Acta Informatica 22, 187–201 (1985)

Acta Informatica

Drawing Plane Graphs Nicely

Norishige Chiba, Kazunori Onoguchi, and Takao Nishizeki

Department of Electrical Communications, Faculty of Engineering,
Tohoku University, Sendai 980, Japan

Summary. This paper presents two efficient algorithms for drawing plane graphs nicely. Both draw all edges of a graph as straight line segments without crossing lines. The first draws a plane graph "convex" if possible, that is, in a way that every inner face and the complement of the outer face are convex polygons. The second, using the first, produces a pleasing drawing of a given plane graph that satisfies the following property as far as possible: the complements of 3-connected components, together with inner faces and the complement of the outer face, are convex polygons. The running time and storage space of both algorithms are linear in the number of vertices of the graph.

1. Introduction

The problem of drawing a planar graph often arises in applications, including Design Automation of VLSI circuits. In this paper we are not interested in a specific practical application, but in producing a pleasing drawing of a given planar graph. We assume throughout this paper that an embedding of a planar graph is given, that is, a *plane* graph is given. Linear time algorithms are known for embedding planar graphs without edge crossing [1, 5]. Restricting given graphs to trees, some recent papers have studied the problem of producing well-shaped drawings [7, 8, 12, 13]. Obviously there are no absolute criteria that accurately capture our intuitive notion of nice drawings of plane graphs. However it seems that the following are desirable properties of pleasing drawings:

(a) all the edges are drawn by straight line segments without crossing lines;

(b) a facial cycle (i.e. a boundary of a face) is drawn as a convex polygon;

(c) an outer facial cycle of a 3-connected component is drawn as a convex polygon.

The justification for property (a) above is rather self-evident. For example, compare the two drawings of the same graph depicted in Fig. 1a and b. One

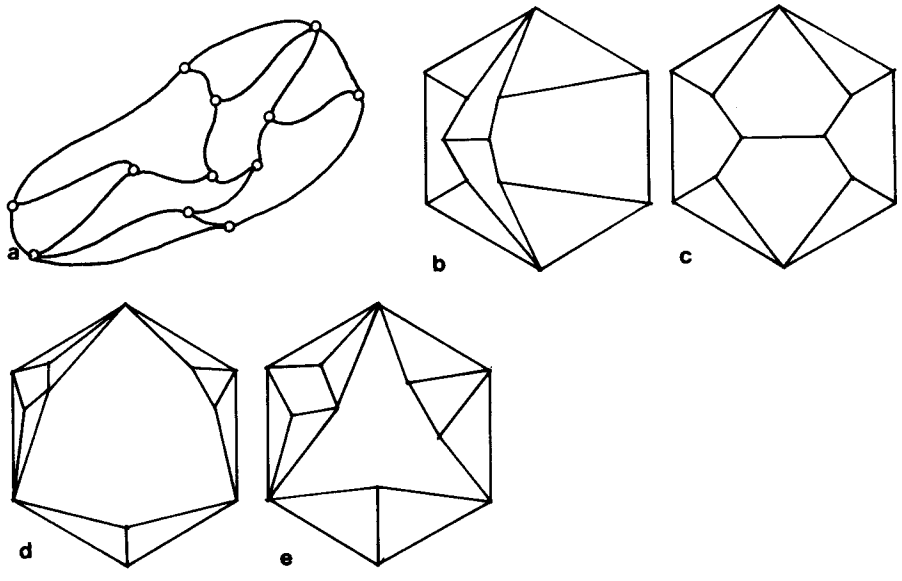

Fig. 1. **a** A planar graph drawn by curves; **b** a drawing by straight lines; **c** a convex drawing; **d** a convex drawing of a 2-connected graph; **e** our drawing

may feel that the drawing in Fig. 1b by straight line segments is more pleasing than the drawing in Fig. 1a by curves. It is known that every planar graph can be drawn by straight lines without crossing lines [3, 9]. Plane graphs corresponding to three-dimensional convex polytopes are most typical examples whose edges must be drawn as straight line segments.

We next consider the justification for property (b). See a "convex" drawing of the same graph in Fig. 1c, in which every facial cycle is a convex polygon. One may feel that the convex drawing is more pleasing than the drawing in Fig. 1b. Thus one can approve of property (b). It should be noted that not every plane graph has a convex drawing although every 3-connected plane graph has a convex one.

Although a convex drawing seems pleasing, this is not always the case. Compare the two drawings of a graph depicted in Fig. 1d and e. The drawing in Fig. 1d is convex, while the drawing in Fig. 1e is not. One may feel that the latter drawing, which we intend to produce, is more pleasing than the former convex one. Such an intuitive feeling seems to come from the following difference. In Fig. 1d, drawing the entire graph convex forces 3-connected components to be drawn in rather ill-shapes. To the contrast, each 3-connected component is drawn convex in Fig. 1e; this makes the 3-connected components easily recognizable from the drawing. Thus one can approve of property (c).

The properties (b) and (c) are competing, that is, both are not always satisfied simultaneously. Therefore we first let a drawing satisfy (c) and then (b) as far as possible.

We may assume that a given plane graph is 2-connected. If a given plane graph is not 2-connected, then the entire drawing can be constructed by one of the following two alternative methods: (1) draw each 2-connected component separately, and then combine them into an entire drawing; (2) first augment the graph into a 2-connected plane graph, draw the graph, and then delete the added edges from the drawing.

This paper is organized as follows. In Sect. 2 we present some preliminary definitions. In Sect. 3 we give a linear-time algorithm for drawing a planar graph convex. In Sect. 4 we present an algorithm which, given a 2-connected plane graph, produces a drawing satisfying properties (a)–(c) as far as possible. The running time and storage space are both linear in the number of vertices of a graph. In Sect. 5 we give some computated examples produced by an implementation of the presented material.

