Drawing Graphs
Methods and Models

Michael Kaufmann and Dorothea Wagner (Eds.)

October 13, 1999

Contents

1 Drawing Planar Graphs

René Weiskircher

1.1 Introduction

When we want to draw a graph to make the information contained in its structure easily accessible, it is highly desirable to have a drawing with as few edge crossings as possible. The class of graphs that can be drawn with no crossings at all is the class of *planar graphs*. Algorithms for drawing planar graphs are the main subject of this chapter.

First we will give some necessary definitions and some basic properties of planar graphs. In section 1.3, we will take a closer look at two linear time algorithms for testing if a graph is planar. When a graph is not planar but we want to apply an algorithm for drawing planar graphs, we can transform the graph into a similar planar graph. Section 1.4 gives an overview of methods for doing this.

Most drawing algorithms presented in this chapter need a 2-connected planar graph as input. If a planar graph does not have this property, we can add edges to make it 2-connected and planar. Section 1.5 describes ways to accomplish this. The following sections describe drawing algorithms for planar graphs. Section 1.6 treats the generation of convex straight-line representations, while section 1.7 gives an overview of some algorithms that use a special ordering of the vertices of a graph called a *canonical* ordering.

1.2 What is a Planar Graph?

To define what we mean by the term *planar graph* we first have to define what is meant by the term *planar representation*.

Definition 1.1 (planar representation) *A* planar representation D *of a graph* $G = (V, E)$ *is a mapping of the vertices in* V *to points in the plane and of the edges in* E *to open Jordan curves with the following properties:*

- *The representation of edge* $e = (v_1, v_2)$ *connects the representation of* v_1 *with the representation of* v_2 *for all edges* $e \in E$.

- *The representations of two disjoint edges* $e_1 = (v_1, v_2)$ *and* $e_2 = (v_3, v_4)$ *have no common points.*
- *The representation of edge* $e = (v_1, v_2)$ *does not contain the representation of* $v_3 \in V$ *with* $v_3 \notin \{v_1, v_2\}$.

If D is a planar representation, the set $\mathbb{R}^2 - D$ is open and its regions are called the *faces* of D. Since D is bounded, exactly one of the faces of D is unbounded. This face is called the *outer face* of D.

Using the definition of planar representations, it is easy to define the term *planar graph*.

Definition 1.2 (planar graph) *A graph* G *is planar if and only if there exists a planar representation of* G.

There is an infinite number of different planar representations of a planar graph. We can define a finite number of equivalence classes of planar representations of the same graph using the term *planar embedding*.

Definition 1.3 (planar embedding) *Two representations* D_1 *and* D_2 *of a planar graph* G *realize the same planar embedding of* G*, if and only if the following two conditions hold:*

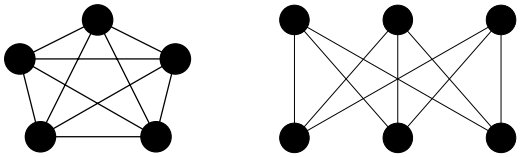
- *The simple cycles of* G *that bound the faces of* D_1 *are the same cycles that bound the faces of* D_2.
- *The outer face in* D_1 *is bounded by the same cycle of* G *as in* D_2.

The definition of a planar graph above is very simple but it is a geometric definition. Since the set of all planar representations of a graph is infinite and uncountable, it is not immediately clear how to test a graph for planarity. Kuratowski found a combinatorial description of planar graphs but to present this description, we have to define the *subdivision* of a graph.

Definition 1.4 (subdivision) *A* subdivision *of a graph* $G = (V, E)$ *is a graph* $G' = (V', E')$ *that can be obtained from* G *by a sequence of split operations where we insert a new vertex* u *and replace an edge* $e = (v_1, v_2)$ *by the two edges* $e_1 = (v_1, u)$ *and* $e_2 = (u, v_2)$.

Thus, a subdivision of a graph is another graph where some edges of the original graph have been replaced by paths where every vertex has degree two. Planar graphs are now characterized by the following:

Theorem 1.5 *A graph* G *is planar if and only if it does not contain a subdivision of* K_5 *(the complete graph with 5 vertices, see Figure 1.1(a)) or* $K_{3,3}$ *(the complete bipartite graph with 3 vertices in each set, see Figure 1.1(b)).*



(a) K_5 (b) $K_{3,3}$

Figure 1.1: The basic non-planar graphs

If a graph G is directed (each edge is an ordered pair of vertices), we can define a more restricted class of planar graphs, the *upward planar* graphs. First we define the term *upward representation.*

Definition 1.6 (upward representation) *Let $G = (V, E)$ with $E \subseteq V \times V$ be a directed graph. A representation of G is called* upward *if the representation of every edge is monotonically nondecreasing in the y-direction.*

We use this definition to define *upward graphs* and *upward planar graphs.*

Definition 1.7 (upward graph) *A directed graph is* upward *if and only if it admits an upward representation.*

Definition 1.8 (upward planar graph) *A directed graph is* upward planar *if and only if it admits an upward and planar representation.*

It is possible to test in linear time whether a directed graph admits an upward representation (because only acyclic graphs admit such a representation) and, as we will see in the next section, we can test in linear time, whether a graph admits a planar representation. But testing whether a graph admits an upward planar representation is for general graphs $\mathcal{NP}$-complete (Garg and Tamassia, 1994).

A survey about upward planarity testing can be found in Garg and Tamassia (1994). We will not treat the topic in this chapter, but algorithms for drawing upward graphs can be found in Di Battista and Tamassia (1988).

1.3 Planarity Testing

The first algorithm for testing whether a given graph is planar was developed by Auslander and Parter (Auslander and Parter, 1961) and Goldstein

(Goldstein, 1963). Hopcroft and Tarjan improved this result to linear running time (Hopcroft and Tarjan, 1974). Another linear time algorithm for planarity testing was developed by Lempel, Even and Cederbaum (Lempel et al., 1967) and Booth and Lueker (Booth and Lueker, 1976). We will only give a short overview of the two linear time algorithms.

1.3.1 The Algorithm of Hopcroft and Tarjan

This overview of the algorithm follows the one in Mutzel (1994). In principle, the algorithm works as follows: Search for a cycle C whose removal disconnects the graph. Then check recursively whether the graphs that are constructed by merging the connected components of $G - C$ and the cycle C are planar. In a second step, combine the computed embeddings for the components to get a planar embedding of the whole graph, if possible.

The algorithm needs a depth first search tree $G' = (V, T, B)$, where V is the set of DFS numbers of the vertices in G, T is the set of *tree edges* of the depth first search tree and B the set of *back edges* (for DFS trees see Mehlhorn (1984)). We assume that G is 2-connected (this is not a restriction, because a graph is planar if and only if all its 2-connected components are planar).

Let C be a *spine cycle* of G, which is a cycle consisting of a path of tree edges starting at the root (vertex 1) of the DFS tree followed by a single back edge back to the root vertex. Because G is 2-connected, such a cycle must exist. We assume that removing all edges of C splits G into the subgraphs $G_1, G_2, \ldots, G_k$. We define the graphs G'_i for $1 \leq i \leq k$ as the graph G_i together with the cycle C and all the edges in G between a vertex in G_i and a vertex on C. First, we recursively check whether each G'_i is planar and compute a planar embedding for it. Planar embeddings are equivalence classes of planar representations that describe the topology of the representation but not the length and shape of edges or the position of vertices (see definition 1.3).

The planar embeddings of the G'_i must have all edges and vertices of C on the outer face. Now we assume that we have found a suitable embedding for each G'_i. We must test whether we can combine these embeddings to a planar embedding of G. The reason why this may fail is that each G_i shares at least two vertices with C. Figure 1.2 shows how this fact can make it impossible to embed two graphs G_i and G_j on the same side of C. We say that the two graphs *interlace*.

