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.

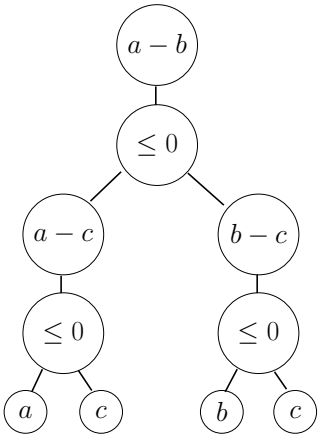
On the positive side, the real RAM model allows to abstract from the lowlands of numeric and algebraic computation and to concentrate on the algorithmic core from a combinatorial point of view.

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

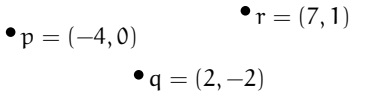
Output: Is $x_i = x_j$, for some $i, j \in \{1, \ldots, n\}$ with $i \neq j$?

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

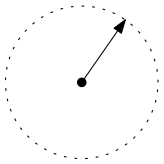
1.2 Basic Geometric Objects

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.

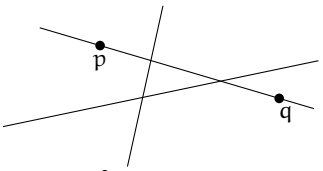
Points. A point p, typically described by its d Cartesian coordinates $p = (x_1, \ldots, x_d)$.



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 $\|v\| = \sqrt{\sum_{i=1}^{d} {x_i}^2} = 1$.

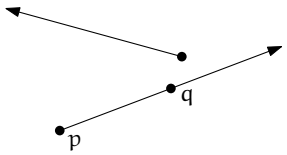


Lines. A line is a one-dimensional affine subspace. It can be described by two distinct points p and q as the set of all points r that satisfy $r = p + \lambda(q - p)$, for some $\lambda \in \mathbb{R}$.

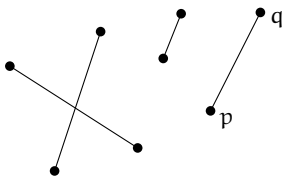


While any pair of distinct points defines a unique line, a line in $\mathbb{R}^2$ contains infinitely many points and so it may happen that a collection of three or more points lie on a line. Such a collection of points is termed *collinear*[3].

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.*

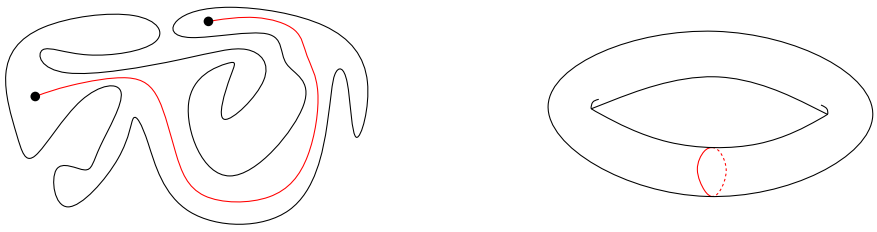


Figure 2.1: *A Jordan curve and two points in one of its faces (left); a simple closed curve that does not disconnect the torus (right).*

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).