2. Preliminaries

In this section, we first define some terms, and then give illustrative examples for the terms.

Let $G=(V, E)$ be a simple *graph*, that is, having no loops or multiple edges, which has vertex set V and edge set A. The vertex set of a graph G is often denoted by $V(G)$. A graph $G'=(V', E')$ is called a *subgraph* of G if $V' \subset V$ and $E' \subset E$. $G-V'$ denotes the maximal subgraph of G with vertex set $V-V'$. A graph G is *connected* if every two vertices of G are joined by a path. The *connected components* of G are its maximal connected subgraphs. A *cut vertex* of G is a vertex whose removal increases the number of connected components. A graph G having no cut vertex is called *2-connected*. The *blocks* of a graph are its maximal 2-connected subgraphs. A graph is *planar* if it is embeddable in the plane without edge crossing. A *plane* graph G is a planar graph which is embedded in the plane. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called the *outer face* of G. We assume that a 2-connected *plane* graph G is given. Since G is 2-connected, the boundary of a face is a cycle, which is called a *facial cycle*. Especially the boundary of the outer face is called an *outer cycle*. A path joining vertices x and y is called an *$x-y$ path*. A drawing of a plane graph is *convex* if all edges are drawn by straight line segments without any crossing in such a way that all the facial cycles are convex polygons. Since all the edges are drawn by straight lines, a convex drawing of a plane graph is uniquely determined only by the positions of vertices. Let S be the outer cycle of G, and let S^* be a convex polygonal drawing (a convex polygon in short) of S. S^* is *extendible* if there exists a convex drawing of G having S^* as the outer cycle.

We borrow some of the following definitions from [4]. A pair $\{x, y\}$ of vertices of a 2-connected graph $G=(V, E)$ is a *separation pair* if there exist two subgraphs $G'_1=(V_1, E'_1)$ and $G'_2=(V_2, E'_2)$ satisfying the following conditions:

(a) $V=V_1 \cup V_2$, $V_1 \cap V_2=\{x, y\}$;

(b) $E=E'_1 \cup E'_2$, $E'_1 \cap E'_2=\emptyset$, $|E'_1| \geqq 2$, $|E'_2| \geqq 2$.

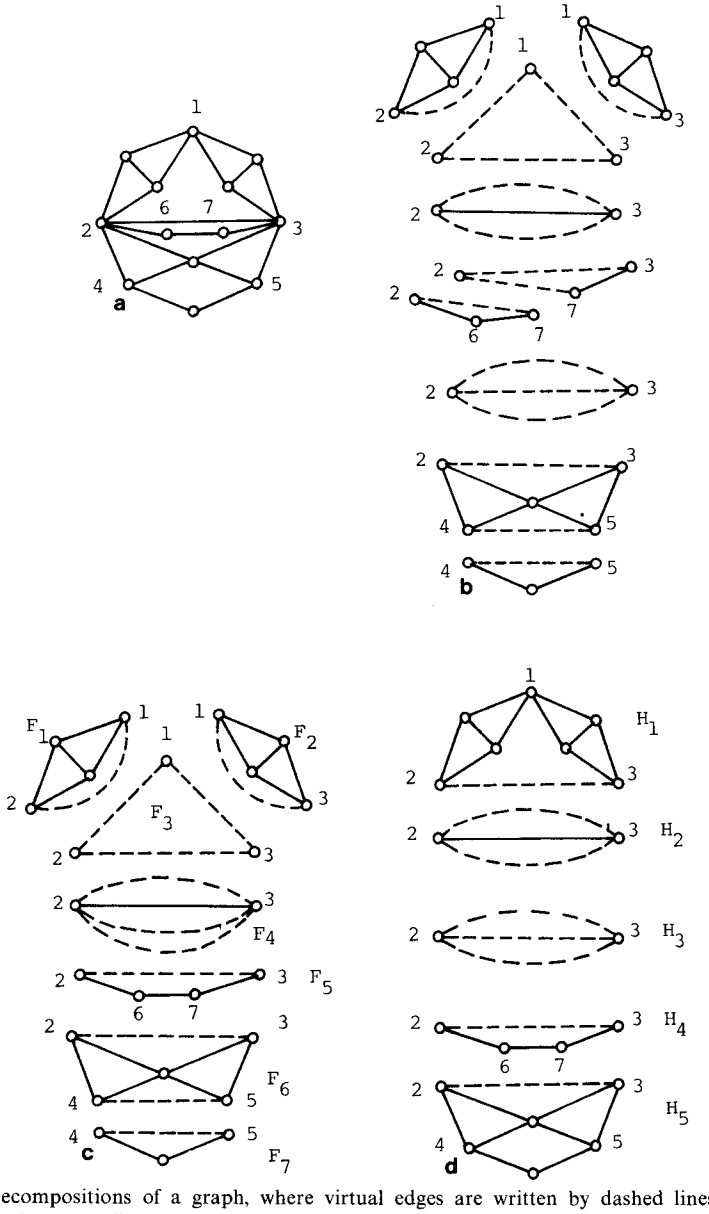

Fig. 2a–d. Decompositions of a graph, where virtual edges are written by dashed lines: **a** a 2-connected graph G; **b** split components of G; **c** 3-connected components of G; and **d** $\{2, 3\}$-split components of C with one exception H_3

A 2-connected graph G is said to be *3-connected* if G has no separation pair. For the separation pair $\{x, y\}$, $G_1=(V_1, E_1'+(x, y))$ and $G_2=(V_2, E_2'+(x, y))$ are called *split graphs* of G. Although G is simple, G_1 and G_2 are not always simple. The new edges (x, y) added to G_1 and G_2 are called *virtual edges*. Dividing a graph G into two split graphs G_1 and G_2 is called *splitting*.

