Chapter 1

Fundamentals

1.1 Models of Computation

When designing algorithms, one has to agree on a model of computation according to which these algorithms can be executed. There are various such models, but when it comes to geometry some are more convenient to work with than others. Even using very elementary geometric operations—such as taking the center of a circle defined by three points or computing the length of a given circular arc—the realms of rational and even algebraic numbers are quickly left behind. Representing the resulting real numbers/coordinates would be a rather painful task in, for instance, a Turing machine type model of computation.

Therefore, other models of computation are more prominent in the area of geometric algorithms and data structures. In this course we will be mostly concerned with two models: the *Real RAM* and the *algebraic computation/decision tree* model. The former is rather convenient when designing algorithms, because it sort of abstracts from the aforementioned representation issues by simply *assuming* that it can be done. The latter model typically appears in the context of lower bounds, that is, proofs that certain problems cannot be solved more efficiently than some function depending on the problem size (and possibly some other parameters).

So let us see what these models are in more detail.

Real RAM Model. A memory cell stores a real number (that is what the "Real" stands for)[1]. Any single arithmetic operation (addition, subtraction, multiplication, division, and k-th root, for small constant k) or comparison can be computed in constant time.[2] This is a quite powerful (and somewhat unrealistic) model of computation, as a single real number in principle can encode an arbitrary amount of information. Therefore we

[1] RAM stands for random access machine, meaning that every memory cell can be accessed in constant time. Not like, say, a list where one always has to start from the first element.

[2] In addition, sometimes also logarithms, other analytic functions, indirect addressing (integral), or floor and ceiling are used. As adding some of these operations makes the model more powerful, it is usually specified and emphasized explicitly when an algorithm uses them.

have to ensure that we do not abuse the power of this model. For instance, we may want to restrict the numbers that are manipulated by any single arithmetic operation to be bounded by some fixed polynomial in the numbers that appear in the input.

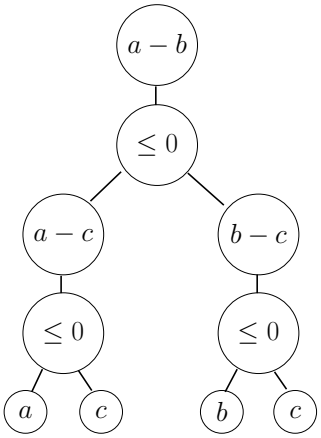
On the positive side, the real RAM model allows to abstract from the lowlands of numeric and algebraic computation and to concentrate on the algorithmic core from a combinatorial point of view.

But there are also downsides to using such a powerful model. In particular, it may be a challenge to efficiently implement a geometric algorithm designed for the real RAM on an actual computer. With bounded memory there is no way to represent general real numbers explicitly, and operations using a symbolic representation can hardly be considered constant time.

When interested in lower bounds, it is convenient to use a model of computation that encompasses and represents explicitly all possible execution paths of an algorithm. This is what the following model is about.

Algebraic Computation Trees (Ben-Or [1]). A computation is regarded as a binary tree.

- The leaves contain the (possible) results of the computation.
- Every node v with one child has an operation of the form $+, -, *, /, \sqrt{}, \ldots$ associated to it. The operands of this operation are constant input values, or among the ancestors of v in the tree.
- Every node v with two children has associated to it a branching of the form > 0, $\geqslant 0$, or $= 0$. The branch is with respect to the result of v's parent node. If the expression yields `true`, the computation continues with the left child of v; otherwise, it continues with the right child of v.



The term *decision tree* is used if all of the final results (leaves) are either `true` or `false`. If every branch is based on a linear function in the input values, we face a *linear decision tree*. Analogously one can define, say, quadratic decision trees.

The complexity of a computation or decision tree is the maximum number of vertices along any root-to-leaf path. It is well known that $\Omega(n \log n)$ comparisons are required to sort n numbers. But also for some problems that appear easier than sorting at first glance, the same lower bound holds. Consider, for instance, the following problem.

Element Uniqueness

Input: $\{x_1, \ldots, x_n\} \subset \mathbb{R}$, $n \in \mathbb{N}$.

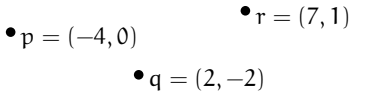
Output: Is $x_i = x_j$, for some $i, j \in \{1, \ldots, n\}$ with $i \neq j$?

Ben-Or [1] has shown that any algebraic decision tree to solve Element Uniqueness for n elements has complexity $\Omega(n \log n)$.

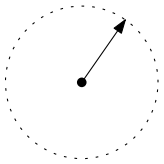
1.2 Basic Geometric Objects

We will mostly be concerned with the d-dimensional Euclidean space $\mathbb{R}^d$, for small $d \in \mathbb{N}$; typically, $d = 2$ or $d = 3$. The basic objects of interest in $\mathbb{R}^d$ are the following.

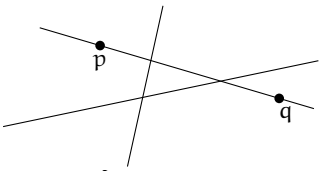
Points. A point p, typically described by its d Cartesian coordinates $p = (x_1, \ldots, x_d)$.



Directions. A vector $v \in \mathbb{S}^{d-1}$ (the $(d-1)$-dimensional unit sphere), typically described by its d Cartesian coordinates $v = (x_1, \ldots, x_d)$, with $\|v\| = \sqrt{\sum_{i=1}^{d} {x_i}^2} = 1$.

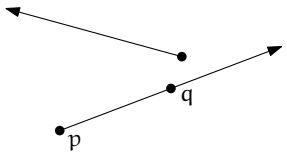


Lines. A line is a one-dimensional affine subspace. It can be described by two distinct points p and q as the set of all points r that satisfy $r = p + \lambda(q - p)$, for some $\lambda \in \mathbb{R}$.

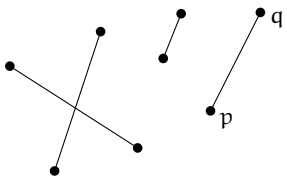


While any pair of distinct points defines a unique line, a line in $\mathbb{R}^2$ contains infinitely many points and so it may happen that a collection of three or more points lie on a line. Such a collection of points is termed *collinear*[3].

Rays. If we remove a single point from a line and take the closure of one of the connected components, then we obtain a ray. It can be described by two distinct points p and q as the set of all points r that satisfy $r = p + \lambda(q - p)$, for some $\lambda \geqslant 0$. The *orientation* of a ray is the direction $(q - p)/\|q - p\|$.



Line segment. A line segment is a compact connected subset of a line. It can be described by two points p and q as the set of all points r that satisfy $r = p + \lambda(q - p)$, for some $\lambda \in [0, 1]$. We will denote the line segment through p and q by $\overline{pq}$. Depending on the context we may allow or disallow *degenerate* line segments consisting of a single point only ($p = q$ in the above equation).



Hyperplanes. A hyperplane $\mathcal{H}$ is a $(d-1)$-dimensional affine subspace. It can be described algebraically by $d + 1$ coefficients $\lambda_1, \ldots, \lambda_{d+1} \in \mathbb{R}$, where $\|(\lambda_1, \ldots, \lambda_{d+1})\| = 1$, as the set of all points $(x_1, \ldots, x_d)$ that satisfy the linear equation $\mathcal{H} : \sum_{i=1}^{d} \lambda_i x_i = \lambda_{d+1}$.

