends, and eventually "consumed" by the faces. Hence, the nodes of the network are

the vertices and faces of the graph. Since all angles we deal with have measure $k\pi/2$, with $1 \leq k \leq 4$, we establish the convention that a unit of flow represents a $\pi/2$ angle.

We associate with an embedded planar graph G a flow network $\mathcal{N}$ whose nodes have supplies and demands, and whose arcs have each a lower bound λ, a capacity μ, and a cost χ, as follows (see Fig. 16):

- the nodes of $\mathcal{N}$ are the vertices and faces of G;
- a vertex-node v of $\mathcal{N}$ produces flow $\sigma(v) = 4$;
- a face-node f of $\mathcal{N}$ consumes flow $\sigma(f) = 2a(f) - 4$ if f is an internal face, and flow $\sigma(h) = 2a(h) + 4$ if $f = h$ is the external face;
- for each dart (u, v) of G, with faces f and g on its left and right, respectively, $\mathcal{N}$ has two arcs (u, f) and (f, g), where:
  - arc (u, f) has lower bound $\lambda(u, f) = 1$, capacity $\mu(u, f) = 4$, and cost $\chi(u, f) = 0$ (see Fig. 16.a);
  - arc (f, g) has lower bound $\lambda(f, g) = 0$, capacity $\mu(f, g) = +\infty$, and cost $\chi(f, g) = 1$ (see Fig. 16.b).

The intuition behind the definition of flow network $\mathcal{N}$ is as follows:

- the flow in arc (u, f) associated with dart (u, v) represents the quantity $\alpha(u, v)$, i.e., the measure of an angle formed at vertex u inside face f;
- the flow in arc (f, g) associated with dart (u, v) represents the quantity $\beta(u, v)$, i.e., the number bends with the $\pi/2$ angle in face f along an edge between faces f and g;
- the conservation of flow at a vertex-node represents the geometric fact that the sum of the measures of the vertex-angles around a vertex is equal to 2π;
- the conservation of flow at a face-node represents the geometric fact that the sum of the measures of the vertex-angles and bend-angles inside an internal face f is equal to $\pi(p-2)$, where p is the total number of such angles (if f is the external face, then the above sum is equal to $\pi(p+2)$);
- the cost of the flow is equal to the number of bends.

In the example of Fig. 17, we show the flow associated with a given orthogonal representation.

An orthogonal representation of G with the minimum number of bends can be computed as follows:

1. Construct the flow network $\mathcal{N}$ associated with G.
2. Compute a flow ϕ of minimum cost for network $\mathcal{N}$.
3. Compute the orthogonal representation of G associated with ϕ.

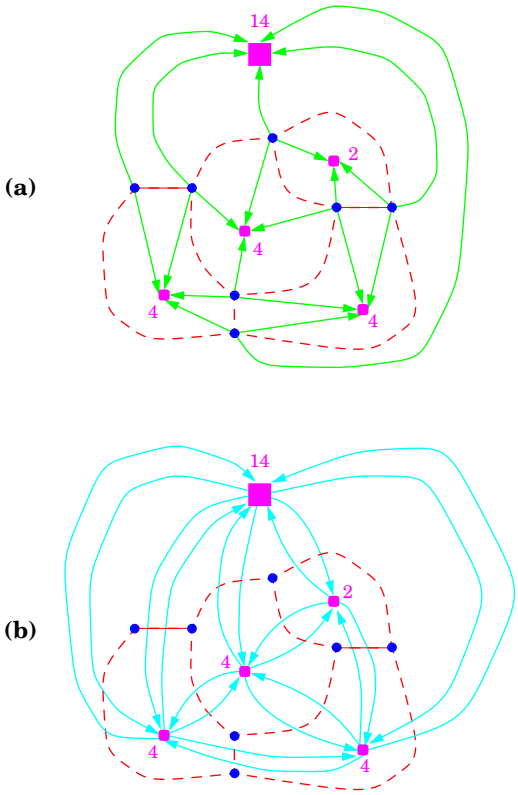

Figure 16. Network $\mathcal{N}$ associated with an embedded planar graph G (shown with dashed lines). Each face-node f of $\mathcal{N}$ is labeled with the amount of flow $\sigma(f)$ consumed. (a) Arcs of $\mathcal{N}$ from vertex-nodes to face-nodes. (b) Arcs of $\mathcal{N}$ between face-nodes.