To test whether there is an assignment of the G_i's to the two sides of C so that the resulting representation is planar, we build the *interlace graph* I_G. This graph has one vertex for each G_i and two vertices are adjacent if and only if they interlace. We can only draw G planar if I_G is bipartite. If there is an embedding with the necessary properties for each G'_i and the interlace graph is bipartite, we know that G is planar and we can construct a planar

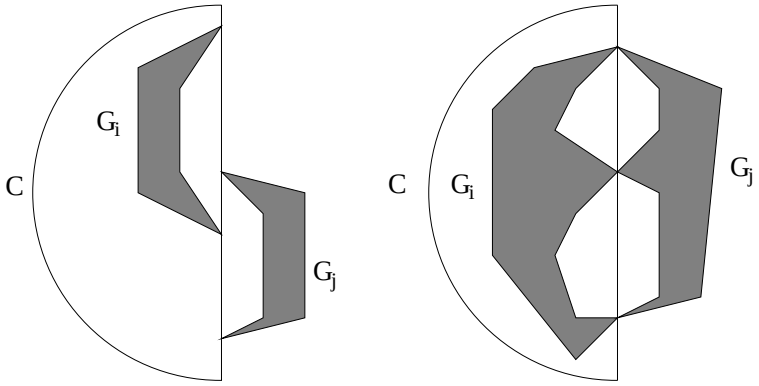


Figure 1.2: Interlacing graphs G_i and G_j that can't be embedded on the same side of C

embedding for it.

1.3.2 The Algorithm of Lempel, Even and Cederbaum

A vertex-based method for planarity testing is the test developed by Lempel, Even and Cederbaum (Lempel et al., 1967; Even, 1979). We say that this test is vertex based because we add the vertices one by one to a special data structure and check after each step if the information seen so far proves that the graph is non-planar. This test runs in linear time, like the algorithm of Hopcroft and Tarjan discussed before.

The input of the algorithm is again a 2-connected graph $G = (V, E)$. We assume $V = \{v_1, v_2, \ldots, v_n\}$ where the numbering of the vertices is an st-numbering.

Definition 1.9 (st-Numbering) *Given an edge $\{s, t\}$ in a graph $G = (V, E)$ with n vertices, an st-numbering is a function $g : V \to \{1, \ldots, n\}$, such that*

- $g(s) = 1$, $g(t) = n$
- $\forall v \in V \setminus \{s, t\}\ \exists u, w \in V \quad (\{u, v\}, \{v, w\} \in E \ \wedge \ g(u) < g(v) < g(w))$

Lempel, Even and Cederbaum showed that for every edge $\{s, t\}$ in an arbitrary graph G, there exists an st-numbering if and only if G is biconnected. A linear time algorithm solving this problem is given in Even (1979).

We define G_k as the subgraph of G induced by the vertices with indices 1 to k. This graph is extended to a graph B_k. For each edge $(u, v) \in E$ with u in G_k and v not in G_k the graph B_k has a new *virtual vertex* and an edge connecting v to this vertex. So there may be several virtual vertices in B_k that

correspond to the same vertex in G. The idea of the algorithm is to check whether we can identify the virtual vertices corresponding to the same vertex in G without losing the planarity property.

If G is planar, B_k has a planar embedding where each vertex v_i for $1 \leq i \leq k$ is drawn on y-coordinate i, all virtual vertices are placed on y-coordinate $k+1$ and all edges are disjoint y-monotone curves (which means that they are only intersected at most once by any horizontal line). Such a representation is called a *bush form*. Figure 1.3 shows an example for a bush form.

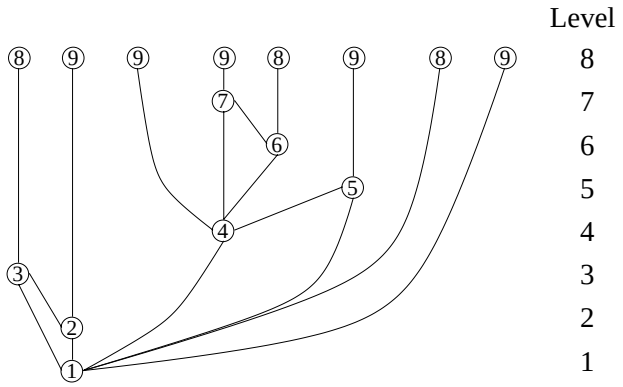


Figure 1.3: A bush form

Let v_i be a vertex in a bush form. If the removal of v disconnects the bush form, we call it a *split vertex*. Let B' be the bush-form after the removal of v_i. The *split-components* of v_i are all the connected components of B', where the indices of all vertices are greater than i. Now consider the bush form in Figure 1.3. Since the labels of the vertices are their st-numbers, it shows the bush form B_7. When we want to draw B_8, we must first transform B_7 so that all virtual vertices with label 8 form a consecutive sequence on level 8. This can be done by flipping the split component of vertex 1 which includes the vertices 2 and 3 around so that the virtual vertices labeled 8 and 9 in the split component swap their positions. We also have to move the virtual vertex labeled 9 adjacent to vertex 4 to the right and flip the split component of vertex 4 with the vertices 6 and 7. The resulting graph is shown in Figure 1.4.

If v is a split vertex of a bush form (which means that removing v disconnects the bush form), then we can freely permute the split components which have vertices with higher st-number than v and we can flip each individual component. There may be several possible ways of producing a consecutive sequence of the vertices labeled $k+1$ and since not all may eventually lead to a planar representation of G, we have to keep track of all of them. This can be done in linear time using a data structure called *PQ-tree* as proposed

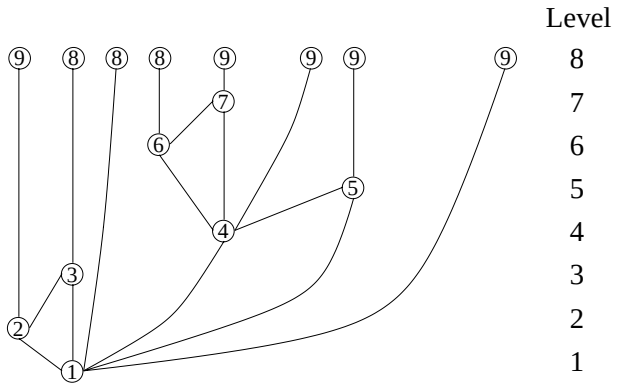


Figure 1.4: The bush form from Figure 1.3 has been transformed so that all vertices labeled 8 form a consecutive sequence

by Booth and Lueker (Booth and Lueker, 1976). If it is not possible to make the vertices labeled $k+1$ consecutive, we know that the graph is not planar. Otherwise, the algorithm will produce a planar embedding of the graph. In Mehlhorn and Näher (1999) a detailed description of the complete algorithm can be found.

1.4 How to Make a Graph Planar

There are many popular algorithms for drawing planar graphs and they produce a great variety of different styles of representations. Therefore, it makes sense to transform a non-planar graph into a similar planar graph, apply a graph drawing algorithm for planar graphs to the result and then modify the resulting representation so that it becomes a representation of the original non-planar graph. A survey of methods for doing this can be found in Liebers (1996).

Quite a drastic way of making a graph planar is to delete vertices. This method is not used very much in graph drawing, because deleting vertices changes a graph quite considerably. The problem of deciding for an integer k if we can make a non-planar graph planar by deleting at most k vertices is $\mathcal{NP}$-complete (Lewis and Yannakakis, 1980).

Another way of making a graph planar is to split vertices. This is a rather complex operation, so we will give the formal definition from Liebers (1996).

Definition 1.10 (vertex splitting) *Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. Then we say G' has been obtained by splitting vertex v of G into*

the vertices v_1 and v_2 if the following conditions are satisfied:

$$\begin{aligned} V &= (V' \backslash \{v_1, v_2\}) \cup \{v\} \\ E &= (E' \backslash \{uv_i \mid u \in V' \text{ and } uv_i \in E' \text{ for } i \in \{1,2\}\} \\ &\quad \cup \{uv \mid u \in V \backslash \{v\} \text{ and } (uv_1 \in E' \vee uv_2 \in E')\} \end{aligned}$$

Splitting a vertex is also a drastic operation and is not commonly used in graph drawing to planarize graphs. To decide if a non-planar graph can be made planar by at most k vertex-splitting operations is $\mathcal{NP}$-complete (Faria et al., 1998).