[3] Not *colinear*, which refers to a notion in the theory of coalgebras.

If the above equation is converted into an inequality, we obtain the algebraic description of a *halfspace* (in $\mathbb{R}^2$: halfplane).

Spheres and balls. A sphere is the set of all points that are equidistant to a fixed point. It can be described by a point c (center) and a number $\rho \in \mathbb{R}$ (radius) as the set of all points p that satisfy $\|p - c\| = \rho$. The *ball* of radius ρ around p consists of all points p that satisfy $\|p - c\| \leqslant \rho$.

1.3 Graphs

In this section we review some basic definitions and properties of graphs. For more details and proofs, refer to any standard textbook on graph theory [2, 3, 5].

An (undirected) graph $G = (V, E)$ is defined on a set V of *vertices*. Unless explicitly stated otherwise, V is always finite. Vertices are associated to each other through *edges* which are collected in the set $E \subseteq \binom{V}{2}$. The two vertices defining an edge are *adjacent* to each other and *incident* to the edge.

For a vertex $v \in V$, denote by $N_G(v)$ the *neighborhood* of v in G, that is, the set of vertices from G that are adjacent to v. Similarly, for a set $W \subset V$ of vertices define $N_G(W) := \bigcup_{w \in W} N_G(w)$. The *degree* $\deg_G(v)$ of a vertex $v \in V$ is the size of its neighborhood, that is, the number of edges from E incident to v. The subscript is often omitted when it is clear which graph it refers to.

Lemma 1.1 (Handshaking Lemma) *In any graph* $G = (V, E)$ *we have* $\sum_{v \in V} \deg(v) = 2|E|$.

Two graphs $G = (V, E)$ and $H = (U, W)$ are *isomorphic* if there is a bijection $\phi : V \to U$ such that $\{u, v\} \in E \iff \{\phi(u), \phi(v)\} \in W$. Such a bijection ϕ is called an *isomorphism* between G and H. The structure of isomorphic graphs is identical and often we do not distinguish between them when looking at them as graphs.

For a graph G denote by $V(G)$ the set of vertices and by $E(G)$ the set of edges. A graph $H = (U, F)$ is a *subgraph* of G if $U \subseteq V$ and $F \subseteq E$. In case that $U = V$ the graph H is a *spanning* subgraph of G. For a set $W \subseteq V$ of vertices denote by $G[W]$ the *induced subgraph* of W in G, that is, the graph $(W, E \cap \binom{W}{2})$. For $F \subseteq E$ let $G \setminus F := (V, E \setminus F)$. Similarly, for $W \subseteq V$ let $G \setminus W := G[V \setminus W]$. In particular, for a vertex or edge $x \in V \cup E$ we write $G \setminus x$ for $G \setminus \{x\}$. The *union* of two graphs $G = (V, E)$ and $H = (W, F)$ is the graph $G \cup H := (V \cup W, E \cup F)$.

For an edge $e = \{u, v\} \in E$ the graph G/e is obtained from $G \setminus \{u, v\}$ by adding a new vertex w with $N_{G/e}(w) := (N_G(u) \cup N_G(v)) \setminus \{u, v\}$. This process is called *contraction* of e in G. Similarly, for a set $F \subseteq E$ of edges the graph G/F is obtained from G by contracting all edges from F.

Graph traversals. A *walk* in G is a sequence $W = (v_1, \ldots, v_k)$, $k \in \mathbb{N}$, of vertices such that v_i and v_{i+1} are adjacent in G, for all $1 \leqslant i < k$. The vertices v_1 and v_k are referred

to as the walk's *endpoints*, the other vertices are called *interior*. A walk with endpoints v_1 and v_k is sometimes referred to as a *walk between* v_1 and v_k. For a walk W denote by $V(W)$ its set of vertices and by $E(W)$ its set of edges (pairs of vertices adjacent along W). We say that W *visits* the vertices and edges in $V(W) \cup E(W)$. A walk for which both endpoints coincide, that is, $v_1 = v_k$, is called *closed*. Otherwise the walk is *open*.

If a walk uses each edge of G at most once, it is a *trail*. A closed walk that visits each edge and each vertex at least once is called a *tour* of G. An *Euler tour* is both a trail and a tour of G, that is, it visits each edge of G exactly once. A graph that contains an Euler tour is termed *Eulerian*.

If the vertices $v_1, \ldots, v_k$ of a closed walk W are pairwise distinct except for $v_1 = v_k$, then W is a *cycle* of size $k-1$. If the vertices $v_1, \ldots, v_k$ of a walk W are pairwise distinct, then W is a *path* of size k. A *Hamilton cycle (path)* is a cycle (path) that visits every vertex of G. A graph that contains a Hamilton cycle is *Hamiltonian*.

Two trails are *edge-disjoint* if they do not share any edge. Two paths are called (internally) *vertex-disjoint* if they do not share any vertices (except for possibly common endpoints). For two vertices $s, t \in V$ any path with endpoints s and t is called an (s,t)-*path* or a *path between* s and t.

Connectivity. Define an equivalence relation "$\sim$" on V by setting $a \sim b$ if and only if there is a path between a and b in G. The equivalence classes with respect to "$\sim$" are called *components* of G and their number is denoted by $\omega(G)$. A graph G is *connected* if $\omega(G) = 1$ and *disconnected*, otherwise.

A set $C \subset V$ of vertices in a connected graph $G = (V, E)$ is a *cut-set* of G if $G \setminus C$ is disconnected. A graph is *k-connected*, for a positive integer k, if $|V| \geqslant k+1$ and there is no cut-set of size less than k. Similarly a graph $G = (V, E)$ is *k-edge-connected*, if $G \setminus F$ is connected, for any set $F \subseteq E$ of less than k edges. Connectivity and cut-sets are related via the following well-known theorem.

Theorem 1.2 (Menger [4]) *For any two non-adjacent vertices u, v of a graph $G = (V, E)$, the size if a minimum cut that disconnects u and v is the same as the maximum number of pairwise internally vertex-disjoint paths between u and v.*

Specific families of graphs. A graph with a maximum number of edges, that is, $(V, \binom{V}{2})$, is called a *clique*. Up to isomorphism there is only one clique on n vertices; it is referred to as the *complete graph* K_n, $n \in \mathbb{N}$. At the other extreme, the *empty graph* $\overline{K_n}$ consists of n isolated vertices that are not connected by any edge. A set of U of vertices in a graph G is *independent* if $G[U]$ is an empty graph. A graph whose vertex set can be partitioned into at most two independent sets is *bipartite*. An equivalent characterization states that a graph is bipartite if and only if it does not contain any odd cycle. The bipartite graphs with a maximum number of edges (unique up to isomorphism) are the *complete bipartite graphs* $K_{m,n}$, for $m, n \in \mathbb{N}$. They consist of two disjoint independent sets of size m and n, respectively, and all mn edges in between.

A *forest* is a graph that is *acyclic*, that is, it does not contain any cycle. A connected forest is called *tree* and its *leaves* are the vertices of degree one. Every connected graph contains a spanning subgraph which is a tree, a so called *spanning tree*. Beyond the definition given above, there are several equivalent characterizations of trees.