Reassembling the two split graphs G_1 and G_2 into G is called *merging*. Merging is the inverse of splitting. Suppose a graph G is split, the split graphs are split, and so on, until no more splits are possible (each remaining graph is 3-connected). The graphs constructed in this way are called *split components* of G. The split components of a graph G are of three types: *triple bonds* (i.e. a set of three multiple edges), *triangles* (i.e. a cycle consisting of three edges), and 3-connected graphs. The *bonds* and the *rings* are obtained from triple bonds and triangles, respectively, by merging as far as possible. We call these bonds, rings, and 3-connected graphs the *3-connected components* of G. (To avoid confusion we use "ring", instead of "polygon" used in [4].) The split components of a graph G are not necessarily unique, but the 3-connected components of G are unique.

We illustrate the decompositions of a 2-connected graph in Fig. 2. The graph G depicted in Fig. 2a has six separation pairs $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, $\{2, 7\}$, $\{3, 6\}$, and $\{4, 5\}$. The graph G is decomposed into nine split components as shown in Fig. 2b, and into seven 3-connected components $F_1, F_2, \ldots, F_7$ as shown in Fig. 2c. The components F_1, F_2, and F_6 are 3-connected graphs; F_3, F_5, and F_7 are rings; and F_4 is a bond.

We now introduce new terms. Suppose that $\{x, y\}$ is a separation pair of a graph G and that G is split at $\{x, y\}$, the split graphs are split at $\{x, y\}$, and so on, until no more split are possible at $\{x, y\}$ (the remaining graphs are not necessarily 3-connected). A graph constructed in this way is called an $\{x, y\}$-*split component* of G if it has at least one real (i.e. non-virtual) edge. In Fig. 2d, the components H_1, H_2, H_4, and H_5 are the $\{2, 3\}$-split components.

3. Convex Drawing Algorithm

In this section we present a linear-time algorithm for producing a convex drawing of a given plane graph (if it exists). Tutte gave a "barycentric mapping" method for producing a convex drawing, which solves a system of $O(n)$ linear equations [11]. The system of equations can be solved in $O(n^3)$ time and $O(n^2)$ space using the ordinary Gaussian elimination method, or in $O(n^{1.5})$ time and $O(n \log n)$ space using the sparse Gaussian elimination method [6]. Thus the barycentric mapping method leads to an $O(n^{1.5})$ time convex drawing algorithm. He also established a necessary and sufficient condition for a plane graph to have a convex drawing [10]. The following lemma is a strong version of his result obtained by Thomassen [9]. Roughly speaking, the lemma states that if a plane graph G has a convex drawing, then almost all the separation pairs must lie on the outer cycle S of G. More precise intuitive interpretation of the lemma is presented in [2].

Lemma 1 (Thomassen [9]). *Let $G=(V, E)$ be a 2-connected plane graph with the outer cycle S, and let S^* be a convex k-gon of S. Let $P_1, P_2, \ldots, P_k$ be the paths in S, each corresponding to a side of S^*. (It should be noted that not every vertex of the cycle S is an apex,* i.e., *a geometrical vertex of the polygon S^*.) Then S^* is extendible if and only if Condition I below holds.*

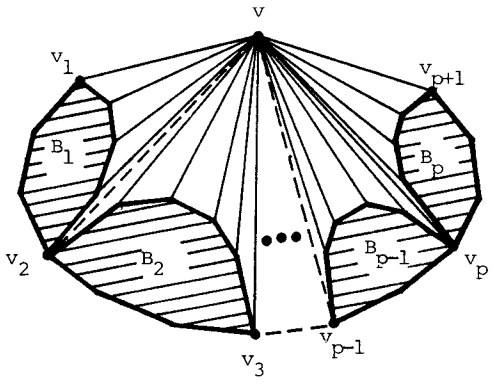

Fig. 3. Reduction of the convex drawing of G into subproblems

Condition I (a) for each vertex v of $G-V(S)$ having degree at least three in G, there exist three paths disjoint except v, each joining v and a vertex of S;

(b) $G-V(S)$ has no connected component C such that all the vertices on S adjacent to vertices in C lie on a single path P_i; and an edge joins two vertices of P_i only if it is in P_i; and

(c) any cycle of G which has no edge in common with S has at least three vertices of degree $\geqq 3$ in G.

Suppose that a 2-connected plane graph G is given together with an extendible convex polygon S^* of the outer cycle S. (If there is an extendible convex drawing of S, then one can easily be obtained by taking the vertices of S on a circle.) Our algorithm extends S^* into a convex drawing of G in linear time. The algorithm is based on Thomassen's proof of Lemma 1. The outline is as follows. We reduce the convex drawing of G to those of several subgraphs of G: delete from G an arbitrary apex v of S^* together with the edges incident to v; divide the resulting graph $G'=G-v$ into the blocks $B_1, B_2, \ldots, B_p$, $p \geqq 1$ (see Fig. 3); determine a convex polygon S_i^* of the outer cycle S_i of each B_i so that B_i with S_i^* satisfies Condition I, and recursively apply the algorithm to each B_i with S_i^* to determine the positions of vertices not in S_i. The details are as follows.

procedure CONVEX-DRAW (G, S, S^*);
begin
{Let G be a given 2-connected plane graph, let S be the outer cycle, and let S^* be an extendible convex polygon of S. For simplicity we reduce the drawing of G to that of a graph G' which has no vertex of degree two not on S.}
for each vertex v of degree two not on S
do replace v together with the two edges incident to v by a single edge joining the vertices adjacent to v;
let G' be the resulting graph;
EXTEND (G', S, S^*); {extend S^* into a convex drawing of G'.}
for each deleted vertex of degree 2
do determine a position for the vertex on the straight line segment joining the two vertices adjacent to it
end.