Two more commonly used ways of transforming a non-planar graph into a planar graph are the insertion of new vertices and the deletion of edges.

1.4.1 Inserting Vertices

Assume we have a non-planar graph G and a representation D of G with k crossings. Then we can transform G into a planar graph G' in the following way:
Let $e = (u, v)$ and $f = (x, y)$ be two edges that cross in D. Then we can add a new vertex v_c to G, remove the edges e and f from G and insert the four new edges $e_1 = (u, v_c)$, $e_2 = (v_c, v)$, $f_1 = (x, v_c)$ and $f_2 = (v_c, y)$. This is equivalent to replacing the crossing in D between e and f by the new vertex v_c. If we do this for every pair of crossing edges, we will transform G into a planar graph G' and D into a planar representation D' of G'.

Since the graph G' is planar, we can draw it by using any algorithm for drawing planar graphs. If D'' is the resulting representation, we can transform this representation into a representation of the original non-planar graph G by replacing all the vertices we introduced by crossings again. Since we want to have as few crossings as possible in the resulting representation, we want to introduce as few new vertices as possible.

The minimum number of vertices we have to insert is equal to the minimum number of crossings in any representation of G. But the problem of deciding for a graph G if it can be drawn with at most k crossings is $\mathcal{NP}$-complete (Garey and Johnson, 1983). The only heuristics that are known for inserting the minimum number of vertices to construct a planar graph are the algorithms for drawing non-planar graphs. By inserting vertices at every crossing of the representation produced we get a planar graph.

1.4.2 Deleting Edges

If G is a non-planar graph, there is a non-empty subgraph of G which is planar. Each spanning tree of G is planar, since every acyclic graph is planar. So we

can derive a planar graph from a non-planar graph by deleting a subset of its edges. But the problem of deciding for a non-planar graph $G = (V, E)$ and a number $k < |E|$ if there is a planar subgraph with at least k edges is $\mathcal{NP}$-complete. This was independently shown by Liu and Geldmacher (Liu and Geldmacher, 1977), Yannakakis (Yannakakis, 1978) and Watanabe et al. (Watanabe et al., 1983). The associated $\mathcal{NP}$-hard maximization problem is to find a planar subgraph of a G with the property that there exists no other planar subgraph that has more edges. This problem is called the *maximum planar subgraph problem*. The problem of finding a planar subgraph, which is not a proper subgraph of another planar subgraph of G is called the *maximal planar subgraph problem* and is solvable in polynomial time.

Definition 1.11 (Maximal planar subgraph) *A maximal planar subgraph of a graph $G = (V, E)$ is a subgraph $G' = (V, E')$ of G in which there exists no edge in $E - E'$ that can be added to G' without losing planarity.*

One approach to solving this problem is to start with the subgraph $G_1 = (V, \emptyset)$ of G and to test for each edge if we can add it to the current solution without losing planarity. If we can do that, we add the edge and proceed to the next edge. Since we have to perform a planarity test for each edge of the graph and such a test can be implemented in linear time, this algorithm has a running time of $O(n \cdot m)$ where n is the number of vertices in the graph and m the number of edges.

Di Battista and Tamassia developed a data structure called SPQR-tree, which can be used for decomposing a planar 2-connected graph into 3-connected components and for fast online planarity testing (Di Battista and Tamassia, 1989; Di Battista and Tamassia, 1990; Di Battista and Tamassia, 1996). Using this data structure, they were able to develop an algorithm for finding a maximal planar subgraph in $O(m \log n)$ running time. There is also an algorithm with the same asymptotic running time developed by Cai, Han and Tarjan (Cai et al., 1993) which is based on the planarity testing algorithm in Hopcroft and Tarjan (1974).

La Poutré (Poutré, 1994) proposed an algorithm for incremental planarity testing yielding an algorithm for the maximal planar subgraph problem running in time $O(n + m \cdot \alpha(m, n))$ where $\alpha(m, n)$ is the inverse of the Ackermann function which grows very slowly. There are even two linear time algorithms for the problem, one by Djidjev (Djidjev, 1995) and one by Hsu (Hsu, 1995), which has the best asymptotic running time possible for solving the maximal planar subgraph problem.

A heuristic for the maximum planar subgraph problem is the Deltahedron heuristic (Foulds and Robinson, 1978; Foulds et al., 1985). This heuristic starts with the complete graph on 4 vertices (tetrahedron) as the initial planar

subgraph and then places the remaining vertices into the faces of the current planar subgraph. The sequence of the vertices depends on a chosen weight function. Leung (Leung, 1992) proposed a generalization of this method. The current planar subgraph has only triangular faces and in each step, we add a single vertex and 3 edges or we add 3 vertices and 9 edges. A list of other heuristics can be found in Liebers (1996).

Jünger and Mutzel (Mutzel, 1994; Jünger and Mutzel, 1996) proposed a branch and cut algorithm for solving the maximum planar subgraph problem based on an integer linear program that excludes the presence of subdivisions of $K_{3,3}$ and K_5 in the solution graph. The advantage of a branch and cut algorithm is that it either finds an optimum solution together with a proof of optimality or finds a solution together with an upper bound on the value of the optimum solution. For problems of moderate size (about 50 vertices), their approach finds an optimal solution in most cases.

1.5 How to Make a Planar Graph 2-Connected Planar

Many graph drawing algorithms only work for 2-connected or 3-connected graphs. This is true for most algorithms presented in this chapter. Therefore if we want to draw a graph which does not have the necessary connectivity property for applying a specific graph drawing algorithm, we can increase its connectivity by adding new edges (*Augmentation).* After a representation of the augmented graph has been computed, we remove the representations of the additional edges to get a representation of the original graph. Since we do not want to change the graph too much, we want to add a minimum number of edges in the augmentation step.

The planar augmentation problem is the problem of adding a minimum number of edges to a given planar graph so that the resulting graph is 2-connected and planar. Kant and Bodlaender (Kant and Bodlaender, 1991) introduced this problem and showed that it is $\mathcal{NP}$-hard. They have also given a 2-approximation algorithm running in time $O(n \log n)$ and a $\frac{3}{2}$-approximation algorithm with running time $O(n^2 \log n)$, where n is the number of vertices in the graph. However, the $\frac{3}{2}$-approximation algorithm is not correct (Kant, 1994) because there are problem instances where it computes only a 2-approximation.

Fialko and Mutzel developed a $\frac{5}{3}$-approximation algorithm (Fialko and Mutzel, 1998). The running time of the algorithm is $O(n^2 T)$ where T is the amortized time bound per insertion operation in incremental planarity testing. Using the algorithm in Poutré (1994), a running time of $O(n^2 \alpha(k, n))$ can be achieved where α is the inverse Ackermann function and k is $O(n^2)$. Recently, the algorithm has been improved by Mutzel to guarantee a $\frac{3}{2}$-approximation,

but this result has not yet been published.

The $\frac{5}{3}$-approximation algorithm works on the *block tree* of the graph we want to make 2-connected. The block tree has two types of vertices: The *b-vertices* correspond to the maximal 2-connected components of the graph and the *c-vertices* to the split vertices (as already mentioned, the removal of a split vertex disconnects the graph). We have an edge between a c-vertex and a b-vertex if and only if the corresponding split vertex belongs to the 2-connected component represented by the b-vertex. The idea is now to insert edges, merging paths of the block tree into single blocks until the tree has only one vertex and is thus 2-connected.