Theorem 1.3 *The following statements for a graph* G *are equivalent.*

(1) G *is a tree (i.e., it is connected and acyclic).*

(2) G *is a connected graph with* n *vertices and* $n-1$ *edges.*

(3) G *is an acyclic graph with* n *vertices and* $n-1$ *edges.*

(4) Any two vertices in G *are connected by a unique path.*

(5) G *is minimally (edge-)connected, that is,* G *is connected but removal of any single edge yields a disconnected graph.*

(6) G *is maximally acyclic, that is,* G *is acyclic but adding any single edge creates a cycle.*

Directed graphs. In a directed graph or, short, *digraph* $D = (V, E)$ the set E consists of ordered pairs of vertices, that is, $E \subseteq V^2$. The elements of E are referred to as *arcs*. An arc $(u, v) \in E$ is said to be directed from its *source* u to its *target* v. For $(u, v) \in E$ we also say "there is an arc from u to v in D". Usually, we consider *loop-free* graphs, that is, arcs of the type (v, v), for some $v \in V$, are not allowed.

The *in-degree* $\deg^-_D(v) := |\{(u, v) \mid (u, v) \in E\}|$ of a vertex $v \in V$ is the number of *incoming* arcs at v. Similarly, the *out-degree* $\deg^+_D(v) := |\{(v, u) \mid (v, u) \in E\}|$ of a vertex $v \in V$ is the number of *outgoing* arcs at v. Again the subscript is often omitted when the graph under consideration is clear from the context.

From any undirected graph G one can obtain a digraph on the same vertex set by specifying a direction for each edge of G. Each of these $2^{|E(G)|}$ different digraphs is called an *orientation* of G. Similarly every digraph $D = (V, E)$ has an *underlying* undirected graph $G = (V, \{\{u, v\} \mid (u, v) \in E \text{ or } (v, u) \in E\})$. Hence most of the terminology for undirected graphs carries over to digraphs.

A *directed walk* in a digraph D is a sequence $W = (v_1, \ldots, v_k)$, for some $k \in \mathbb{N}$, of vertices such that there is an arc from v_i to v_{i+1} in D, for all $1 \leqslant i < k$. In the same way we define *directed trails*, *directed paths*, *directed cycles*, and *directed tours*.

References

[1] Michael Ben-Or, Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pp. 80–86, 1983, URL `http://dx.doi.org/10.1145/800061.808735`.

[2] John Adrian Bondy and U. S. R. Murty, *Graph Theory*, vol. 244 of *Graduate texts in Mathematics*. Springer-Verlag, New York, 2008, URL http://dx.doi.org/10.1007/978-1-84628-970-5.

[3] Reinhard Diestel, *Graph Theory*. Springer-Verlag, Heidelberg, 4th edn., 2010.

[4] Karl Menger, Zur allgemeinen Kurventheorie. *Fund. Math.*, **10**, 1, (1927), 96—-115, URL http://matwbn.icm.edu.pl/ksiazki/fm/fm10/fm1012.pdf.

[5] Douglas B. West, *An Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 2nd edn., 2001.

Chapter 2

Plane Embeddings

In this chapter we investigate properties of plane embeddings and under which conditions they hold.

2.1 Embeddings and planarity

A *curve* is a set $C \subset \mathbb{R}^2$ that is of the form $\{\gamma(t) \mid 0 \leqslant t \leqslant 1\}$, where $\gamma : [0,1] \to \mathbb{R}^2$ is a continuous function. The function γ is called a *parameterization* of C. The points $\gamma(0)$ and $\gamma(1)$ are the *endpoints* of the curve. For a *closed* curve, we have $\gamma(0) = \gamma(1)$. A curve is *simple*, if it admits a parameterization γ that is injective on $[0,1]$. For a closed simple curve we allow as an exception that $\gamma(0) = \gamma(1)$. The following famous theorem describes an important property of the plane. A proof can, for instance, be found in the book of Mohar and Thomassen [18].

Theorem 2.1 (Jordan) *Any simple closed curve C partitions the plane into exactly two regions (connected open sets), each bounded by C.*

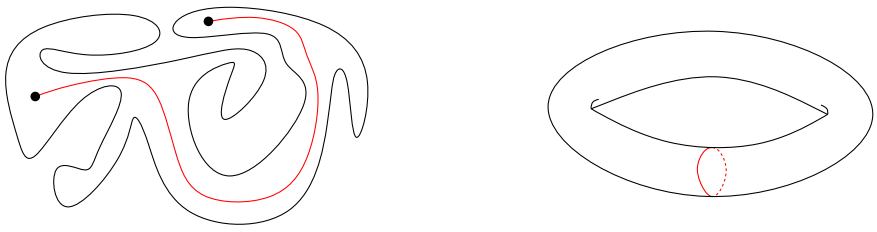


Figure 2.1: *A Jordan curve and two points in one of its faces (left); a simple closed curve that does not disconnect the torus (right).*

Observe that, for instance, on the torus there are closed curves that do not disconnect the surface (and so the theorem does not hold there).

An *embedding* or *drawing* of a (multi-)graph $G = (V, E)$ into the plane is a function $f : V \cup E \to \mathbb{R}^2$ that assigns

- a point $f(v)$ to every vertex $v \in V$ and
- a simple curve $f(\{u, v\})$ with endpoints $f(u)$ and $f(v)$ to every edge $\{u, v\} \in E$,

such that f is injective on V and $f(\{u, v\}) \cap f(V) = \{f(u), f(v)\}$, for every edge $\{u, v\} \in E$. A common point $f(e) \cap f(e')$ between two curves that represent edges $e \neq e' \in E$ is called a *crossing*, unless it is a common endpoint of e and e'. In many cases it is convenient to demand that no three edges share a crossing.

Planar vs. plane. A multigraph is *planar* if it admits an embedding without crossings into the plane. Such an embedding is also called a *plane* or *crossing-free* embedding. A planar graph together with a particular plane embedding is called a *plane* graph. Note the distinction between "planar" and "plane": the former indicates the possibility of an embedding, whereas the latter refers to a concrete embedding (Figure 2.2).

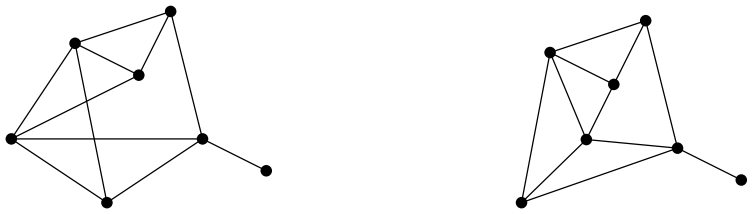


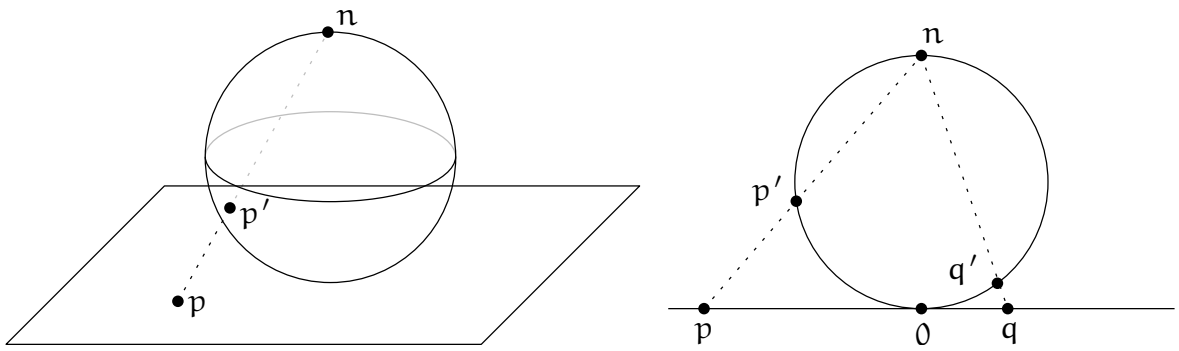
Figure 2.2: *A planar graph (left) and a plane drawing of it (right).*