THEOREM 5 *Let G be an embedded planar graph with n vertices. An orthogonal representation of G with the minimum number of bends can be computed in time $O(T(n))$, where $T(n)$ is the time for computing a minimum cost flow in the flow network $\mathcal{N}$ associated with G.*

In Fig. 18, we show the orthogonal representation with the minimum number of bends associated with a flow of minimum cost.

A simple algorithm for computing a minimum cost flow in network $\mathcal{N}$ is based on the standard technique of augmenting the flow along minimum cost paths (Ahuja, Magnanti, and Orlin (1993). It runs in time $T(n) = O(n^2 \log n)$ using $O(n)$ space. A more complex algorithm, which exploits the sparsity of network $\mathcal{N}$, runs in time $T(n) = O(n^{7/4} \log n)$ using $O(n)$ space (Garg and Tamassia, 1997).

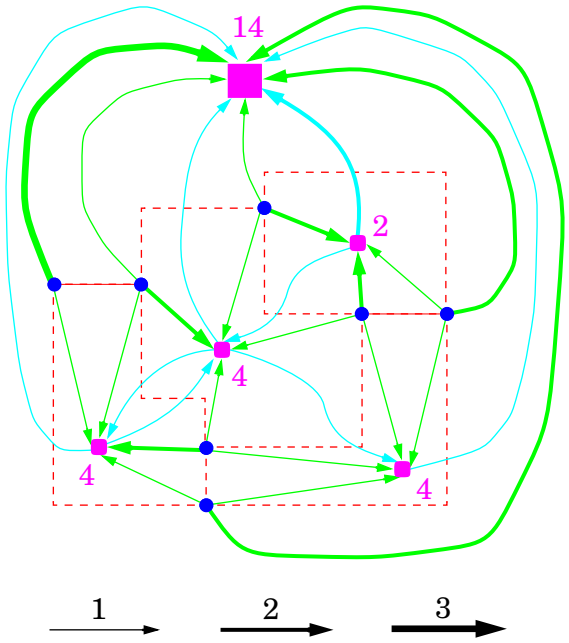

Figure 17. Example of flow ϕ associated with a planar orthogonal representation (drawn with dashed lines). Only the arcs with nonzero flow are shown. The thickness of the arc is proportional to the amount of flow. Each face-node f is labeled with the amount of flow $\sigma(f)$ consumed.

6.3. Algorithm for Bend Minimization

Algorithm 6.3 (*Optimal-Orthogonal*), shown below, constructs a planar orthogonal drawing with the number of bends. The computation of Step 2 can be carried out in $O(n)$ time. Thus, the time complexity of Algorithm 6.3 (*Optimal-Orthogonal*) is dominated by the minimum cost flow computation.

THEOREM 6 *Given an embedded planar graph G with n vertices of degree at most four, Algorithm 6.3 (Optimal-Orthogonal) constructs in $O(T(n))$ time a planar orthogonal grid drawing of G with area $O(n^2)$ and the minimum number of bends, where $T(n)$ is the time for computing a minimum cost flow in the flow network $\mathcal{N}$ associated with G.*

Algorithm 6.3 (*Optimal-Orthogonal*) can be extended to graphs with vertex degree higher than 4, and can be used in conjunction with the planarization approach (see Section 3) to construct orthogonal drawings of general graphs (Tamassia, Di Battista, and Batini, 1988).