procedure EXTEND (G, S, S^*);
{This procedure extends a convex polygon S^* of the outer cycle S of a plane graph G into a convex drawing of G, where G has no vertex of degree 2 not on S}
begin
if G has at least four vertices {otherwise, a convex drawing of G has been obtained}
then
begin
select an arbitrary apex v of S^*, and let $G' := G - v$;
divide the plane graph G' into the blocks B_i ($1 \leqq i \leqq p$);
let v_1 and v_{p+1} be the two vertices on S adjacent to v, and let v_i, $2 \leqq i \leqq p$, be the cut vertices of G' such that $v_1 \in V(B_1)$, $v_{p+1} \in V(B_p)$ and $v_i = V(B_{i-1}) \cap V(B_i)$; {see Fig. 3.}
{Every v_i, $1 \leqq i \leqq p+1$, is necessarily on S since the extendible S^* with G satisfies Condition I and G has no vertex of degree two not on S.}
for each block B_i
do begin
{We determine a convex polygon S_i^* of the outer cycle S_i of B_i below. Since the positions of the vertices in $V(S_i) \cap V(S)$ have already been determined on S^*, we should determine the positions of the vertices in $V(S_i) - V(S)$.}
place the vertices in $V(S_i) - V(S)$ in the interior of the triangle $v \cdot v_i \cdot v_{i+1}$ in such a way that the vertices adjacent to v are apices of a convex polygon S_i^* and the others are on the straight line segments of S_i^*;
EXTEND (B_i, S_i, S_i^*);
{extend S_i^* to a convex drawing of B_i.}
end
end
end.

We have the following result on the algorithm.

Theorem 1. *Let G be a 2-connected plane graph, let S be the outer cycle S, and let S^* be an extendible convex polygon of S. Then the algorithm Convex-Draw extends S^* into a convex drawing of G, and uses linear time and space.*

Proof. Since G satisfies Condition I, the graph G' with no vertices of degree two not on S also satisfies Condition I and is simple, that is, no multiple edges appear in G'. In the produced drawing, the inner facial cycles containing the apex v are all triangles, and hence convex polygons. (See Fig. 3.) Therefore, in order to prove inductively the correctness of the algorithm, one should show that every block B_i with S_i^* satisfies Condition I and B_i has no vertex of degree two not on S_i. We omit the proof since it is similar to that of Theorem 5.1 in [9]. Thus we shall establish the claims on time and space.

As a data structure to represent a plane graph $G=(V, E)$, we use doubly linked adjacency lists, in each of which the edges adjacent to a vertex are stored in a list in the order of the plane embedding, clockwise around the vertex. The two copies of each edge (v, w) in the adjacency lists of v and w are linked together so that one can be accessed directly from the other. Given an edge e, one can directly access the edge clockwise next to e around an end of e. Clearly such a data structure uses linear space.

Evidently deletions and insertions of all vertices having degree two can be executed in $O(n)$ time. Therefore we shall prove that EXTEND spends at most linear time. Let P be the v_1-v_{p+1} path of $G'=G-v$ which newly appears on the outer cycle S' of G'. While traversing P, one can easily (1) find the cut vertices v_i, $2\leqq i\leqq p$, which are also on S, (2) obtain the outer cycle S_i of B_i as the union of the traversed v_i-v_{i+1} path on P and the $v_{i+1}-v_i$ path on S, and (3) decides the positions of the vertices of S_i as specified in EXTEND. Thus the time required by the procedure EXTEND, exclusive of recursive calls to itself, is proportional to the number of the traversed edges in P. Hence the time is proportional to the number of the edges newly appeared on the outer cycles. Since every edge appears on an outer cycle at most once, the number of edges traversed during an execution of EXTEND is at most $|E|$ in total. Thus EXTEND runs in linear time. Q.E.D.

4. Drawing Algorithm

In this section we present an algorithm which, given an arbitrary 2-connected plane graph G, produces a pleasing drawing satisfying properties (a)–(c) in Sect. 1 as far as possible. It should be noted that an extendible convex polygon S^* of the outer cycle is not assumed in the algorithm of this section unlike that in the preceding section. The algorithm proceeds as follows.

(1) decompose G into 3-connected components;

(2) by merging some components, construct a subgraph, called a *frame*;

(3) produce a convex drawing of the frame, which will work as the core of an entire drawing of G;

(4) draw all the components convex, one by one, or two by two;

(5) embed the drawings of components into the drawing of the frame.

We first show how to choose the frame of G.

Lemma 2. *Let G be a 2-connected plane graph, and let S be the outer cycle. Split G at a separation pair if exactly one of the resulting split graphs contains no edge of S, and then abandon the split graph. Repeat this operation for all separation pairs. Let G' be the resulting graph for which any splitting produces split graphs both having edges of S. Then G' contains S and has a convex drawing.*

Proof. Immediately follows from Lemma 1. Q.E.D.