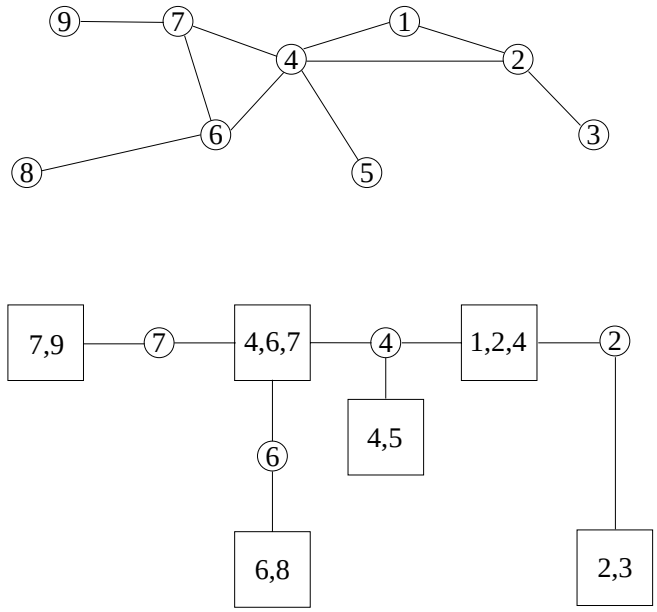


Figure 1.5: A graph and its block tree

A crucial role in the algorithm is played by the *pendants* of the block tree which are b-vertices with degree one. The algorithm connects pendants via edges if possible and otherwise connects pendants to non-pendant blocks. To achieve the approximation ratio, pendants are combined to form larger structures that are called *labels*. The algorithm looks at these labels in the order of decreasing number of pendants and tries to connect the pendants of two labels by introducing new edges. Inserting edges that connect the pendants of two labels is called a *label matching*.

The algorithm prefers certain matchings, but because the resulting graph has to be planar, not all of the preferred label matchings can be realized. Some labels can not be matched at all and so the algorithm introduces edges that

connect pendants of the same label and an additional edge from one of the pendants to a non-pendant vertex outside the label.

The approximation guarantee of the algorithm is tight which means that graphs exist for which the number of added edges is $\frac{5}{3}$ of the optimum number. On realistic instances, the algorithm performs very well and almost always finds a solution that uses at most one edge more than the optimum solution. This has been tested using a branch and cut algorithm for the planar augmentation problem developed by Mutzel (Mutzel, 1995) which is able to optimally solve instances of realistic size.

1.6 Convex Representations

Some planar graphs can be drawn in such a way that all cycles that bound faces are drawn as convex polygons. An example for such a drawing is shown in Figure 1.6. Such a representation is only possible if all face boundaries of the graph are simple cycles. Thus, a graph that is not 2-connected can have no convex representation. It has been shown that such a convex representation exists for all 3-connected graphs (Tutte, 1960) and Tutte gave an algorithm for producing representations of 3-connected graphs which involves solving $O(n)$ linear equations where n is the number of vertices in the graph (Tutte, 1963).

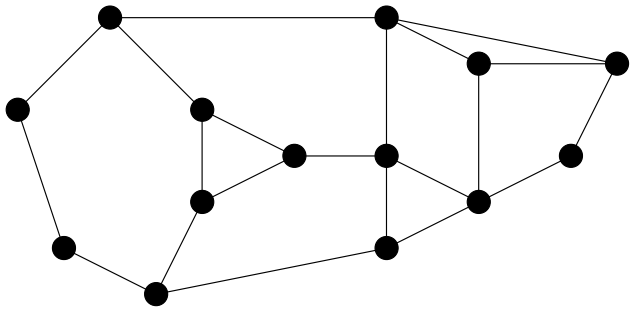


Figure 1.6: A convex drawing of a graph

Chiba, Yamanouchi and Nishizeki have developed an algorithm for producing a convex representation of a 2-connected planar graph (if it admits a convex representation) in linear time (Nishizeki and Chiba, 1988). In the same paper, they gave a linear time algorithm for testing whether a 2-connected planar graph admits a convex representation. The drawing algorithm is based on the proof of Tutte's result given by Thomassen (Thomassen, 1980). The testing algorithm works by dividing a 2-connected planar graph into 3-connected components as described in Hopcroft and Tarjan (1973) and testing planarity

of a special graph constructed from the original graph using the algorithm described in Hopcroft and Tarjan (1974).

To give a short outline of the drawing algorithm, we have to define what we mean by the term *extendible polygonal representation* of a *face cycle* of a graph G. A face cycle is a cycle in the graph that is the boundary of a face (region) of a planar representation of the graph. A convex representation S^* of a face cycle S is a convex polygon in which all vertices of S are drawn on the boundary of S^* and each apex of S^* is occupied by the representation of a vertex on S. The polygonal representation S^* of S is called extendible if there is a convex representation of G, in which S^* is the outer face of the representation.

Thomassen showed in Thomassen (1980), that the polygonal representation S^* of S is extendible if and only if the following conditions hold.

1. For each vertex v of G not on S, there are three vertex disjoint paths from v to vertices on S.

2. There are no connected components C in $G - S$, in which all vertices in S adjacent to a vertex in C are located on the same straight segment P of S^*.

3. There is no edge that connects two vertices on a straight segment of S^*.

4. Any cycle in G that does not share an edge with S has at least three vertices with degree greater than 2.

If the conditions above are satisfied, the following algorithm will correctly compute a convex representation of G.

The input of the algorithm `convex-draw` is a triple consisting of the graph G, a face cycle S of G and an extendible polygonal representation S^* of S.

Algorithm `convex-draw` (G, S, S^*):

1. We assume that G has more than 3 vertices, and some of them do not belong to S, otherwise, our problem is already solved. Select an arbitrary apex vertex v of S^* and set $G' = (G - v)$. Divide G' into the blocks $B_1, \ldots, B_p$ as shown in Figure 1.7 according to the cut vertices on S^*.

2. Draw each B_i convex applying the following procedure:

   (a) Let v_i and v_{i+1} be the cut vertices that split B_i from the rest of G'. Then these two vertices have already a fixed position, because they belong to S. These vertices also belong to the outer facial cycle S_i of B_i. We now draw all the vertices of S_i that do not belong to S on a convex polygon S_i^* inside the triangle given by the vertices v, v_i and

v_{i+1}. Each apex of the polygon is occupied by a vertex of S_i which is in G adjacent to v. The other vertices of S_i are drawn on the straight line segments of S_i^*.

(b) Recursively call the procedure `convex-draw` for all blocks with the arguments (B_i, S_i, S_i^*).

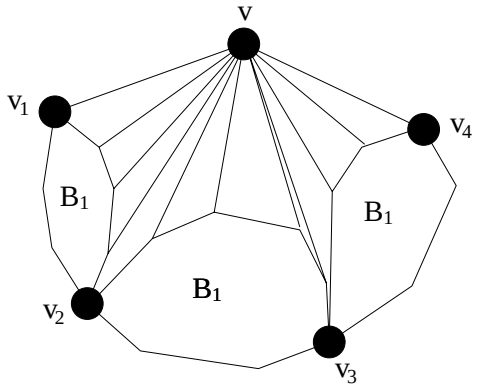


Figure 1.7: Recursive computation of a convex representation

The algorithm for testing whether a 2-connected planar graph has a convex representation relies on determining the *separation pairs* of the graph. A separation pair is a pair of vertices whose removal disconnects the graph.

Definition 1.12 (separation pair) *A separation pair of a graph is a pair of vertices $\{x, y\} \subset V$ so that there exist two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ which satisfy the following conditions:*

1. $V = V_1 \cup V_2$, $V_1 \cap V_2 = \{x, y\}$
2. $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$, $|E_1| \geq 2$, $|E_2| \geq 2$.

A separation pair is called *prime separation pair* if at least one of the graphs G_1 and G_2 is either 2-connected or is a subdivision of an edge joining two vertices with degree greater than two.

In the algorithm for testing convex planarity, the *forbidden separation pairs (FSPs)* and the *critical separation pairs* (CSPs) play a crucial role.