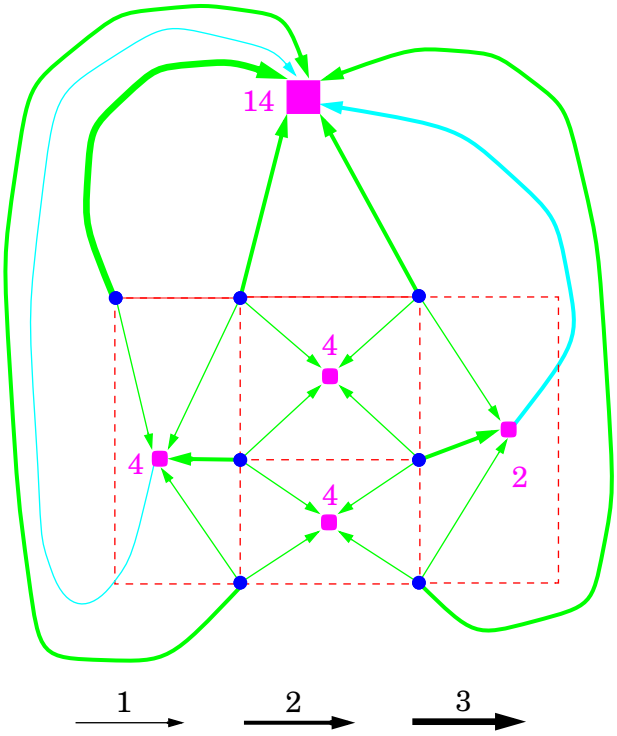

Figure 18. Example of planar orthogonal representation with the minimum number of bends (drawn with dashed lines) associated with a flow ϕ of minimum cost. Only the arcs with nonzero flow are shown. The thickness of the arc is proportional to the amount of flow. Each face-node f is labeled with the amount of flow $\sigma(f)$ consumed.

Algorithm *Optimal-Orthogonal*

Input: embedded planar graph G with n vertices of degree at most four

Output: planar orthogonal grid drawing Γ of G with area $O(n^2)$ and the minimum number of bends

1. Construct an orthogonal representation H of G with the minimum number of bends.
2. Construct a planar orthogonal grid drawing Γ of H.

□

6.4. Constraints

Algorithm 6.3 (*Optimal-Orthogonal*) can be easily modified to support user-defined constraints of the following types on the drawing:

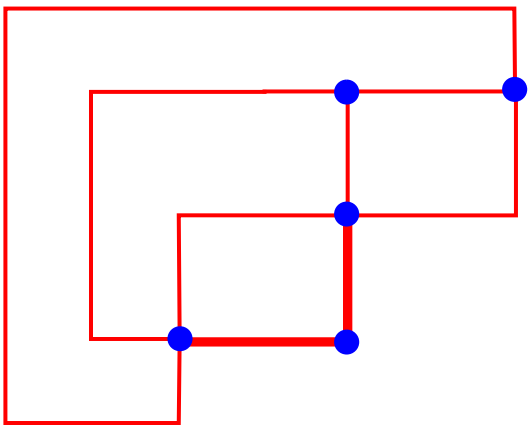

Figure 19. Planar orthogonal drawing of the embedded planar graph of Fig. 14. The drawing has the minimum number of bends subject to bend constraints requiring that the two edges drawing with thick lines have zero bends. Imposing this constraint causes the minimum number of bends to increase from 6 to 8.

- *vertex-angle constraints*: upper and lower bounds on a vertex-angle, i.e., on $\alpha(u, v)$, for a dart (u, v);

- *bend constraints*: upper and lower bounds on the bends of an edge e, i.e., upper and lower bounds on $\beta(u, v)$ and $\beta(v, u)$ for the darts (u, v) and (v, u) of edge e.

These constraints can be imposed by modifying the capacities and lower bounds of the arcs of $\mathcal{N}$ associated with appropriate darts. For example, to prescribe at most two bends on an edge e with darts (u, v) and (v, u), we set equal to two the capacity of the arcs (f, g) and (g, f) of $\mathcal{N}$ associated with the faces f and g on the left of (u, v) and (v, u), respectively, i.e., $\mu(f, g) = \mu(g, f) = 2$. Note that a set of vertex-angle and bend constraints allow to prescribe the orthogonal representation of a subgraph. The time complexity of Algorithm 6.3 (*Optimal-Orthogonal*) is not affected by the constraint-satisfaction mechanism.