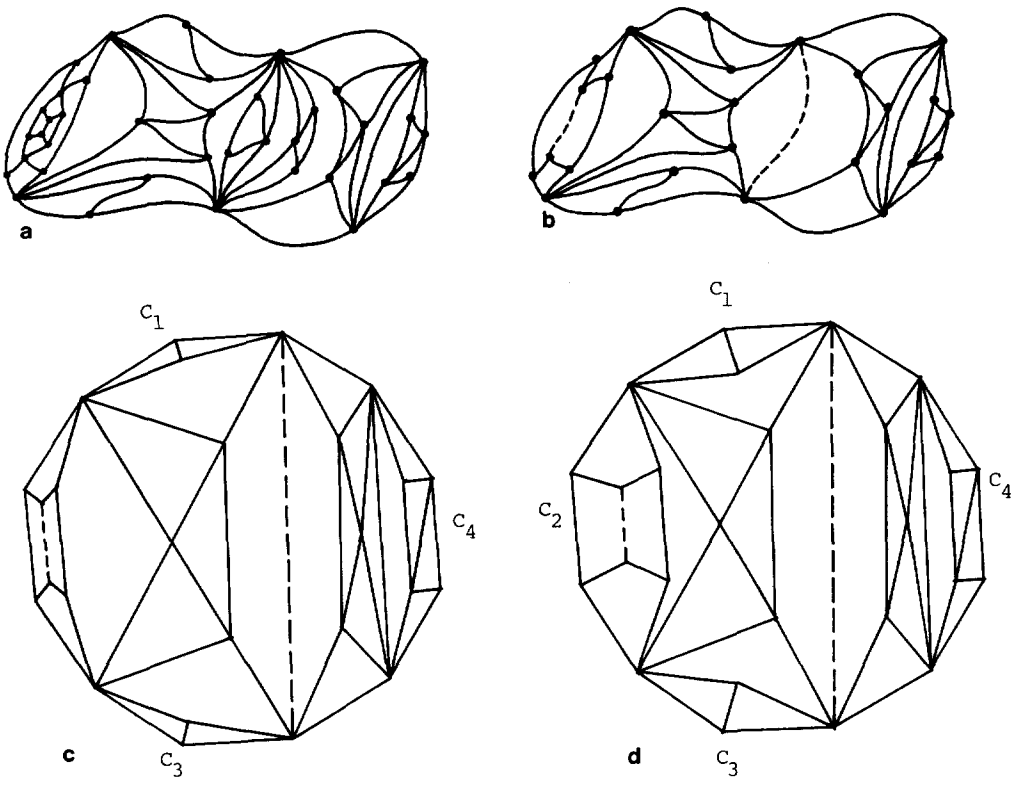

Fig. 4a–d. An illustrative example of Lemma 2: **a** a given graph; **b** the maximal subgraph G' of G having a convex drawing; **c** a convex drawing of G'; and **d** a straight line drawing of G'

We illustrate Lemma 2 in Fig. 4. An original 2-connected plane graph G is depicted in Fig. 4a, and G' in Fig. 4b. G' can be drawn convex as in Fig. 4c.

In Fig. 4c split components having edges of S, such as C_1, C_2, and C_3, are drawn in rather ill-shapes (not round), as we pointed earlier in the introduction. Therefore we do not use G' as the frame, but furthermore split from G' $\{x, y\}$-split components having edges of S. Thus we split C_1, C_2 and C_3 from G'. Since the 3-connected component C_4 is parallel with two other components, G' has no straight-line drawing in which the outer cycle of C_4 is drawn round. Hence we do not split C_4 from G' although C_4 has edges of S. More precisely, we do not split from G' any $\{x, y\}$-split component if G' has edge (x, y). The resulting graph is called a *frame* F of G. We draw F convex, and embed the convex drawings of splitted components into the drawing of F. The resulting drawing is depicted in Fig. 4d.

The algorithm is not so simple as above, because a splitted component, C_2 for example, may also have a split component. Thus we must use recursion. Our algorithm proceeds as follows.

```
procedure DRAW (G);
    {G is a 2-connected plane graph.}
    begin
        determine a frame F of G;
        let S_F be the outer cycle of F;
        choose as S_F* an appropriate extendible convex polygon of S_F;
        draw the frame convex by CONVEX-DRAW (F, S_F, S_F*);
        for each virtual edge e_i in F
            do find in the drawing of F a region R_i, called a usable region, in
               which the 3-connected component C_i corresponding to e_i can be
               drawn;
        while there exists a virtual edge e_i in the drawing already obtained
            do begin
                   let C_i be a 3-connected component having a virtual edge e_i
                   which is not drawn so far;
                   determine in the usable region R_i an appropriate extendible
                   convex polygon S_i* of the outer cycle S_i of C_i − e_i;
                   draw C_i − e_i by CONVEX-DRAW (C_i − e_i, S_i, S_i*);
                   embed the convex drawing of C_i − e_i into R_i in the drawing
                   obtained so far
                   for each new virtual edge e_j
                       do find a usable region R_j
               end
    end;
```

An execution of the algorithm is illustrated in Fig. 5. We now consider more precisely how to draw a 3-connected component C_i. First consider the case that C_i is a ring. In this case we can easily draw $C_i - e_i$ by placing the vertices of C_i on the straight line between the two ends of e_i. Next consider the case that C_i is 3-connected. In this case one can easily show that $C_i - e_i$ satisfies Condition I and hence we draw $C_i - e_i$ convex by CONVEX-DRAW. Finally consider the case that C_i is a bond. Let x and y be the ends of e_i. Clearly $C_i - e_i$ cannot be drawn convex. Therefore we draw the 3-connected components having virtual edges (x, y) as follows. (We illustrate the case in Fig. 6.) While there exist two or more 3-connected components which have not been drawn so far, iterate the following:

(a) merge a pair of them with respect to the common virtual edge (x, y);

(b) draw it convex in the usable region R_i;

(c) redefine R_i as a convex region (shaded in Fig. 7c) determined appropriately in the interior of the drawing of the pair.

When the iteration above terminates, there are four possibilities.

(1) The entire drawing of $C_i - e_i$ has been obtained;

(2) Exactly one 3-connected component C *having virtual edge* (x, y) remains undrawn;

(3) Exactly one real edge (x, y) remains undrawn; and

(4) Exactly one real edge $e_r = (x, y)$ and one 3-connected component C having virtual edge $e = (x, y)$ remain drawn.