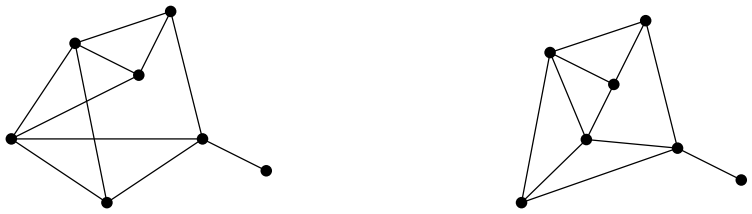


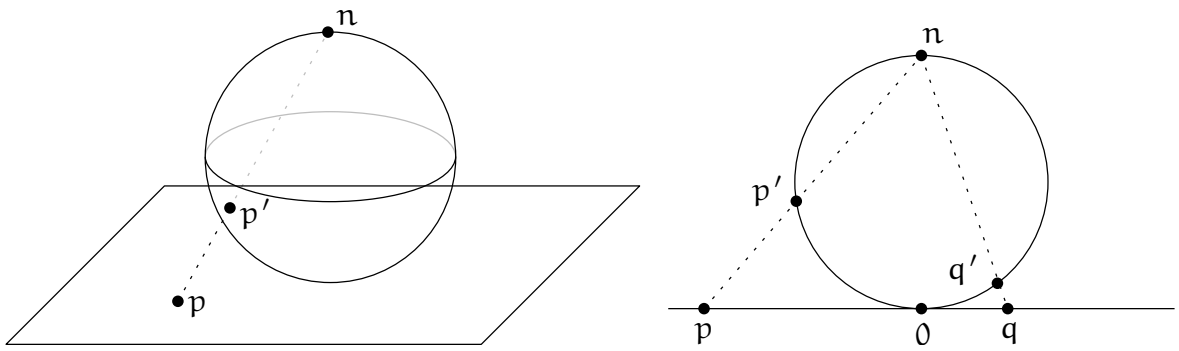
Figure 2.2: *A planar graph (left) and a plane drawing of it (right).*

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.*

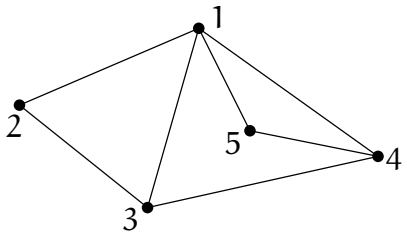


Figure 2.4: *Plane embedding of* G.

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.

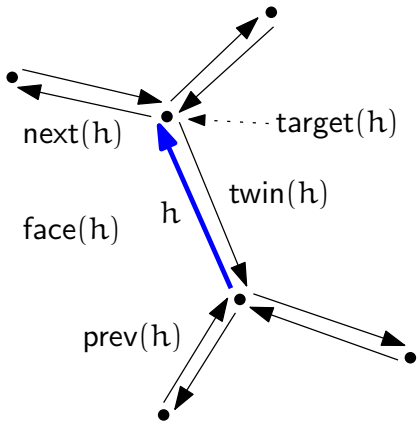


Figure 2.7: *A halfedge in a DCEL.*

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)

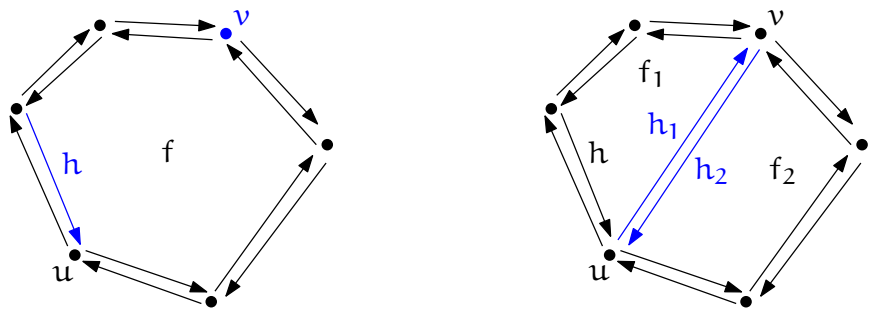


Figure 2.9: *Split a face by an edge* uv.

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.*

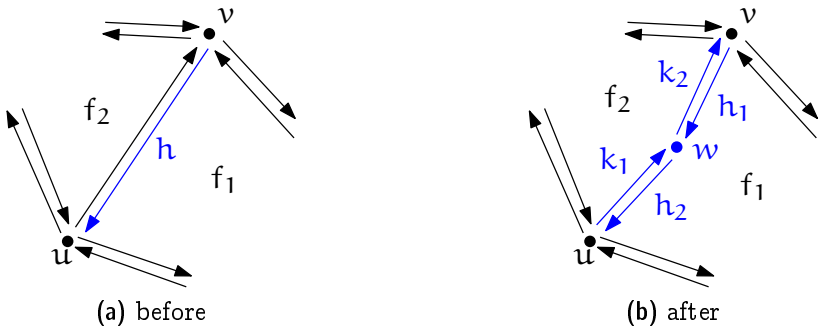


Figure 2.10: *Split an edge by a new vertex.*

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.