A *geometric graph* is a graph together with an embedding, in which all edges are realized as straight-line segments. Note that such an embedding is completely defined by the mapping for the vertices. A plane geometric graph is also called a *plane straight-line graph* (PSLG). In contrast, a plane graph in which the edges may form arbitrary simple curves is called a *topological plane graph*.

The *faces* of a plane multigraph are the maximally connected regions of the plane that do not contain any point used by the embedding (as the image of a vertex or an edge). Each embedding of a finite multigraph has exactly one *unbounded face*, also called *outer* or *infinite* face. Using stereographic projection, it is not hard to show that the role of the unbounded face is not as special at it may seem at first glance.

Theorem 2.2 *If a graph G has a plane embedding in which some face is bounded by the cycle $(v_1, \ldots, v_k)$, then G also has a plane embedding in which the unbounded face is bounded by the cycle $(v_1, \ldots, v_k)$.*

Proof. (Sketch) Take a plane embedding Γ of G and map it to the sphere using *stereographic projection*: Imagine $\mathbb{R}^2$ being the x/y-plane in $\mathbb{R}^3$ and place a unit sphere S such that its south pole touches the origin. We obtain a bijective continuous mapping between $\mathbb{R}^2$ and $S \setminus \{n\}$, where n is the north pole of S, as follows: A point $p \in \mathbb{R}^2$ is mapped to the point p' that is the intersection of the line through p and n with S, see Figure 2.3.



(a) Three-dimensional view. (b) Cross-section view.

Figure 2.3: *Stereographic projection.*

Consider the resulting embedding Γ' of G on S: The infinite face of Γ corresponds to the face of Γ' that contains the north pole n of S. Now rotate the embedding Γ' on S such that the desired face contains n. Mapping back to the plane using stereographic projection results in an embedding in which the desired face is the outer face. □

Exercise 2.3 *Consider a graph* G *with the plane embedding depicted in Figure 2.4. Give a plane embedding of* G *in which the cycle* $1, 2, 3$ *bounds the outer face.*

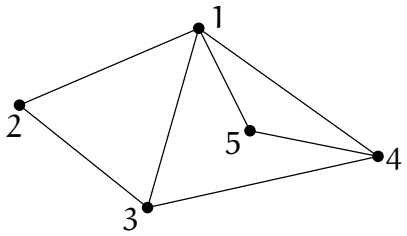


Figure 2.4: *Plane embedding of* G.

Duality. Every plane graph G has a *dual* G^*, whose vertices are the faces of G and two are connected by an edge in G^*, if and only if they have a common edge in G. In general, G^* is a multigraph (may contain loops and multiple edges) and it depends on the embedding. That is, an abstract planar graph G may have several non-isomorphic duals. If G is a connected plane graph, then $(G^*)^* = G$. We will show later in Section 2.3 that the dual of a 3-connected planar is unique (up to isomorphism).

The Euler Formula and its ramifications. One of the most important tools for planar graphs (and more generally, graphs embedded on a surface) is the Euler–Poincaré Formula.

Theorem 2.4 (Euler's Formula) *For every connected plane graph with* n *vertices,* e *edges, and* f *faces, we have* $n - e + f = 2$.

Figure 2.5: *Two plane drawings and their duals for the same planar graph.*

In particular, this shows that for any planar graph the number of faces is the same in every plane embedding. Therefore, the number of faces is actually a parameter of an abstract planar graph. It also follows (stated below as a corollary) that planar graphs are *sparse*, that is, they have a linear number of edges (and faces) only. So the asymptotic complexity of a planar graph is already determined by its number of vertices.

Corollary 2.5 *A simple planar graph on $n \geqslant 3$ vertices has at most $3n-6$ edges and at most $2n-4$ faces.*

Proof. The statement is easily checked for $n = 3$, where G is either a triangle or a path and, therefore, has no more than $3 \cdot 3 - 6 = 3$ edges and no more than $2 \cdot 3 - 4 = 2$ faces. So consider a simple planar graph G on $n \geqslant 4$ vertices. Without loss of generality we may assume that G is connected. (If not, add edges between components of G until the graph is connected. The number of faces remains unchanged and the number of edges only increases.) Consider a plane drawing of G and denote by E the set of edges and by F the set of faces of G. Let

$$X = \{(e, f) \in E \times F \mid e \text{ bounds } f\}$$

denote the set of incident edge-face pairs. We count X in two different ways.

First note that each edge bounds at most two faces and so $|X| \leqslant 2 \cdot |E|$.

Second note that in a simple connected planar graph on four or more vertices every face is bounded by at least three vertices: Every bounded face needs at least three edges to be enclosed and if there is no cycle on the boundary of the unbounded face, then—given that G is connected—G must be a tree on four or more vertices and so its has at least three edges, all of which bound the unbounded face. Therefore $|X| \geqslant 3 \cdot |F|$.

Using Euler's Formular we conclude that

$$\begin{aligned} 4 &= 2n - 2|E| + 2|F| \leqslant 2n - 3|F| + 2|F| = 2n - |F| \text{ and} \\ 6 &= 3n - 3|E| + 3|F| \leqslant 3n - 3|E| + 2|E| = 3n - |E| , \end{aligned}$$

which yields the claimed bounds. □

It also follows that the degree of a "typical" vertex in a planar graph is a small constant. There exist several variations of this statement, a few more of which we will encounter during this course.

Corollary 2.6 *The average vertex degree in a simple planar graph is less than six.*

Exercise 2.7 *Prove Corollary 2.6.*

Exercise 2.8 *Show that neither* K_5 *(the complete graph on five vertices) nor* $K_{3,3}$ *(the complete bipartite graph where both classes have three vertices) is planar.*

Characterizing planarity. The classical theorems of Kuratowski and Wagner provide a characterization of planar graphs in terms of forbidden sub-structures. A *subdivision* of a graph $G = (V, E)$ is a graph that is obtained from G by replacing each edge with a path.

Theorem 2.9 (Kuratowski [16, 23]) *A graph is planar if and only if it does not contain a subdivision of* $K_{3,3}$ *or* K_5*.*

A *minor* of a graph $G = (V, E)$ is a graph that is obtained from G using zero or more edge contractions, edge deletions, and/or vertex deletions.

Theorem 2.10 (Wagner [26]) *A graph is planar if and only if it does not contain* $K_{3,3}$ *or* K_5 *as a minor.*

In some sense, Wagner's Theorem is a special instance[1] of a much more general theorem.