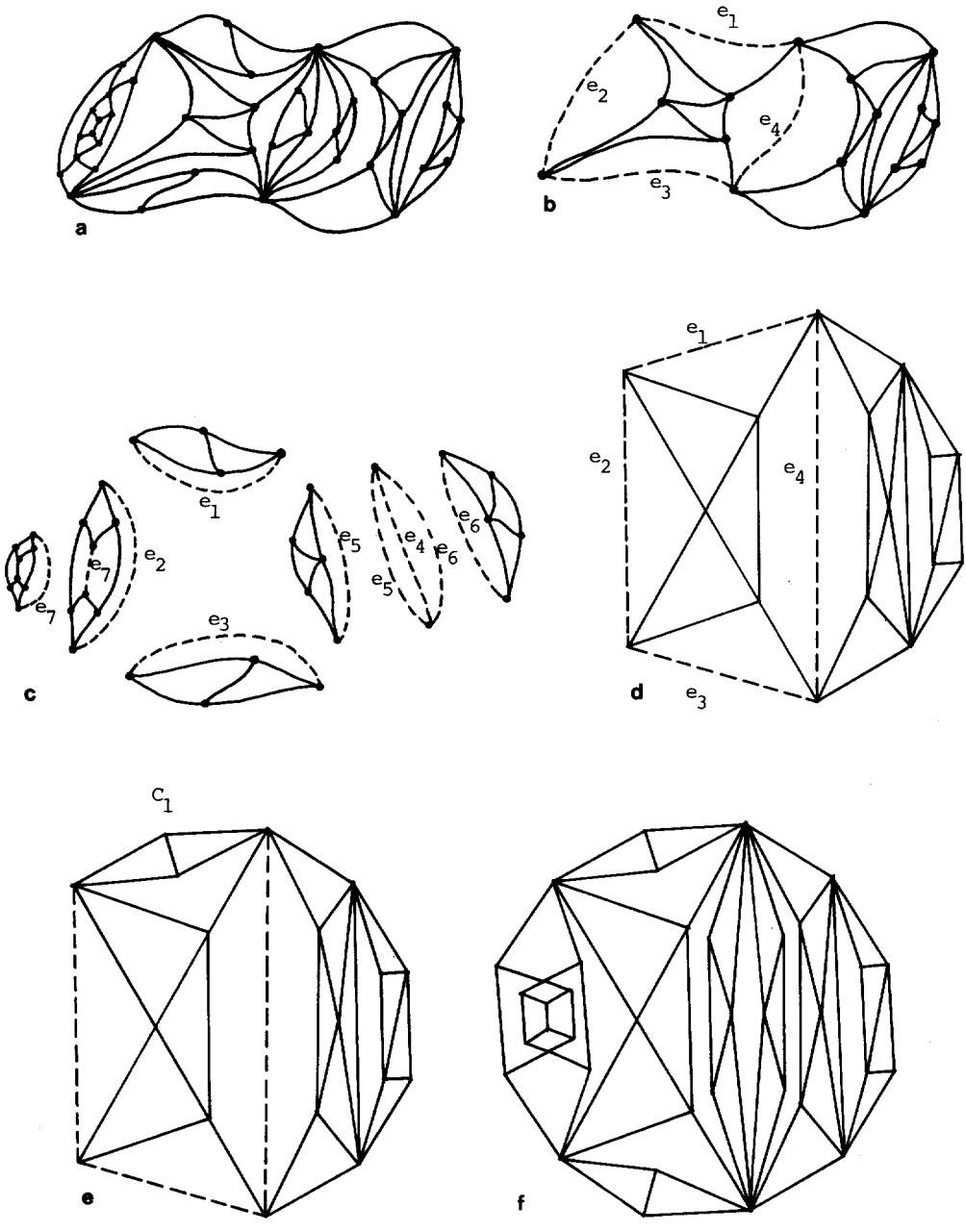

Fig. 5a–f. Illustrations of an execution of algorithm Draw: **a** a given graph G; **b** frame F; **c** remaining 3-connected components; **d** a convex drawing of F; **e** a drawing after merging F and component C_1; and **f** a final drawing

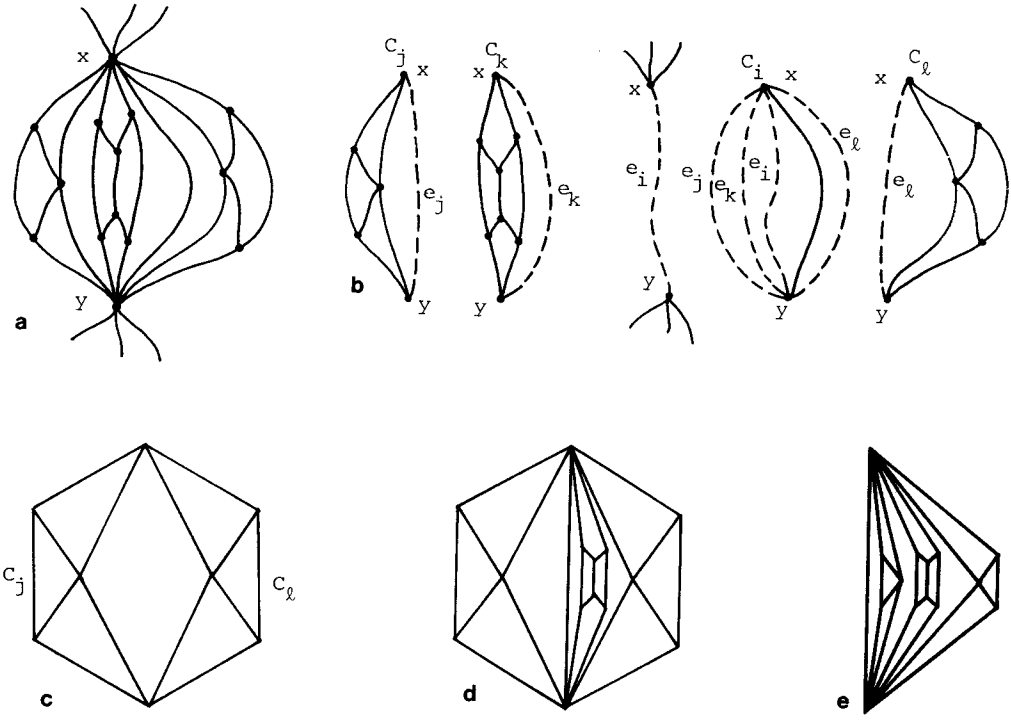

Fig. 6a–e. Illustrations of drawing a bond: **a** a part of a given graph; **b** the $\{x, y\}$-split components; **c** a drawing of a graph constructed by merging C_j and C_i; **d** an entire drawing of the $\{x, y\}$-split components; **e** an ill-shaped drawing