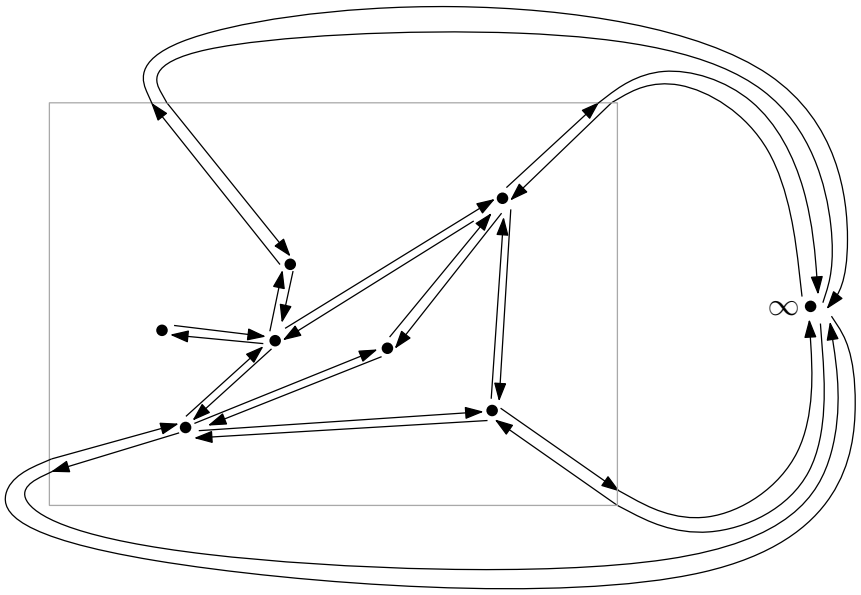


Figure 2.11: *A DCEL with unbounded edges. Usually, we will not show the infinite vertex and draw all edges as straight-line segments. This yields a geometric drawing, like the one within the gray box.*

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.

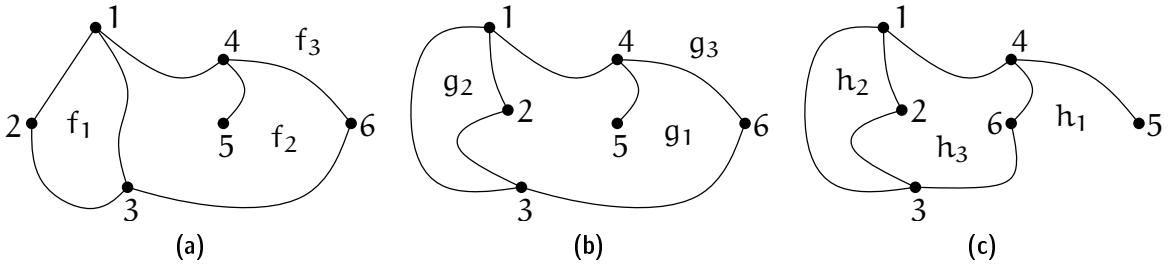


Figure 2.12: *Equivalent embeddings?*

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).

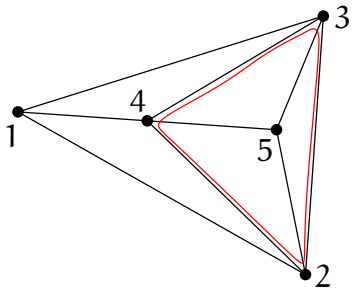


Figure 2.13: *The cycle* $(1,2,3)$ *is facial and we can show that* $(2,3,4)$ *is not.*

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.

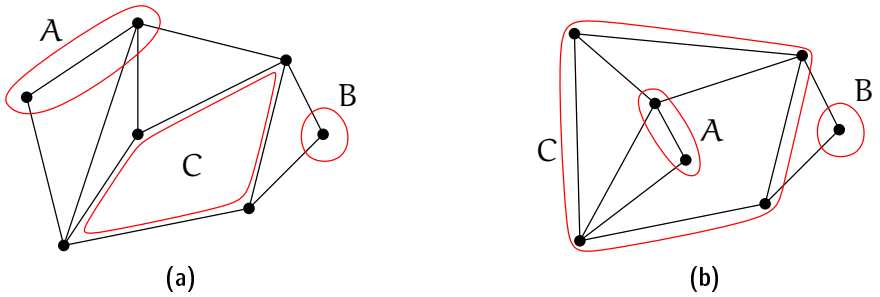


Figure 2.14: *Construct a plane embedding of* G *in which* C *does not bound a face.*

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.