We show in Fig. 19 a drawing with the minimum number of bends subject to bend constraints requiring two given edges to have zero bends.

7. Constraint-Based Drawing Approaches

A formalism for the declarative specification of drawings of graphs with Prolog predicates and an associated constraint solving mechanism have been developed by Kamada (1989). Using this formalism, one can express several simple geometric constraints among the vertices, such as horizontal or vertical alignment, circular arrangement, and relative x- or y-ordering. Each constraint can be specified as being rigid or "pliable". Rigid constraints must be satisfied exactly, while pliable constraints can be satisfied approximately. The constraint-solving mechanism is based on the least square method.

Dengler, Friedell, and Marks (1993), Kosak, Marks, and Shieber (1994), Marks (1991) provide a notation for describing the desired perceptual organization of a layout of a graph by means of a collection of layout patterns called *visual organization features*, which include clustering, zoning, sequential placement, T shape, and hub shape. They also present three methods that take as input a graph and a set of visual organization features, and produce an aesthetically pleasing drawing that exhibits the specified visual organization features. The first method is rule-based and implemented in Prolog. The second method is a genetic algorithm to be executed by a massively parallel computer. The third, method (Dengler, Friedell, and Marks, 1993) incrementally improves an initial randomly-generated drawing using a force-directed technique, and is the most practical. This approach has also been used within an interactive graph drawing system (Ryall, Marks, and Shieber, 1997).

Luders et al. (1995) present a combinatorial approach for satisfying inequality constraints between vertex coordinates within a graph drawing system. Kamps et al. (1996) show how to extend force-directed methods to support the following geometric constraints: fixed vertex positions, fixed distances between pairs of vertices, relative x- or y-ordering, and horizontal or vertical alignment.

A comprehensive approach to constrained graph drawing is presented by He and Marriott (1997). They give a general model that supports:

- the specification of arbitrary arithmetic linear equality and inequality constraints on the coordinates of the vertices; and
- suggested coordinates for the vertices, each with an associated weight, which denotes the strength of the suggestion.

They show how to extend the force-directed approach by Kamada and Kawai (1989) to support such constraints using a technique based on the "active set method" that is fast and gives good results in practice. They also present a simplified approach for constrained drawings of rooted trees.

In related work, Eades and Lin (1995) attempt at combining algorithmic and constraint-based declarative methods in drawings of trees, and Brandenburg presents a comprehensive approach to graph drawing based on graph grammars (Brandenburg, 1995), where drawings are generated by rule-based methods.

8. Visual Graph Drawing

A visual approach to graph drawing, where the layout of a graph is pictorially specified "by example," is proposed by Cruz and Garg (1995). Within this approach, a graph is stored in an object-oriented database, and its drawing is visually defined using recursive rules of the visual meta-language DOODLE (Cruz, 1992). The application of the visual rules to the input graph yields a system of constraints that can be solved in linear time for the following types of drawings:

- layered drawings and box inclusion drawings of binary trees;
- Δ-drawings of series-parallel digraphs (Bertolazzi et al., 1994);