Theorem 2.11 (Graph Minor Theorem, Robertson/Seymour [21]) *Every minor-closed family of graphs can be described in terms of a finite set of forbidden minors.*

Being *minor-closed* means that for every graph from the family also all of its minors belong to the family. For instance, the family of planar graphs is minor-closed because planarity is preserved under removal of edges and vertices and under edge contractions. The Graph Minor Theorem is a celebrated result that was established by Robertson and Seymour in a series of twenty papers, see also the survey by Lovász [17]. They also described an $O(n^3)$ algorithm (with horrendous constants, though) to decide whether a graph on n vertices contains a fixed (constant-size) minor. Later, Kawarabayashi et al. [14] showed that this problem can be solved in $O(n^2)$ time. As a consequence, every minor-closed property can be decided in polynomial time.

Unfortunately, the result is non-constructive in the sense that in general we do not know how to obtain the set of forbidden minors for a given family/property. For instance, for the family of toroidal graphs (graphs that can be embedded without crossings on the torus) more than 16′000 forbidden minors are known, and we do not know how many there are in total. So while we know that there exists a quadratic time algorithm to test membership for minor-closed families, we have no idea what such an algorithm looks like in general.

[1] Strictly speaking, it is more than just a special instance because it also specifies the forbidden minors explicitly.

Graph families other than planar graphs for which the forbidden minors are known include forests (K_3) and outerplanar graphs ($K_{2,3}$ and K_4). A graph is *outerplanar* if it admits a plane drawing such that all vertices appear on the outer face (Figure 2.6).

Figure 2.6: *An outerplanar graph (left) and a plane drawing of it in which all vertices are incident to the outer face (right).*

Exercise 2.12 *(a) Give an example of a 6-connected planar graph or argue that no such graph exists.*

(b) Give an example of a 5-connected planar graph or argue that no such graph exists.

(c) Give an example of a 3-connected outerplanar graph or argue that no such graph exists.

Planarity testing. For planar graphs we do not have to contend ourselves with a cubic-time algorithm, as there are several approaches to solve the problem in linear time. In fact, there is quite a number of papers that describe different linear time algorithms, all of which—from a very high-level point of view—can be regarded as an annotated depth-first-search. The first such algorithm was described by Hopcroft and Tarjan [13], while the current state-of-the-art [29] is probably among the "path searching" method by Boyer and Myrwold [4] and the "LR-partition" method by de Fraysseix et al [10]. Although the overall idea in all these approaches is easy to convey, there are many technical details, which make a in-depth discussion rather painful to go through.

2.2 Graph representations

There are two standard representations for an abstract graph $G = (V, E)$ on $n = |V|$ vertices. For the *adjacency matrix* representation we consider the vertices to be ordered as $V = \{v_1, \ldots, v_n\}$. The adjacency matrix of an undirected graph is a symmetric $n \times n$-matrix $A = (a_{ij})_{1 \leqslant i,j \leqslant n}$ where $a_{ij} = a_{ji} = 1$, if $\{i, j\} \in E$, and $a_{ij} = a_{ji} = 0$, otherwise. Storing such a matrix explicitly requires $\Omega(n^2)$ space, and allows to test in constant time whether or not two given vertices are adjacent.

In an *adjacency list* representation, we store for each vertex a list of its neighbors in G. This requires only $O(n+|E|)$ storage, which is better than for the adjacency matrix in case that $|E| = o(n^2)$. On the other hand, the adjacency test for two given vertices is not

a constant-time operation, because it requires a search in one of the lists. Depending on the representation of these lists, such a search takes $O(d)$ time (unsorted list) or $O(\log d)$ time (sorted random-access representation, such as a balanced search tree), where d is the minimum degree of the two vertices.

Both representations have their merits. The choice of which one to use (if any) typically depends on what one wants to do with the graph. When dealing with embedded graphs, however, additional information concerning the embedding is needed beyond the pure incidence structure of the graph. The next section discusses a standard data structure to represent embedded graphs.

2.2.1 The Doubly-Connected Edge List

The *doubly-connected edge list* (DCEL) is a data structure to represent a plane graph in such a way that it is easy to traverse and to manipulate. In order to avoid unnecessary complications, let us discuss only connected graphs here that contain at least two vertices. It is not hard to extend the data structure to cover all plane graphs. For simplicity we also assume that we deal with a straight-line embedding and so the geometry of edges is defined by the mapping of their endpoints already. For more general embeddings, the geometric description of edges has to be stored in addition.

The main building block of a DCEL is a list of *halfedges*. Every actual edge is represented by two halfedges going in opposite direction, and these are called *twins*, see Figure 2.7. Along the boundary of each face, halfedges are oriented counterclockwise.

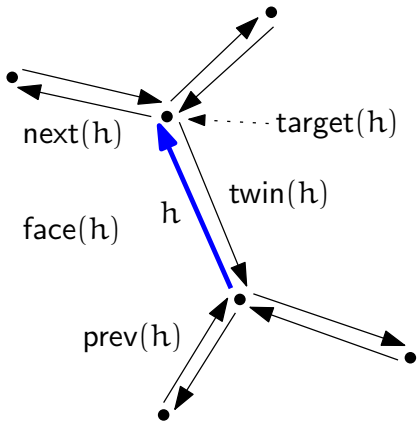


Figure 2.7: *A halfedge in a DCEL.*

A DCEL stores a list of halfedges, a list of vertices, and a list of faces. These lists are unordered but interconnected by various pointers. A vertex v stores a pointer $\mathsf{halfedge}(v)$ to an arbitrary halfedge originating from v. Every vertex also knows its coordinates, that is, the point $\mathsf{point}(v)$ it is mapped to in the represented embedding. A face f stores a pointer $\mathsf{halfedge}(f)$ to an arbitrary halfedge within the face. A halfedge h stores *five* pointers:

- a pointer target(h) to its target vertex,
- a pointer face(h) to the incident face,
- a pointer twin(h) to its twin halfedge,
- a pointer next(h) to the halfedge following h along the boundary of face(h), and
- a pointer prev(h) to the halfedge preceding h along the boundary of face(h).

A constant amount of information is stored for every vertex, (half-)edge, and face of the graph. Therefore the whole DCEL needs storage proportional to $|V| + |E| + |F|$, which is $O(n)$ for a plane graph with n vertices by Corollary 2.5.

This information is sufficient for most tasks. For example, traversing all edges around a face f can be done as follows:

```
s ← halfedge(f)
h ← s
do
    something with h
    h ← next(h)
while h ≠ s
```

Exercise 2.13 *Give pseudocode to traverse all edges incident to a given vertex v of a DCEL.*

Exercise 2.14 *Why is the previous halfedge* prev($\cdot$) *stored explicitly and the source vertex of a halfedge is not?*

2.2.2 Manipulating a DCEL

In many applications, plane graphs appear not just as static objects but rather they evolve over the course of an algorithm. Therefore the data structure used to represent the graph must allow for efficient update operations to change it.

First of all, we need to be able to generate new vertices, edges, and faces, to be added to the corresponding list within the DCEL and—symmetrically—the ability to delete an existing entity. Then it should be easy to add a new vertex v to the graph within some face f. As we maintain a connected graph, we better link the new vertex to somewhere, say, to an existing vertex u. For such a connection to be possible, we require that the open line segment uv lies completely in f.