In Case (2) we draw $C-(x, y)$ convex in R_i. In Case (3) we simply draw the real edge (x, y) as a straight line in R_i. Finally in Case (4) we draw $(C-e)\cup e_r$ convex in R_i. One can easily verify that $(C-e)\cup e_r$ satisfies Condition I. To avoid an ill-shaped drawing as shown in Fig. 6e, we draw 3-connected components in parallel as above, although it may change the given embedding of G.

We now consider how to determine a usable region R_i for each virtual edge e_i. First consider the case in which e_i is in F. We first place a point in each interior face (for example, at the centroid of the convex polygon). Then draw straight lines going from the point to each apex of the convex polygon containing the point. Thus convex polygons are divided into triangles. If a virtual edge e_i is not on the outer cycle S_F of F, then we determine R_i as the union of the two triangles adjacent to e_i (see Fig. 7a). If a virtual edge e_i is on S_F, then we determine R_i as the union of the two triangles, a triangle adjacent to e_i and the mirror image (see Fig. 7b). Next consider the case in which e_i is not in F. Assume that e_i is contained in a 3-connected component C_j drawn so far. Let e_j be the virtual edge contained in C_j, and let R_j be the usable region for e_j in which C_j-e_j is drawn. R_j is a quadrangle which is divided into two triangles by e_j. If a virtual edge e_i is not on the outer cycle of a component,

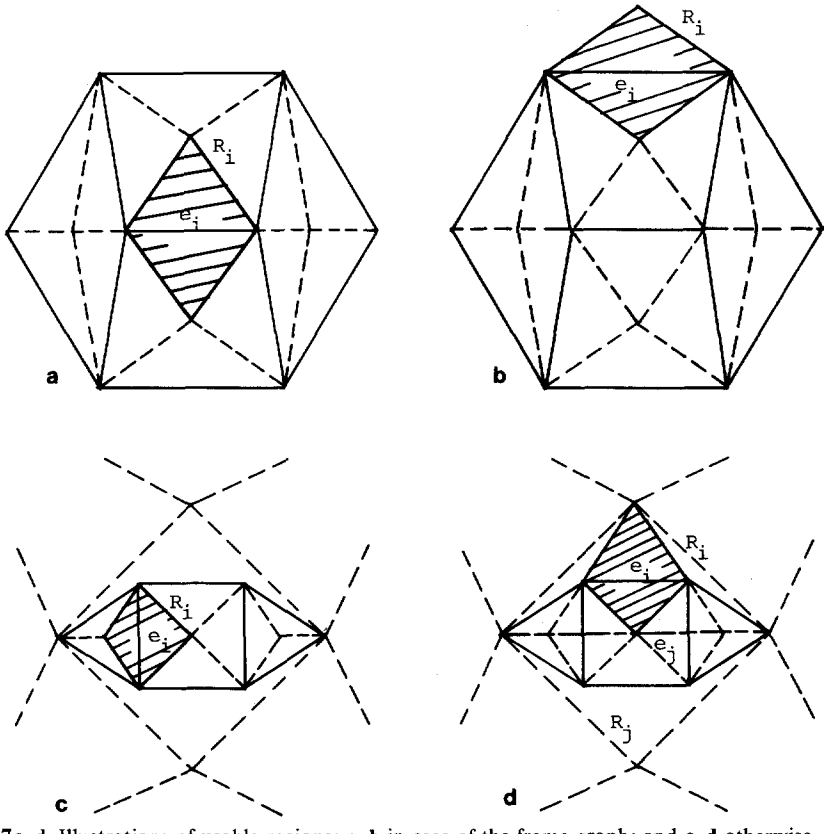

Fig. 7a–d. Illustrations of usable regions: **a**, **b** in case of the frame graph; and **c**, **d** otherwise

then we determine R_i in the same way as in Fig. 7a (see Fig. 7c). If virtual edge e_i is on the outer cycle (see Fig. 7d), then we determine R_i as the union of two triangles, one is a triangle adjacent to e_i in the drawing of C_j, and the other is a triangle with three apices: the two ends of e_i and the apex of R_j which lies on the same side as e_i.

A usable region R_i obtained as above is necessarily a quadrangle which is divided into two triangles by e_i. Although R_i is not necessarily convex, we can determine in R_i a convex polygon of the outer facial cycle of a component C_i. Moreover it is obvious that all the usable regions are pairwise disjoint. Thus we have the following theorem.

Theorem 2. *Given a 2-connected plane graph G, the algorithm Draw produces a straight line drawing of G in linear time.*

Proof. Using Hopcroft and Tarjan's decomposition algorithm [4], we can determine a frame and 3-connected components within linear time. Algorithm CONVEX-DRAW produces the drawings of the frame and the remaining 3-connected components in linear time. Therefore a drawing of the whole graph G is obtained within time and space proportional to the total number of real

and virtual edges. It is clear from the definition of splitting that the number of virtual edges is at most three times of real edges [4]. Therefore the total is linear in the size of G. Thus we can conclude that DRAW runs in linear time. Q.E.D.

5. Examples

In this section we present computational examples. The algorithms CONVEX-DRAW and DRAW have been implemented in PASCAL and run on a small computer FACOM 230/38 s.