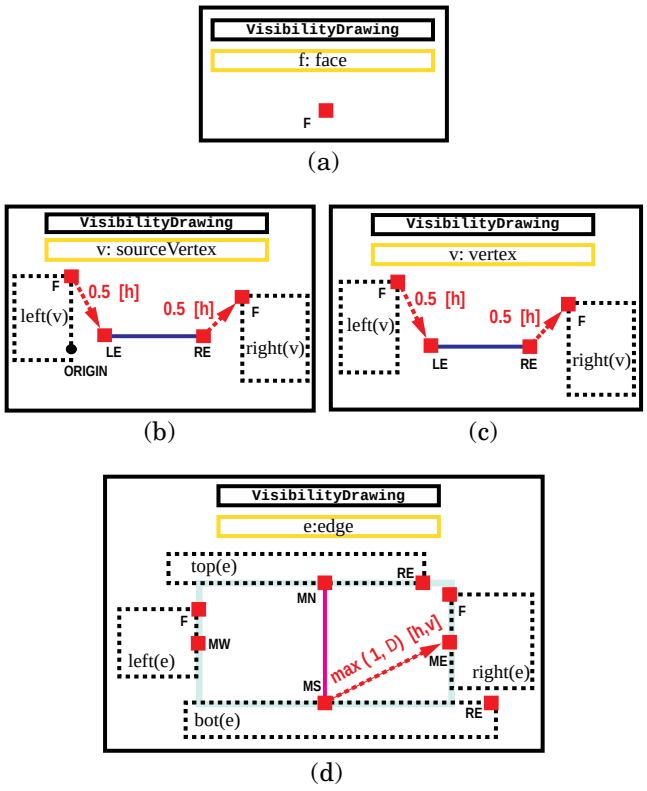

Figure 20. Visual rules for constructing a visibility representation of a planar st-digraph: (a) rule for a face; (b) special rule for the source vertex; (c) rule for a vertex; (d) rule for an edge.

- polyline drawings (Di Battista and Tamassia, 1988), visibility drawings (Tamassia and Tollis, 1986), and tessellation drawings (Tamassia and Tollis, 1989) of upward planar digraphs.

Complete visual programs for visibility representations and upward polyline drawings are shown in Figures 20 and 21, respectively.

We assume that the vertices, edges, and faces of the input planar st-digraph G are database objects, where for each object o the following attributes describing the embedding are stored: left face $left(o)$, right face $right(o)$, bottom vertex $bot(o)$, and top vertex $top(o)$. Note that the value of each attribute is another database object. Each rule defines the visual representation of a database object of a certain class (vertex, edge, and face).

The visual notation in a rule that applies to an object o includes:

- geometric figures that give the visual representation of object o, such as circles, segments, and rectangles;

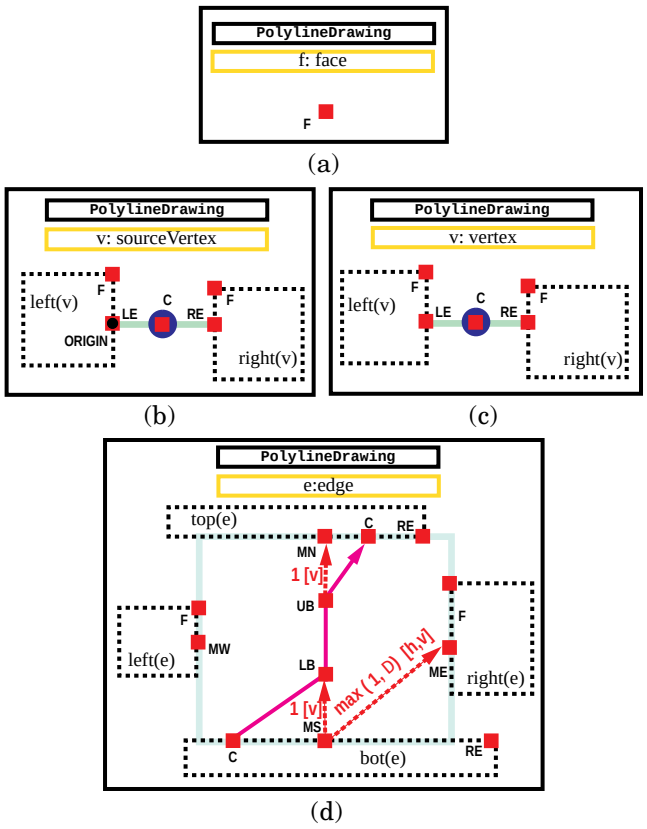

Figure 21. Visual rules for constructing an upward polyline drawing of a planar *st*-digraph: (a) rule for a face; (b) special rule for the source vertex; (c) rule for a vertex; (d) rule for an edge.