Of course, two halfedges are to be added connecting u and v. But where exactly? Given that from a vertex and from a face only some arbitrary halfedge is directly accessible, it turns out convenient to use a halfedge in the interface. Let h denote the halfedge incident to f for which target(h) $= u$. Our operation then becomes (see also Figure 2.8)

add-vertex-at(v, h)
Precondition: the open line segment $\overline{\text{point}(v)\text{point}(u)}$, where $u := \text{target}(h)$, lies completely in $f := \text{face}(h)$.
Postcondition: a new vertex v has been inserted into f, connected by an edge to u.

(a) before (b) after

Figure 2.8: *Add a new vertex connected to an existing vertex* u.

and it can be realized by manipulating a constant number of pointers as follows.

add-vertex-at(v, h) {
$h_1 \leftarrow$ a new halfedge
$h_2 \leftarrow$ a new halfedge
$\text{halfedge}(v) \leftarrow h_2$
$\text{twin}(h_1) \leftarrow h_2$
$\text{twin}(h_2) \leftarrow h_1$
$\text{target}(h_1) \leftarrow v$
$\text{target}(h_2) \leftarrow u$
$\text{face}(h_1) \leftarrow f$
$\text{face}(h_2) \leftarrow f$
$\text{next}(h_1) \leftarrow h_2$
$\text{next}(h_2) \leftarrow \text{next}(h)$
$\text{prev}(h_1) \leftarrow h$
$\text{prev}(h_2) \leftarrow h_1$
$\text{next}(h) \leftarrow h_1$
$\text{prev}(\text{next}(h_2)) \leftarrow h_2$
}

Similarly, it should be possible to add an edge between two existing vertices u and v, provided the open line segment uv lies completely within a face f of the graph, see Figure 2.9. Since such an edge insertion splits f into two faces, the operation is called *split-face*. Again we use the halfedge h that is incident to f and for which $\text{target}(h) = u$. Our operation becomes then

split-face(h, v)

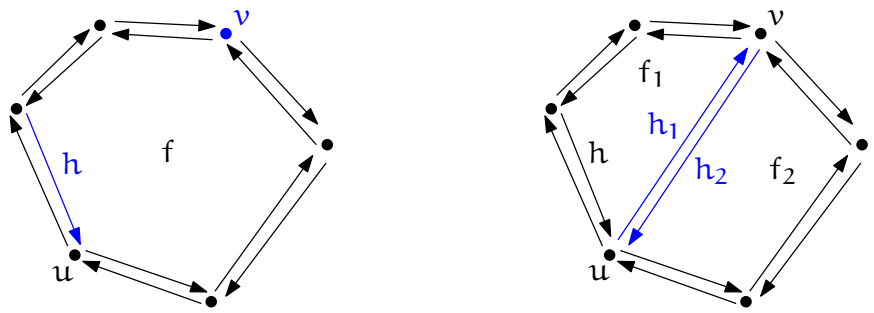


Figure 2.9: *Split a face by an edge* uv.

Precondition: v is incident to $f := \text{face}(h)$ but not adjacent to $u := \text{target}(h)$.
The open line segment $\overline{\text{point}(v)\text{point}(u)}$ lies completely in f.
Postcondition: f has been split by a new edge uv.

The implementation is slightly more complicated compared to add-vertex-at above, because the face f is destroyed and so we have to update the face information of all incident halfedges. In particular, this is not a constant time operation, but its time complexity is proportional to the size of f.

```
split-face(h, v) {
    f1 ← a new face
    f2 ← a new face
    h1 ← a new halfedge
    h2 ← a new halfedge
    halfedge(f1) ← h1
    halfedge(f2) ← h2
    twin(h1) ← h2
    twin(h2) ← h1
    target(h1) ← v
    target(h2) ← u
    next(h2) ← next(h)
    prev(next(h2)) ← h2
    prev(h1) ← h
    next(h) ← h1
    i ← h2
    loop
        face(i) ← f2
        if target(i) = v break the loop
        i ← next(i)
    endloop
    next(h1) ← next(i)
```

```
        prev(next(h1)) ← h1
        next(i) ← h2
        prev(h2) ← i
        i ← h1
        do
            face(i) ← f1
            i ← next(i)
        until target(i) = u
        delete the face f
    }
```

In a similar fashion one can realize the inverse operation join-face(h) that removes the edge (represented by the halfedge) h, thereby joining the faces face(h) and face(twin(h)).

It is easy to see that every connected plane graph on at least two vertices can be constructed using the operations add-vertex-at and split-face, starting from an embedding of K_2 (two vertices connected by an edge).

Exercise 2.15 *Give pseudocode for the operation join-face(h). Also specify preconditions, if needed.*

Exercise 2.16 *Give pseudocode for the operation split-edge(h), that splits the edge (represented by the halfedge) h into two by a new vertex w, see Figure 2.10.*

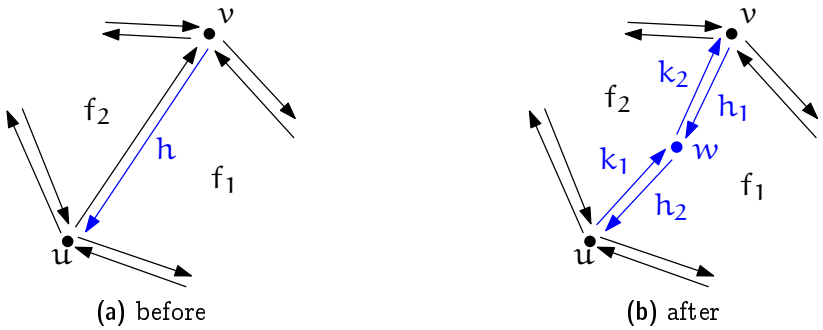


Figure 2.10: *Split an edge by a new vertex.*

2.2.3 Graphs with unbounded edges

In some cases it is convenient to consider plane graphs, in which some edges are not mapped to a line segment but to an unbounded curve, such as a ray. This setting is not really much different from the one we studied before, except that one vertex is placed "at infinity". One way to think of it is in terms of *stereographic projection* (see the proof of Theorem 2.2). The further away a point in $\mathbb{R}^2$ is from the origin, the closer its image on the sphere S gets to the north pole n of S. But there is no way to reach n except in the

limit. Therefore, we can imagine drawing the graph on S instead of in $\mathbb{R}^2$ and putting the "infinite vertex" at n.