Definition 1.13 (Forbidden separation pair) *A prime separation pair is called* forbidden separation pair *(FSP) if it has at least four split components or three split components none of which is a path.*

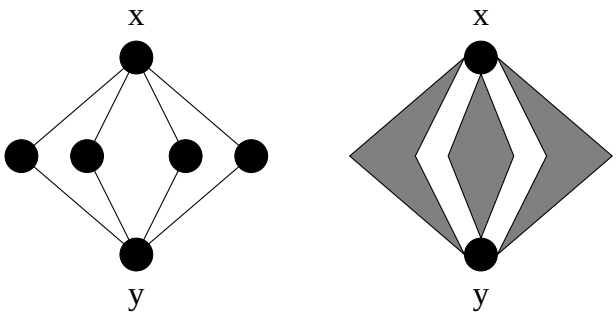


Figure 1.8: Two examples of FSPs $\{x, y\}$. The shaded regions in the drawing on the right are subgraphs.

If a graph has an FSP, there can be no convex representation of the graph. Figure 1.8 shows two examples of FSPs. There is no convex drawing of these graphs.

Definition 1.14 (Critical separation pair) *A prime separation pair is called critical separation pair (CSP) if it has 3 split components of which at least one is a path or if it has two split components of which none is a path.*

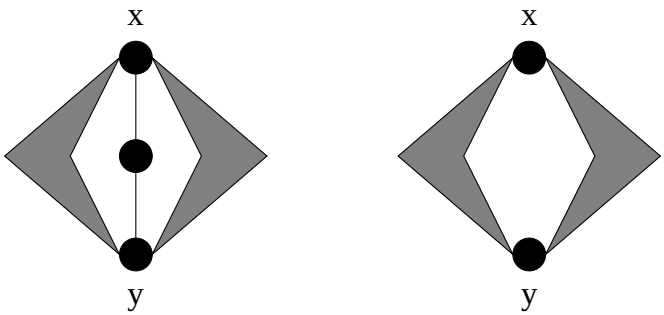


Figure 1.9: Two examples for CSPs $\{x, y\}$. The shaded regions in the drawings are subgraphs.

The algorithm `convex-test` works as follows:

1. Find all separation pairs of G by the linear time algorithm described in Hopcroft and Tarjan (1973) for finding 3-connected components. Determine the set F of FSPs and the set C of CSPs.

2. If $F \neq \emptyset$, then there is no convex representation of G. If both F and C are empty, we can produce a convex representation choosing any face

cycle of G as the cycle S that starts the computation. If there is exactly one pair in C, we choose S as a cycle with the CSP on it, depending on the structure of the split components. If there is more than one pair in the set C, we go to the next step.

3. We transform each CSP with three split components by removing one component that is a path. Then we connect all vertices of all pairs in C to a new vertex v_S and check if the resulting graph G' is planar. If this is not the case, we know that there is no convex representation of G. Otherwise, let Z be any planar representation of G'. Let S be the face cycle that surrounds v_S in Z after deleting all edges incident to v_S. Then we know that there is a convex representation of G if we choose S as the start cycle for the recursive computation of the algorithm `convex-draw`. This is the case because all CSPs belong to S.

1.7 Methods Based on Canonical Orderings

There are several methods for drawing a planar graph that rely on a special ordering of the vertices which is often called *canonical ordering*. The vertices are ordered and successively added in this special order to a data structure that describes a representation of the graph. In some of these algorithms, the vertices are added one by one while in others a set of vertices can be added in one step. Before the execution of one step, the data structure always describes a representation of the subgraph induced by the vertices that have already been added.

The vertex orderings used in all these algorithms and the algorithms themselves have several common properties:

1. The ordering is defined by some embedding of the graph.

2. The ordering of the vertices defines an ordered partition $V_1, V_2, \ldots, V_k$ of the vertices in the vertex set V of the graph. The union of the V_i is V, each V_i has at least one vertex and the V_i are pairwise disjoint.

3. In step i of the algorithm, the vertices in V_i together with the edges that connect them to the vertices in $V_1 \cup V_2 \cup \ldots \cup V_{i-1}$ and the edges between the vertices in V_i are added to the data structure that defines the representation.

4. The set V_1 has at least 2 elements and there is at least one edge in the subgraph induced by V_1 which is on the outer face of every representation D_i.

5. Let S_i be the data structure after inserting the vertices in V_i and let D_i be the corresponding representation. Then D_i is the representation of a 2-connected graph where all the vertices adjacent to vertices in $V_{i+1} \cup \ldots \cup V_k$ are on the outer face of the representation.

The last point is not true for the algorithm proposed by Schnyder (Schnyder, 1990) because in this algorithm, the vertices are inserted inside the triangle given by the three vertices in V_1. This algorithm (described in subsection 1.7.2 is quite different from all others treated in this section, because it computes three barycentric coordinates for each vertex before computing the x- and y-coordinates.

1.7.1 The Algorithm of De Fraysseix, Pach and Pollack

The first algorithm using a canonical ordering for drawing planar graphs with straight edges using polynomial area was described by De Fraysseix, Pach and Pollack in 1990 (de Fraysseix et al., 1990). The algorithm draws a planar triangulated graph on a grid of size $(2n-4) \times (n-2)$ where n is the number of vertices in the graph. The running time of the algorithm is $O(n \log n)$. In the same paper, the authors give a linear time and space algorithm for adding edges to a planar connected graph to produce a planar triangulated graph. The outer face of the representation is always a triangle. This result was later improved by Kant (Kant, 1996), but his algorithm is very similar to the one described in de Fraysseix et al. (1990).

Let $G = (V, E)$ be a triangulated graph with a planar representation D where $(u, v) \in E$ is on the outer face. Let $\pi = (v_1, \ldots, v_k)$ be a numbering of the vertices in V with $v_1 = u$ and $v_2 = v$. We define G_i as the subgraph induced by the vertex set $\{v_1, \ldots, v_i\}$. The face C_i is the outer face of the representation D_i of G_i that we get by removing all representations of vertices and edges from D that do not belong to G_i.

Then π is a canonical ordering if and only if the following conditions hold for all $4 \leq i \leq k$:

- The subgraph G_{i-1} is 2-connected and C_{i-1} contains the edge (v_1, v_2).
- In the representation D, the vertex v_i is in the outer face of G_{i-1} and its neighbors in G_{i-1} form a subinterval of the path C_{i-1} with at least two elements.

Such a canonical ordering exists for any triangulated planar graph and can be computed in linear time by starting with the representation D and successively removing single vertices from the outer face that are not incident to any chords of the outer face. It is easy to show that such a vertex always exists for a triangulated planar graph.

The invariants of the actual drawing algorithm are that after step i (inserting the vertex v_i and the necessary edges), the following conditions hold:

- The vertex v_1 is at position $(0, 0)$ and v_2 at position $(2i - 4, 0)$.
- If the sequence of the vertices on the outer face is $c_1, c_2, \ldots, c_k$ with $c_1 = v_1$ and $c_k = v_2$, then we have $x(c_j) < x(c_{j+1})$ for $1 \leq j < k$.
- The edge (c_j, c_{j+1}) has slope $+1$ or -1 for $1 \leq i < k$.

To describe the idea of the drawing algorithm, we define the *left-vertex* c_l of vertex v_i as the leftmost vertex on C_{i-1} that is adjacent to v_i. With leftmost we mean that the vertex comes first on the path from v_1 via C_{i-1} to v_2 that does not use the edge (v_1, v_2). The *right-vertex* c_r of v_i is defined as the rightmost vertex on C_{i-1} adjacent to v_i. From now on we will refer to the vertex c_{l+1} on C_i as the vertex directly left of c_l on C_i.

When we want to add the vertex v_i, we move the vertices c_{l+1} to c_{r-1} one unit to the right while we move the vertices c_r to c_k two units to the right. We also have to move some inner vertices of the representation to the right to make sure that the representation remains planar. This is achieved by storing for every vertex v on C_i a set of *dependent vertices* that have to be moved in parallel with v. When v vanishes from the outer cycle, we add v to its own list of dependent vertices and make this updated list the set of dependent vertices of the new vertex on the outer cycle.