- references to the visual representation of other objects given by attributes of *o*, denoted with dashed boxes labeled by the attribute;
- landmarks of the visual representations of *o* and of other referenced objects, shown as small squares with labels (e.g., MS, the "middle South" landmark, denotes the middle point of the bottom edge of a rectangle); and
- landmarks of the coordinate system, shown with small circles (e.g., ORIGIN denotes point (0, 0));
- explicit constraints between landmarks, shown as arrows joining two landmarks with labels defining the constraint imposed on the coordinates of the landmarks (e.g., in rule (d), the dashed arrow with label max(1,Δ)[h,v] is an explicit constraint specifying minimum horizontal and vertical distance 1 from the "midpoint South" MS to the "midpoint East" of the rectangle);

- implicit constraints between landmarks, given by their horizontal or vertical alignment (e.g., in rule (d), the "midpoint East" ME of the rectangle associated with edge e and the "top endpoint" TE of the referenced visual representation of the right face of e $right(e)$ must have the same x-coordinate because they are drawn vertically aligned).

In these two programs, the visual representation of the faces is a single point associated with landmark F, which is invisible but contributes to the definition of the constraints. Also, the visual representation of an edge includes a visible portion (vertical segment for a visibility representation and polygonal chain with three segments for an upward polyline drawing) and an invisible portion drawn with a conventional "transparent color" (a rectangle or segment with shaded lines in the figures).

9. Conclusions

Because of the importance of constraint satisfaction in practical graph visualization applications, the study of constraints is a strategic research direction for the graph drawing field.

In this paper, we have overviewed fundamental algorithmic techniques for drawing graphs, and we have shown how they can be extended to support a limited constraint satisfaction capability. The force-directed drawing approach appears to be the most suited for the satisfaction of geometric constraints, while topological and shape constraints seem to be better handled by techniques based on planarization and network flow. We have also mentioned methods for the specification of constraints and declarative rule-based approaches for constraint resolution.

10. Acknowledgment

I would like to thank Ioannis G. Tollis for useful comments.

References

R. K. Ahuja, T. L. Magnanti, & J. B. Orlin. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ.

P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, & I. G. Tollis. (1994). How to draw a series-parallel digraph. *Internat. J. Comput. Geom. Appl.* 4: 385–402.

F. J. Brandenburg. (1995). Designing graph drawings by layout graph grammars. In R. Tamassia and I. G. Tollis (eds.), *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 416–427. Springer-Verlag.

F. J. Brandenburg, M. Himsolt, & C. Rohrer. (1996). An experimental comparison of force-directed and randomized graph drawing algorithms. In F. J. Brandenburg (ed.), *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 76–87. Springer-Verlag.

I. F. Cruz. (1992) DOODLE: A visual language for object-oriented databases. In *Proc. ACM SIGMOD*, pages 71–80.

I. F. Cruz & A. Garg. (1995). Drawing graphs by example efficiently: Trees and planar acyclic digraphs. In R. Tamassia and I. G. Tollis (eds.), *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 404–415. Springer-Verlag.

R. Davidson & D. Harel. (1996). Drawing graphics nicely using simulated annealing. *ACM Trans. Graph*. 15(4): 301–331.

E. Dengler, M. Friedell, & J. Marks. (1993). Constraint-driven diagram layout. In *Proc. IEEE Sympos. on Visual Languages*, pages 330–335.

G. Di Battista, P. Eades, R. Tamassia, & I. G. Tollis. (1994). Algorithms for drawing graphs: An annotated bibliography. *Comput. Geom. Theory Appl*. 4: 235–282.

G. Di Battista & R. Tamassia. (1988). Algorithms for plane representations of acyclic digraphs. *Theoret. Comput. Sci*. 61: 175–198.

G. Di Battista, R. Tamassia, & I. G. Tollis. (1992). Constrained visibility representations of graphs. *Inform. Process. Lett*. 41: 1–7.

P. Eades. (1984). A heuristic for graph drawing. *Congr. Numer*. 42: 149–160.

S. Even. (1979). *Graph Algorithms*. Computer Science Press, Potomac, Maryland.

A. Frick, A. Ludwig, & H. Mehldau. (1995). A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I. G. Tollis (eds.), *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 388–403. Springer-Verlag.