Figure 8a, b, and c depict three drawings obtained by our algorithm DRAW. The shape of a drawing depends on not only a convex drawing

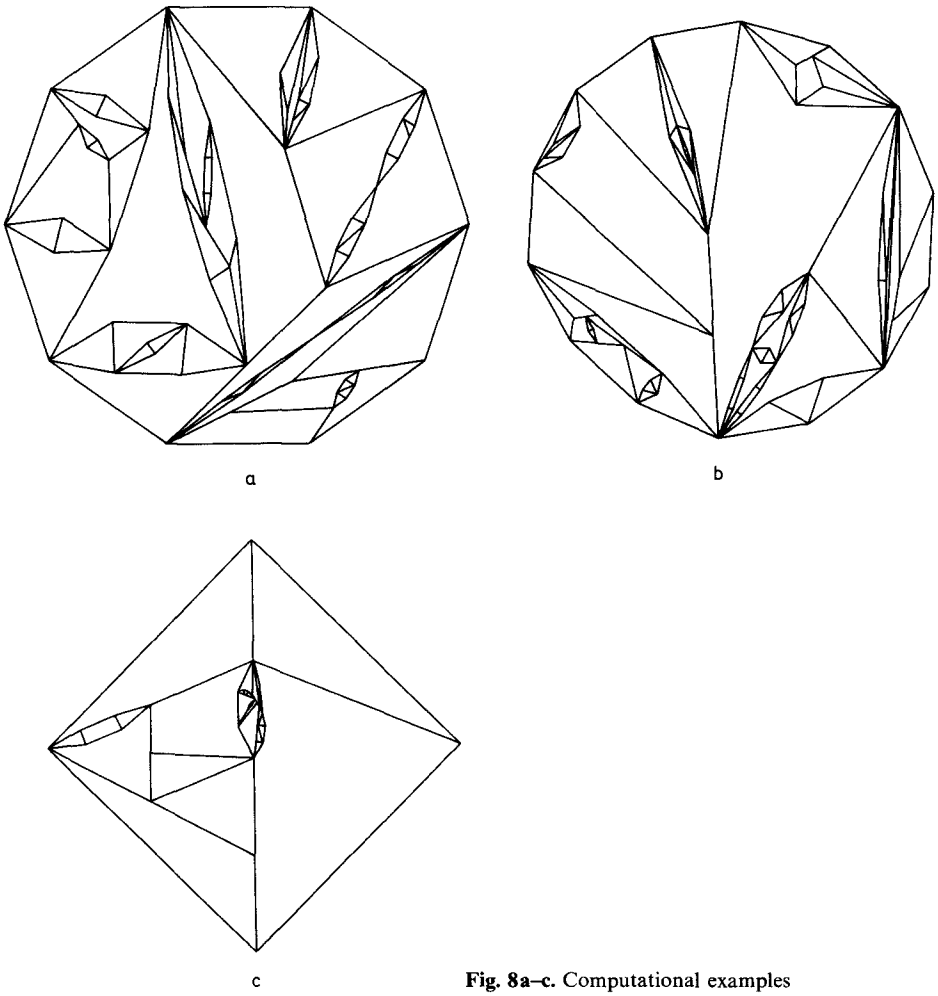

Fig. 8a–c. Computational examples

algorithm employed but also an embedding of a graph. It is not obvious which embedding gives us a favorable drawing. One reasonable choice is to embed a graph so that the outer cycle contains as many separation pairs as possible. However for practical applications, it is sufficient to choose a pleasing drawing from several drawings obtained by executing the algorithm for various embeddings. It remains open to improve the drawing algorithm so that it gives more favorable drawings with respect to another criterion, for example, in a way that the areas of faces are balanced.

Acknowledgement. We wish to thank professor Nobuji Saito for stimulating discussion on the subjects, and also the referees for many helpful comments which improved the presentation of the paper.

References

1. Chiba, N., Nishizeki, T., Abe, S., Ozawa, T.: A linear algorithm for embedding planar graphs using PQ-trees. J. Comput. Syst. Sci. (To appear)
2. Chiba, N., Yamanouchi, T., Nishizeki, T.: Linear algorithms for convex drawings of planar graphs. In: Progress in Graph Theory (J.A. Bondy, U.S.R. Murty, eds.), pp. 153–173. Toronto: Academic Press 1984
3. Fary, I.: On straight representations of planar graphs. Acta Sci. Math. Szeged **11**, 229–233 (1948)
4. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. Comput. **2, 3**, 135–158 (1973)
5. Hopcroft, J.E., Tarjan, R.E.: Efficient planarity testing. J. ACM **21**, 549–568 (1974)
6. Lipton, R.J., Rose, D.J., Tarjan, R.E.: Generalized nested dissection. SIAM J. Numer. Anal. **16, 2**, 346–358 (1979)
7. Reingold, E.M., Tilford, J.S.: Tidier drawings of trees. IEEE Trans. Software Eng. **3**, 223–228 (1981)
8. Supowit, K.J., Reingold, E.M.: The complexity of drawing trees nicely. Acta Inf. **18**, 377–392 (1983)
9. Thomassen, C.: Planarity and duality of finite and infinite graphs. J. Comb. Theory, Ser. B **29**, 244–271 (1980)
10. Tutte, W.T.: Convex representations of graphs. Proc. Lond. Math. Soc., (3) **10**, 304–320 (1960)
11. Tutte, W.T.: How to draw a graph. Proc. Lond. Math. Soc. **13**, 743–768 (1963)
12. Vaucher, J.G.: Pretty-printing of trees. Software, Pract. Exper. **10**, 553–561 (1980)
13. Wetherell, C., Shannon, A.: Tidy drawings of trees. IEEE Trans. Software Eng. **5**, 514–520 (1970)

Received January 27, 1984/September 27, 1984