We place v_i at the intersection of the line with slope $+1$ starting at c_l and the line with slope -1 starting at c_r. Figure 1.10 shows an example for the construction of such a representation. This approach can also be applied to non-triangulated graphs by first adding edges to make the graph triangulated (augmentation), applying the algorithm, and deleting the additional edges in the computed representation.

1.7.2 The Barycentric Algorithm of Schnyder

In the same year, Schnyder described an algorithm for solving the same task in time $O(n)$ using a grid of size $(n - 2) \times (n - 2)$ (Schnyder, 1990). This algorithm computes three coordinates for each vertex in the sequence given by the same canonical ordering used by de Fraysseix, Pach and Pollack. In a second step, it computes the actual grid coordinates for the vertices using the barycentric coordinates.

The vertex positions are defined using a *barycentric representation* of the input graph G.

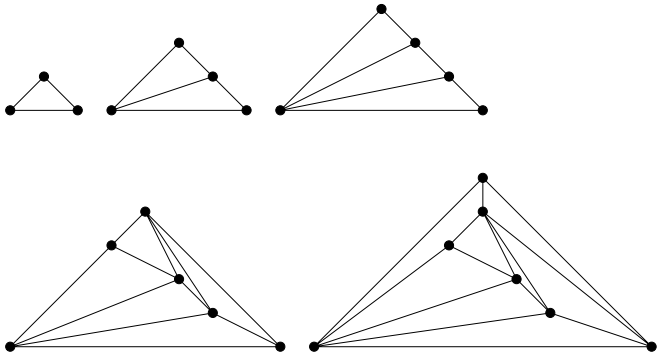


Figure 1.10: An example for the straight-line algorithm of de Fraysseix, Pach and Pollack

Definition 1.15 (barycentric representation) *A barycentric representation of G is an injective function*

$$v \in V \to (v_1, v_2, v_3) \in \mathbb{R}^3$$

satisfying the following conditions:

1. $v_1 + v_2 + v_3 = 1$ *for all vertices* v.
2. *For each edge* $\{u, v\}$ *and each vertex* $w \in \{u, v\}$ *there is some* $k \in \{1, 2, 3\}$ *such that* $u_k < w_k$ *and* $v_k < w_k$.

A barycentric representation of the input graph is computed by first constructing a *normal labeling* of the angles of the faces of the input graph. Since the input graph is triangulated, every face has exactly three angles. The angles of each face are numbered 1, 2 and 3 so that the numbers appear in counterclockwise order around the face and for each interior vertex, the angles around it in counterclockwise order form a nonempty sequence of 1's followed by a nonempty sequence of 2's followed by a nonempty sequence of 3's. Such a labeling can be constructed in linear time.

For each normal labeling, every edge has two different labels on one end while the labels on both sides of the other end are the same. We call the repeated label the label of the edge. Thus, each normal labeling defines a *realizer* of the graph.

Definition 1.16 (realizer) *A realizer of a triangular graph G is a partition of the interior edges of G into three sets T_1, T_2 and T_3 of directed edges so that for each interior vertex v the following conditions are satisfied:*

1. *The vertex v has outdegree 1 in T_1, T_2 and T_3.*

2. *The counterclockwise order of the edges around v is: leaving in T_1, entering in T_3, leaving in T_2, entering in T_1, leaving in T_3, entering in T_2.*

Every normal labeling has the following property: For each number in $\{1, 2, 3\}$ there is exactly one vertex on the outer face where every adjacent angle is labeled i. For each interior vertex, there is exactly one path leaving the vertex where all edges are labeled i for $i \in \{1, 2, 3\}$. This path ends in the vertex of the outer face where all adjacent edges are labeled i. These 3 paths leaving each interior vertex define 3 regions of the graph and the number of faces in each of these regions are the 3 barycentric coordinates of the vertex.

If we have 3 arbitrary noncolinear points α, β and γ in the plane and vertex v has the barycentric coordinates (v_1, v_2, v_3), then drawing every vertex v at position $v_1\alpha + v_2\beta + v_3\gamma$ will result in a planar straight-line embedding of the graph.

1.7.3 The Straight-Line Algorithm of Kant

Kant used the canonical ordering approach to develop several drawing algorithms (Kant, 1996). The first one also produces straight-line representations, but in contrast to the algorithms mentioned before, it guarantees that the inner regions are convex for 3-connected graphs, even if it is not the case that every face of the graph is bounded by 3 edges. This is not necessarily the case for the algorithms mentioned before, because if we want to apply them to non-triangulated graphs, we first have to augment the graph by adding edges to produce a second graph where every face is a triangle, then produce a representation for this graph and finally delete the added edges from the final representation. Thus it might happen that not every inner face of the representation is convex. The algorithm of Kant has a maximum grid-size of $(2n - 4) \times (n - 2)$ and runs in $O(n)$ time. Chrobak and Kant later improved this algorithm so that it only uses an area of $(n - 2) \times (n - 2)$ (Chrobak and Kant, 1993).

Since this algorithm is an improved version of the algorithm of de Fraysseix, Pach and Pollack (de Fraysseix et al., 1990), we will only give an overview of the differences. The algorithm of Kant can also cope with 3-connected graphs that are not triangulated. This is achieved by defining the canonical ordering not as an ordering of the vertices but rather as an ordered partition of the vertices. Let $G = (V, E)$ be a 3-connected graph with a planar representation D where $v_1 \in V$ is on the outer face. Let $\pi = (V_1, \ldots, V_k)$ be a partition of V and G_i the subgraph of G induced by $V_1 \cup V_2 \cup \ldots \cup V_i$. The face C_i is the outer face of the representation D_i of G_i that we get by removing all representations of vertices and edges from D that do not belong to G_i.

Then π is a canonical ordering if and only if the following conditions hold:

- $V_1 = \{v_1, v_2\}$, v_1 and v_2 both lie on the outer face of D and $(v_1, v_2) \in E$.
- $V_k = v_n$ and v_n lies on the outer face of D with $(v_1, v_n) \in E$ and $v_n \neq v_2$.
- Each C_i for $k > 1$ is a cycle containing (v_1, v_2).
- Each G_i is 2-connected and internally 3-connected (removing any two inner vertices will not disconnect the graph).
- For each $i \in \{2, \ldots, k-1\}$ one of the following conditions holds:
  1. V_i is a single vertex z belonging to C_i and having at least one neighbor in $G - G_i$.
  2. The vertices in V_i form a *chain* (a path where all inner vertices have degree 2) $(z_1, \ldots, z_l)$ on C_i where each z_j has at least one neighbor in $G - G_i$. The vertices z_1 and z_l each have exactly one neighbor in C_{i-1}, and these are the only neighbors of the vertices in V_i.

This canonical ordering can be computed in linear time by starting with the representation D and successively removing chains or single vertices from the outer face so that the resulting graph G' is 2-connected. To do this in linear time, we have to store and update for each face the number of its vertices and edges on the outer face and for each vertex the number of adjacent faces having a separation pair.

To compute the actual representation, the canonical ordering is first transformed into a *leftmost canonical ordering* which can be computed in linear time from a canonical ordering and is necessary for achieving linear running time. The invariants of the drawing algorithm after step i (inserting the vertices of the set V_i and the necessary edges) are:

- v_1 is at position $(0, 0)$ and v_2 at position $(2i - 4, 0)$.
- If the sequence of the vertices on the outer face is $c_1, c_2, \ldots, c_k$ with $c_1 = v_1$ and $c_k = v_2$, then we have $x(c_j) < x(c_{j+1})$ for $1 \leq j < k$.

The only difference in the actual drawing algorithm compared to the algorithm of de Fraysseix, Pach and Pollack is that we can now insert several vertices at once. These vertices form a chain and we give them the same y-coordinate. Figure 1.11 shows an example for the construction of such a representation.