T. Fruchterman & E. Reingold. (1991). Graph drawing by force-directed placement. *Softw. – Pract. Exp*. 21(11): 1129–1164.

M. R. Garey & D. S. Johnson. (1983). Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods* 4(3): 312–316.

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

A. Gibbons. (1980). *Algorithmic Graph Theory*. Cambridge University Press, Cambridge.

D. Harel & M. Sardas. (1995). Randomized graph drawing with heavy-duty preprocessing. *J. Visual Lang. Comput*. 6(3). (Special issue on Graph Visualization, edited by I. F. Cruz and P. Eades.)

W. He & K. Marriott. (1997). Constrained graph layout. In S. North (ed.), *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 217–232. Springer-Verlag.

M. Jünger & P. Mutzel. (1996). Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica* 16(1). (Special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia.)

T. Kamada. (1989). *Visualizing Abstract Objects and Relations*. World Scientific Series in Computer Science.

T. Kamada & S. Kawai. (1989). An algorithm for drawing general undirected graphs. *Inform. Process. Lett*. 31: 7–15.

T. Kamps, J. Kleinz, & J. Read. (1996). Constraint-based spring-model algorithm for graph layout. In F. J. Brandenburg (ed.), *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 349–360. Springer-Verlag.

C. Kosak, J. Marks, & S. Shieber. (1994). Automating the layout of network diagrams with specified visual organization. *IEEE Trans. Syst. Man Cybern*. 24(3): 440–454.

J. B. Kruskal & J. B. Seery. (1980). Designing network diagrams. In *Proc. First General Conference on Social Graphics*, pages 22–50. U.S. Department of the Census.

T. Lin & P. Eades. (1995). Integration of declarative and algorithmic approaches for layout creation. In R. Tamassia & I. G. Tollis (eds.), *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 376–387. Springer-Verlag.

P. Luders, R. Ernst, & S. Stille. (1995). An approach to automatic display layout using combinatorial optimization. *Software-Practice and Experience* 25(11): 1183–1202.

J. Marks. (1991). A formal specification for network diagrams that facilitates automated design. *J. Visual Lang. Comput*. 2: 395–414.

K. Mehlhorn. (1984). *Data Structures and Algorithms*. Volumes 1–3. Springer-Verlag.

T. Nishizeki & N. Chiba. (1988). Planar graphs: Theory and algorithms. *Ann. Discrete Math*. 32.

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

K. Ryall, J. Marks, & S. Shieber. (1997). An interactive system for drawing graphs. In S. North (ed.), *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 387–393. Springer-Verlag.

K. Sugiyama & K. Misue. (1995). Graph drawing by magnetic-spring model. *J. Visual Lang. Comput*. 6(3). (Special issue on Graph Visualization, edited by I. F. Cruz and P. Eades.)

R. Tamassia. (1985). New layout techniques for entity-relationship diagrams. In *Proc. 4th Internat. Conf. on Entity-Relationship Approach*, pages 304–311.

R. Tamassia. (1987). On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput*. 16(3): 421–444.
R. Tamassia. (1997). Graph drawing. In J. E. Goodman & J. O'Rourke (eds.), *CRC Handbook of Discrete and Computational Geometry*. CRC Press.
R. Tamassia, G. Di Battista, & C. Batini. (1988). Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern*. SMC-18(1): 61–79.
R. Tamassia & I. G. Tollis. (1986). A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom*. 1(4): 321–341.
R. Tamassia & I. G. Tollis. (1989). Tessellation representations of planar graphs. In *Proc. 27th Allerton Conf. Commun. Control Comput.*, pages 48–57.
W. T. Tutte. (1960). Convex representations of graphs. *Proceedings London Mathematical Society* 10(3): 304–320.
W. T. Tutte. (1963). How to draw a graph. *Proceedings London Mathematical Society* 13(3): 743–768.