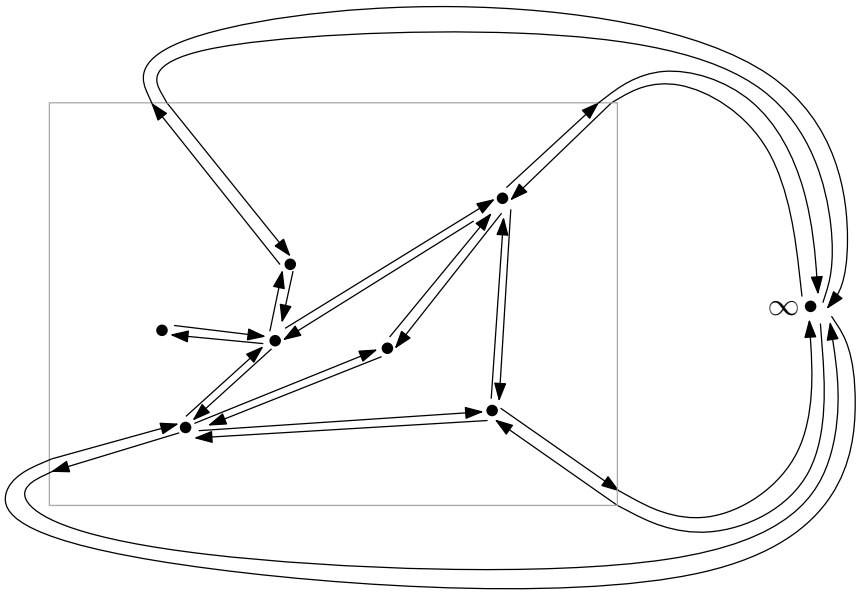


Figure 2.11: *A DCEL with unbounded edges. Usually, we will not show the infinite vertex and draw all edges as straight-line segments. This yields a geometric drawing, like the one within the gray box.*

All this is just for the sake of a proper geometric interpretation. As far as a DCEL representation of such a graph is concerned, there is no need to consider spheres or, in fact, anything beyond what we have discussed before. The only difference to the case with all finite edges is that there is this special infinite vertex, which does not have any point/coordinates associated to it. But other than that, the infinite vertex is treated in exactly the same way as the finite vertices: it has in– and outgoing halfedges along which the unbounded faces can be traversed (Figure 2.11).

Remarks. It is actually not so easy to point exactly to where the DCEL data structure originates from. Often Muller and Preparata [19] are credited, but while they use the term DCEL, the data structure they describe is different from what we discussed above and from what people usually consider a DCEL nowadays. Overall, there are a large number of variants of this data structure, which appear under the names *winged edge* data structure [2], *halfedge* data structure [27], or *quad-edge* data structure [12]. Kettner [15] provides a comparison of all these and some additional references.

2.2.4 Combinatorial embeddings

The DCEL data structure discussed in the previous section provides a fully fleshed-out representation of what is called a *combinatorial embedding*. From a mathematical point

of view this can be regarded an equivalence relation on embeddings: Two embeddings are equivalent if their face boundaries—regarded as circular sequences of edges (or vertices) in counterclockwise order—are the same (as sets) up to a global change of orientation (reversing the order of all sequences simultaneously). For instance, the faces of the plane graphs shown in Figure 2.12a are (described as a list of vertices)

$$\begin{aligned}
&\text{(a)} \;:\; \{(1,2,3),(1,3,6,4,5,4),(1,4,6,3,2)\},\\
&\text{(b)} \;:\; \{(1,2,3,6,4,5,4),(1,3,2),(1,4,6,3)\}, \text{and}\\
&\text{(c)} \;:\; \{(1,4,5,4,6,3),(1,3,2),(1,2,3,6,4)\}.
\end{aligned}$$

Note that a vertex can appear several times along the boundary of a face (if it is a cut-vertex). Clearly (b) is not equivalent to (a) nor (c), because it is the only graph that contains a face bounded by seven vertices. However, (a) and (c) turn out to be equivalent: after reverting orientations f_1 takes the role of h_2, f_2 takes the role of h_1, and f_3 takes the role of h_3.

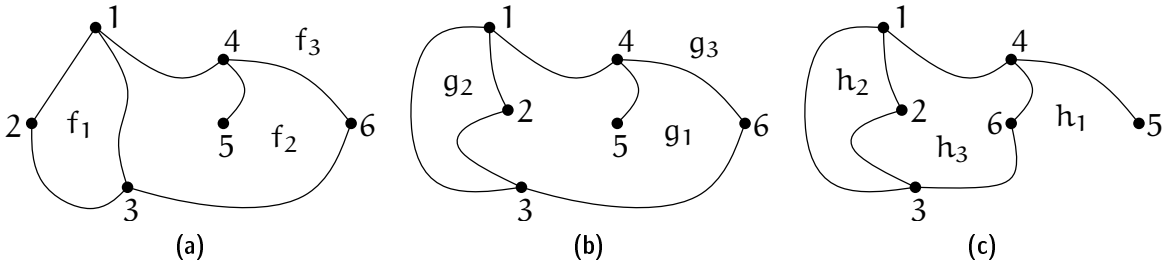


Figure 2.12: *Equivalent embeddings?*

In a dual interpretation one can just as well define equivalence in terms of the cyclic order of neighbors around all vertices. In this form, a compact way to describe a combinatorial embedding is as a so-called *rotation system* that consists of a permutation π and an involution ρ, both of which are defined on the set of halfedges (in this context often called *darts* or *flags*) of the embedding. The orbits of π correspond to the vertices, as they iterate over the incident halfedges. The involution ρ maps each halfedge to its twin.

Many people prefer this dual view, because one does not have to discuss the issue of vertices or edges that appear several times on the boundary of a face. The following lemma shows that such an issue does not arise when dealing with biconnected graphs.

Lemma 2.17 *In a biconnected plane graph every face is bounded by a cycle.*

We leave the proof as an exercise. Intuitively the statement is probably clear. But we believe it is instructive to think about how to make a formal argument. An easy consequence is the following corollary, whose proof we also leave as an exercise.

Corollary 2.18 *In a 3-connected plane graph the neighbors of a vertex lie on a cycle.*

Note that the statement does not read "form a cycle" but rather "lie on a cycle".

Exercise 2.19 *Prove Lemma 2.17 and Corollary 2.18.*

2.3 Unique embeddings

We have seen in Lemma 2.17 that all faces in biconnected plane graphs are bounded by cycles. Conversely one might wonder which cycles of a planar graph G bound a face in *some* plane embedding of G. Such a cycle is called a *facial cycle* (Figure 2.13).

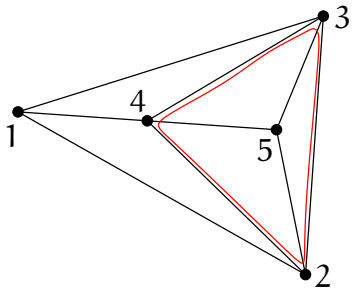


Figure 2.13: *The cycle* $(1,2,3)$ *is facial and we can show that* $(2,3,4)$ *is not.*

In fact, we will look at a slightly different class of cycles, namely those that bound a face in *every* plane embedding of G. The lemma below provides a complete characterization of those cycles. In order to state it, let us introduce a bit more terminology. A *chord* of a cycle C in a graph G is an edge that connects two vertices of C but is not an edge of C. A cycle C in a graph G is an *induced cycle*, if $C = G[V(C)]$, that is, C does not have any chord in G.

Lemma 2.20 *Let* C *be a cycle in a planar graph* G *such that* $G \neq C$ *and* G *is not* C *plus a single chord of* C*. Then* C *bounds a face in every plane embedding of* G *if and only if* C *is an induced cycle and it is not separating (i.e.,* $G \setminus C$ *is connected).*

Proof. "$\Leftarrow$": Consider any plane embedding Γ of G. As $G \setminus C$ is connected, by the Jordan Curve Theorem it is contained either in the interior of C or in the exterior of C in Γ. In either case, the other component of the plane is bounded by C, because there are no edges among the vertices of C.