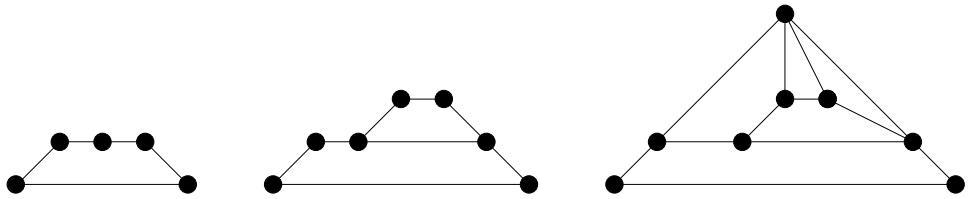


Figure 1.11: An example for the straight-line algorithm of Kant

1.7.4 The Orthogonal Algorithms of Kant

In the same paper (Kant, 1996), Kant also gives two algorithms for producing *orthogonal representations* of planar graphs. In an orthogonal representation, all edges consist only of horizontal and vertical segments. If every vertex is drawn as a point, such a representation can only be used for planar graphs in which every vertex has at most degree 4 (*4-planar graphs*). Since orthogonal drawing algorithms are explicitly treated in Chapter **??**, we will only give a short overview of the two algorithms. The first algorithm draws 3-connected 4-planar graphs on an $n \times n$ grid with at most $\lceil \frac{3}{2}n \rceil + 4$ bends so that each edge has at most two bends. The second algorithm produces an orthogonal representation for planar graphs with maximum degree 3 having at most $\lfloor \frac{n}{2} \rfloor + 1$ bends on a grid of size $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$. The running time of both algorithms is linear.

The algorithm for producing orthogonal representations of 3-connected 4-planar graphs given in Kant (1996) also uses the canonical ordering. Since the edges only consist of vertical and horizontal segments, there are exactly 4 directions from which an edge can attach to a vertex v. They are called *up(v)*, *down(v)*, *left(v)* and *right(v)*. One of these directions is called *free* if we have not yet attached an edge to it. The idea of the algorithm is to add each vertex v in the canonical ordering to the subgraph that is already placed so that *down(v)* is not free and *up(v)* is free. The algorithm works in two phases. In the first phase, we assign the 4 directions of each vertex to the incident edges and give the vertices the y-coordinates. We also store for each vertex and bend a pointer to its column. During the algorithm, we may have to add new columns. In the second phase, we assign x-coordinates to the columns and thus indirectly to the vertices and bends of the representation.

The algorithm in Kant (1996) for drawing planar graphs with maximum degree 3 is based on an algorithm for 3-connected graphs with maximum degree 3. This algorithm is similar to the algorithm of the last paragraph, but we can place all vertices of the same partition of the canonical ordering on the same y-coordinate. The algorithm is generalized for working on 2-connected 3-planar graphs using SPQR-trees. We recursively use the algorithm for drawing 3-connected 3-planar graphs and then merge the representations into a

representation for the whole graph. This method is again generalized to connected 3-planar graphs by drawing every 2-connected component so that the cut vertex is in the upper-left corner and then merging the representations into a representation of the whole graph without introducing new bends.

1.7.5 The Mixed Model

Kant also introduces a new method for drawing 3-connected planar graphs which he calls the *Mixed Model*. In this model, each edge is a poly-line which may have at most three bends. Each edge consists of at most four parts. The parts connected to the vertices may be diagonal, while the two middle parts of each edges are vertical and horizontal. The principle of the algorithm is to define a set of points around each vertex where the orthogonal edges coming from other vertices connect. These points define the boundary of the *bounding box* of the vertex. The points are then connected by straight lines to the vertex itself. Each edge consists of a straight line segment between the start vertex and a point on the boundary of the bounding box, an orthogonal part with at most one bend from the bounding box of the start-vertex to the bounding box of the target-vertex and another straight part from the boundary of the bounding box of the target-vertex to the target-vertex itself.

The grid size for this algorithm is $(2n-6) \times (3n-9)$ and the number of bends is at most $5n-15$. An important property of the algorithm is that it guarantees that the angle between two edges emanating from the same vertex is larger than $2/d$ radians where d is the degree of the vertex. The minimum angle of two edges emanating from the same vertex in a representation is called the *angular resolution* of the representation. Having a large angular resolution improves the readability of a drawing.

Gutwenger and Mutzel have improved Kant's algorithm for the Mixed Model to achieve a grid size of $(2n-5) \times (\frac{3}{2}n - \frac{7}{2})$ (Gutwenger and Mutzel, 1998). They also have improved the angular resolution for graphs which are not 3-connected. Since Kant's algorithm only works for 3-connected graphs, graphs that are not 3-connected have to be augmented by adding additional edges before the algorithm is applied and afterwards the additional edges have to be deleted from the representation. This can lead to an angular resolution of $\frac{4}{3d+7}$ where d is the maximum degree in the original graph. Since the algorithm in Gutwenger and Mutzel (1998) can be applied directly to 2-connected graphs, an angular resolution of $2/d$ can be guaranteed for any planar graph. The running time for both algorithms is linear.

The algorithm for drawing graph G works in three phases:

1. If the graph is not 2-connected, edges are added to produce a planar 2-connected graph G'

2. A suitable canonical ordering for G' is computed
3. The ordering produced in the last step is used to draw the original graph G.

For each vertex, we define a set of *inpoints* and *outpoints*. The inpoints are the points where the edges from vertices that have already been placed arrive and the outpoints are the points where the edges to vertices that have not already been placed leave. The inpoints and outpoints of each vertex are located on the boundary of the roughly diamond shaped bounding box and will be placed on grid coordinates. Figure 1.12 shows two examples of bounding boxes.

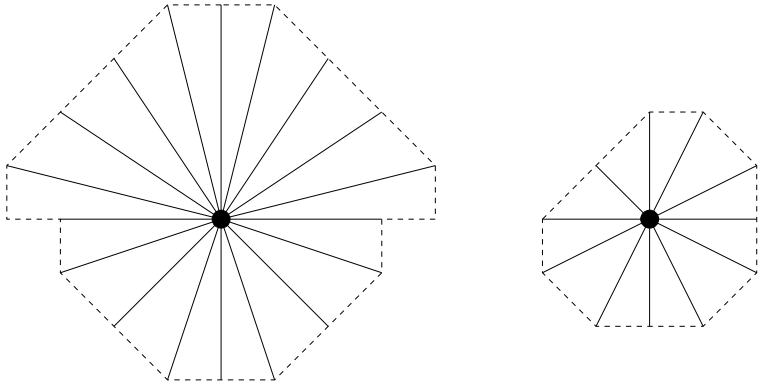


Figure 1.12: Two examples of bounding boxes

The point straight down from a vertex as well as the two points straight to the right and to the left are inpoints, while the point straight above the vertex is always an outpoint. Thus, a vertex with an indegree not greater than 3 and an outdegree of at most 1 will have no adjacent diagonal edges. These inpoints together with the edges that connect them to the vertex form a cross. We call the four sectors defined by this cross NW, NE, SE and SW, like the points of a compass.

If there are more than three incoming edges, we distribute the remaining inpoints evenly among the sectors SE and SW. If there are more than one outgoing edges, they are distributed evenly between the sectors NE and NW. If the remaining number of edges is not even, we get an asymmetric configuration like in the righthand bounding box of Figure 1.12.

When a vertex is placed, we always have to avoid overlapping bounding boxes except if we can identify the outpoint of an adjacent vertex with the vertex we want to place. If the set V_i of vertices in the canonical order we want to add in step i has only one element v, we place this vertex directly above

the adjacent vertex which is connected by the inedge going straight down. We choose the y-coordinate so that the minimum vertical distance between the bounding box of an adjacent vertex and the bounding box of v is 1. We may have to shift the adjacent vertices already placed and their dependent sets to the right to make room for the edges. If V_i has more than one element, all the vertices in the set will get the same y-coordinate. Figure 1.13 shows an example of a drawing produced with Kant's original algorithm.

Figure 1.14 shows two drawings computed using the algorithm of Gutwenger and Mutzel (Gutwenger and Mutzel, 1998).

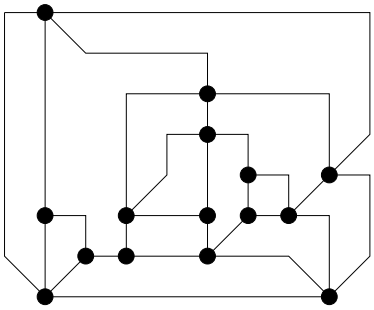


Figure 1.13: A drawing produced by the Mixed Model algorithm of Kant

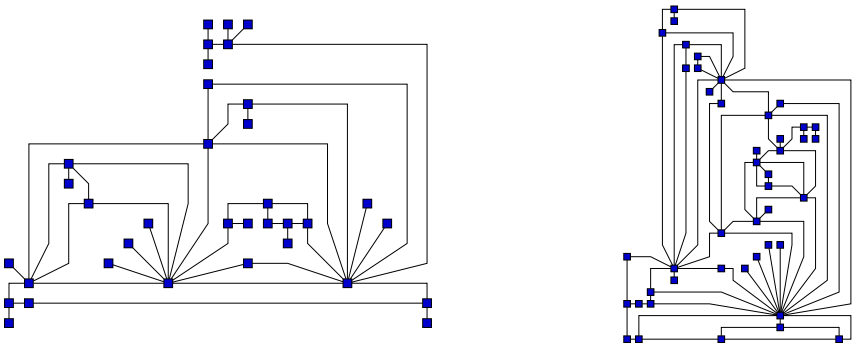


Figure 1.14: Example drawings produced by the algorithm of Gutwenger and Mutzel

Bibliography

Auslander, L. and Parter, S. V. (1961). On embedding graphs in the sphere. *Journal of Mathematics and Mechanics*, 10(3):517–523.

Booth, K. S. and Lueker, G. S. (1976). Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379.

Cai, J., Han, X., and Tarjan, R. E. (1993). An $O(m \log n)$-time algorithm for the maximal planar subgraph problem. *SIAM Journal on Computing*, 22:1142–1162.

Chrobak, M. and Kant, G. (1993). Convex grid drawings of 3-connected planar graphs. Technical Report RUU-CS-93-45, Dept. of Comp. Sci., Utrecht University.

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

Di Battista, G. and Tamassia, R. (1988). Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2-3):175–198.

Di Battista, G. and Tamassia, R. (1989). Incremental planarity testing. In *Proceedings of the 30th Symposium on the Foundations of Computer Science (FOCS'89)*, pages 436–441.

Di Battista, G. and Tamassia, R. (1990). On-line graph algorithms with SPQR-trees. In *Proceedings of the 17th International Colloqium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 598–611.

Di Battista, G. and Tamassia, R. (1996). On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997.

Djidjev, H. N. (1995). A linear algorithm for the maximal planar subgraph problem. In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95).* Springer Lecture Notes in Computer Science 955, pages 369–380.

Even, S. (1979). *Graph Algorithms.* Pitman.

Faria, L., De Figueiredo, C. M. H., and Mendonca, C. F. X. (1998). Splitting number is NP-complete. *Lecture Notes in Computer Science*, 1517:285–??

Fialko, S. and Mutzel, P. (1998). 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.

Foulds, L. R., Gibbons, P. B., and Giffin, J. W. (1985). Facilities layout adjacency determination: An experimental comparison of three graph theoretic heuristics. *Operations Research.*

Foulds, L. R. and Robinson, D. F. (1978). Graph theoretic heuristics for the plant layout problem. *International Journal of Production Research*, 16:27–37.

Garey, M. R. and Johnson, D. S. (1983). Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316.

Garg, A. and Tamassia, R. (1994). On the computational complexity of upward and rectilinear planarity testing. In *Proceedings of the DIMACS International Workshop on Graph Drawing (GD'94).* Springer Lecture Notes in Computer Science 894, pages 286–297.

Goldstein, A. J. (1963). An efficient and constructive algorithm for testing wether a graph can be embedded in a plane. In *Graph and Combinatorics Conference, Contract No. NONR 1858-(21).*

Gutwenger, C. and Mutzel, P. (1998). Planar polyline drawings with good angular resolution. In *Proceedings of the 6th International Symposium on Graph Drawing (GD'98).* Springer Lecture Notes in Computer Science 1547, pages 167–182.

Hopcroft, J. and Tarjan, R. E. (1974). Efficient planarity testing. *Journal of the ACM*, 21:549–568.

Hopcroft, J. E. and Tarjan, R. E. (1973). Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158.

Hsu, W.-L. (1995). A linear time algorithm for finding maximal planar subgraphs. In *Proceedings of the 6th International Symposium on Algorithms and Computation (ISAAC'95).* Springer Lecture Notes in Computer Science 1004, pages 352–361.

Jünger, M. and Mutzel, P. (1996). Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59.

Kant, G. (1994). personal communication.

Kant, G. (1996). Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32.

Kant, G. and Bodlaender, H. L. (1991). Planar graph augmentation problems. In *Proceedings of the 2nd Workshop on Algorithms and Data Structures (WADS'91)*, volume 519 of *Lecture Notes in Computer Science*, pages 286–298.

Lempel, A., Even, S., and Cederbaum, I. (1967). An algorithm for planarity testing of graphs. In *Theory of Graphs, International Symposium, Rome*, pages 215–232.

Leung, J. (1992). A new graph theoretic heuristic for facility layout. *Management Science*.

Lewis, J. M. and Yannakakis, M. (1980). The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230.

Liebers, A. (1996). Methods for planarizing graphs - A survey and annotated bibliography. Technical Report Konstanzer Schriften in Mathematik und Informatik Nr. 12, Fakultät für Mathematik und Informatik, Universität Konstanz. ISSN 1430-3558.

Liu, P. C. and Geldmacher, R. C. (1977). On the deletion of nonplanar edges of a graph. In *Proceedings of the 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 727–738.

Mehlhorn, K. (1984). *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness EATCS Monographs on Theoretical Computer Science*. Springer.

Mehlhorn, K. and Näher, S. (1999). *The* LEDA *Platform of Combinatorial and Geometric Computing*. Cambridge University Press. Project home page at `http://www.mpi-sb.mpg.de/LEDA/`.

Mutzel, P. (1994). *The Maximum Planar Subgraph Problem*. PhD thesis, Universität zu Köln, Köln, Germany.

Mutzel, P. (1995). A polyhedral approach to planar augmentation and related problems. In *Proceedings of the 3rd Annual European Symposium on Algorithms (ESA'95)*, volume 979 of *Lecture Notes in Computer Science*, pages 494–507.

Nishizeki, T. and Chiba, N. (1988). *Planar Graphs : Theory and Algorithms*, volume 140/32 of *North-Holland mathematics studies*. North-Holland, Dordrecht.

Poutré, J. A. L. (1994). Alpha-algorithms for incremental planarity testing. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computation (STOC'94)*, pages 706–715.

Schnyder, W. (1990). Embedding planar graphs on the grid. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA'90)*, pages 138–148.

Thomassen, C. (1980). Planarity and duality of finite and infinite planar graphs. *Journal of Combinatorial Theory, Series B*, 29:244–271.

Tutte, W. T. (1960). Convex representations of graphs. *Proceedings of the London Mathematical Society, Third Series*, 10:304–320.

Tutte, W. T. (1963). How to draw a graph. *Proceedings of the London Mathematical Society, Third Series*, 13:743–768.

Watanabe, T., Ae, T., and Nakamura, A. (1983). On the NP-hardness of edge-deletion and -contraction problems. *Discrete Applied Mathematics*, 6:63–78.

Yannakakis, M. (1978). Node- and edge-deletion NP-complete problems. In *Proceedings 10th Annual ACM Symposium on the Theory of Computing (STOC'78)*, pages 253–264.