"$\Rightarrow$": Using contraposition, suppose that C is not induced or $G \setminus C$ is disconnected. We have to show that there exists a plane embedding of G in which C does not bound a face.

If C is not induced, then there is a chord c of C in G. As $G \neq C \cup c$, either G has a vertex v that is not in C or G contains another chord $d \neq c$ of C. In either case, consider any plane embedding Γ of G in which C bounds a face. (If such an embedding does not exist, there is nothing to show.) We can modify Γ by drawing the chord c in the face bounded by C to obtain an embedding Γ' of G in which C does not bound a face: one of

the two regions bounded by C according to the Jordan Curve Theorem contains c and the other contains either the vertex v or the other chord d.

If $G \setminus C$ contains two components A and B, then consider a plane embedding Γ of G. If C is not a face in Γ, there is nothing to show. Hence suppose that C is a face of Γ (Figure 2.14a). From Γ we obtain induced plane embeddings Γ_A of $G \setminus B = A \cup C$ and Γ_B of $G \setminus A = B \cup C$. Using Theorem 2.2 we may suppose that C bounds the outer face in Γ_A and it does not bound the outer face in Γ_B. Then we can glue both embeddings at C, that is, extend Γ_B to an embedding of G by adding Γ_A within the face bounded by C (Figure 2.14b). The resulting embedding is a plane drawing of G in which C does not bound a face.

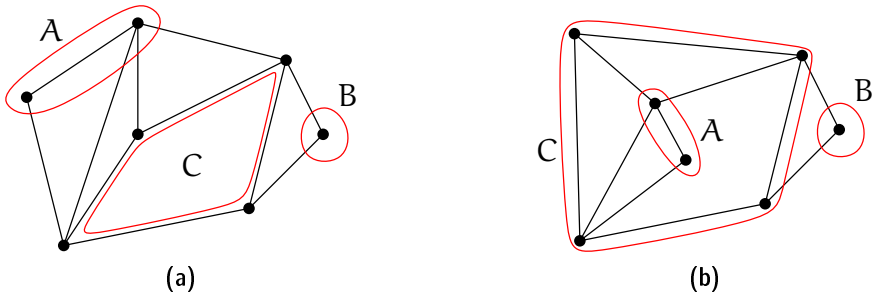


Figure 2.14: *Construct a plane embedding of* G *in which* C *does not bound a face.*

Finally, consider the case that $G \setminus C = \emptyset$ (which is not a connected graph according to our definition). As we considered above the case that C is not an induced cycle, the only remaining case is $G = C$, which is excluded explicitly. □

For both special cases for G that are excluded in Lemma 2.20 it is easy to see that all cycles in G bound a face in every plane embedding. This completes the characterization. Also observe that in these special cases G is not 3-connected.

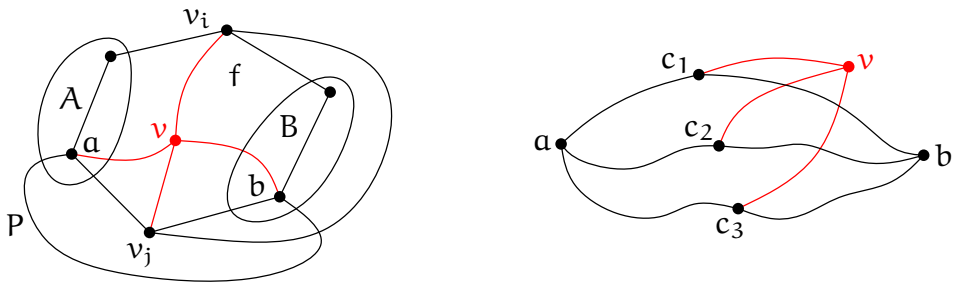
Corollary 2.21 *A cycle* C *of a* 3*-connected planar graph* G *bounds a face in every plane embedding of* G *if and only if* C *is an induced cycle and it is not separating.* □

The following theorem tells us that for a wide range of graphs we have little choice as far as a plane embedding is concerned, at least from a combinatorial point of view. Geometrically, there is still a lot of freedom, though.

Theorem 2.22 (Whitney [28]) *A* 3*-connected planar graph has a unique combinatorial plane embedding (up to equivalence).*

Proof. Let G be a 3-connected planar graph and suppose there exist two embeddings Φ_1 and Φ_2 of G that are not equivalent. That is, there is a cycle $C = (v_1, \ldots, v_k)$, $k \geqslant 3$, in G that bounds a face in, say, Φ_1 but C does not bound a face in Φ_2. By Corollary 2.21 such a cycle has a chord or it is separating. We consider both options.

Case 1: C has a chord $\{v_i, v_j\}$, with $j \geqslant i+2$. Denote $A = \{v_x \mid i < x < j\}$ and $B = \{v_x \mid x < i \vee j < x\}$ and observe that both A and B are non-empty (because $\{v_i, v_j\}$ is a chord and so v_i and v_j are not adjacent in C). Given that G is 3-connected, there is at least one path P from A to B that does not use either of v_i or v_j. Let a denote the last vertex of P that is in A, and let b denote the first vertex of B that is in b. As C bounds a face f in Φ_1, we can add a new vertex v inside the face bounded by C and connect v by four pairwise internally disjoint curves to each of v_i, v_j, a, and b. The result is a plane graph $G' \supset G$ that contains a subdivision of K_5 with branch vertices v, v_i, v_j, a, and b. By Kuratowski's Theorem (Theorem 2.9) this contradicts the planarity of G'.



(a) Case 1. (b) Case 2.

Figure 2.15: *Illustration of the two cases in Theorem 2.22.*

Case 2: C is separating and, therefore, $G \setminus C$ contains two distinct components A and B. (We have $G \neq C$ because G is 3-connected.) Consider now the embedding Φ_1 in which C bounds a face, without loss of generality (Theorem 2.2) a bounded face f. Hence both A and B are embedded in the exterior of f.

Choose vertices $a \in A$ and $b \in B$ arbitrarily. As G is 3-connected, by Menger's Theorem (Theorem 1.2), there are at least three pairwise internally vertex-disjoint paths from a to b. Fix three such paths $\alpha_1, \alpha_2, \alpha_3$ and denote by c_i the first point of α_i that is on C, for $1 \leqslant i \leqslant 3$. Note that c_1, c_2, c_3 are well defined, because C separates A and B, and they are pairwise distinct. Therefore, $\{a, b\}$ and $\{c_1, c_2, c_3\}$ are branch vertices of a $K_{2,3}$ subdivision in G. We can add a new vertex v inside the face bounded by C and connect v by three pairwise internally disjoint curves to each of c_1, c_2, and c_3. The result is a plane graph $G' \supset G$ that contains a $K_{3,3}$ subdivision. By Kuratowski's Theorem (Theorem 2.9) this contradicts the planarity of G'.

In both cases we arrived at a contradiction and so there does not exist such a cycle C. Thus Φ_1 and Φ_2 are equivalent. □

Whitney's Theorem does not provide a characterization of unique embeddability, because there are both biconnected graphs that have a unique plane embedding (such as cycles) and biconnected graphs that admit several non-equivalent plane embeddings (for instance, a triangulated pentagon).