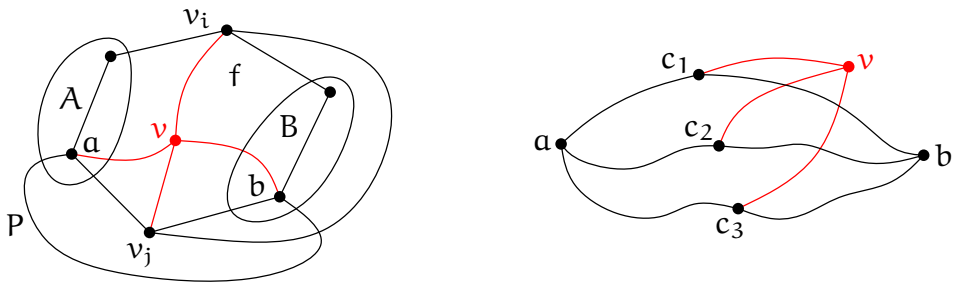
Corollary 2.21 *A cycle* C *of a* 3*-connected planar graph* G *bounds a face in every plane embedding of* G *if and only if* C *is an induced cycle and it is not separating.* □

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).

2.4 Triangulating a plane graph

A large and important class of 3-connected graphs is formed by the maximal planar graphs. A graph is *maximal planar* if no edge can be added so that the resulting graph is still planar.

Lemma 2.23 *A maximal planar graph on $n \geqslant 3$ vertices is biconnected.*

Proof. Consider a maximal planar graph $G = (V, E)$. If G is not biconnected, then it has a cut-vertex v. Take a plane drawing Γ of G. As $G \setminus v$ is disconnected, removal of v also splits $N_G(v)$ into at least two components. Therefore, there are two vertices $a, b \in N_G(v)$ that are adjacent in the circular order of vertices around v in Γ and are in different components of $G \setminus v$. In particular, $\{a, b\} \notin E$ and we can add this edge to G (routing it very close to the path (a, v, b) in Γ) without violating planarity. This is in contradiction to G being maximal planar and so G is biconnected. □

Lemma 2.24 *In a maximal planar graph on $n \geqslant 3$ vertices, all faces are topological triangles, that is, each is bounded by exactly three edges.*

Proof. Consider a maximal planar graph $G = (V, E)$ and a plane drawing Γ of G. By Lemma 2.23 we know that G is biconnected and so by Lemma 2.17 every face of Γ is bounded by a cycle. Suppose that there is a face f in Γ that is bounded by a cycle $v_0, \ldots, v_{k-1}$ of $k \geqslant 4$ vertices. We claim that at least one of the edges $\{v_0, v_2\}$ or $\{v_1, v_3\}$ is not present in G.

Suppose to the contrary that $\{\{v_0, v_2\}, \{v_1, v_3\}\} \subseteq E$. Then we can add a new vertex v' in the interior of f and connect v' inside f to all of v_0, v_1, v_2, v_3 by an edge (curve) without introducing a crossing. In other words, given that G is planar, also the graph $G' = (V \cup \{v'\}, E \cup \{\{v_i, v'\} \mid i \in \{0, 1, 2, 3\}\})$ is planar. However, v_0, v_1, v_2, v_3, v' are branch vertices of a K_5 subdivision in G': v' is connected to all other vertices within f, along the boundary ∂f of f each vertex v_i is connected to both $v_{(i-1) \bmod 4}$ and $v_{(i+1) \bmod 4}$ and the missing two connections are provided by the edges $\{v_0, v_2\}$ and $\{v_1, v_3\}$ (Figure 2.16a). By Kuratowski's Theorem this is in contradiction to G' being planar. Therefore, one of the edges $\{v_0, v_2\}$ or $\{v_1, v_3\}$ is not present in G, as claimed.

So suppose without loss of generality that $\{v_1, v_3\} \notin E$. But then we can add this edge (curve) within f to Γ without introducing a crossing (Figure 2.16b). It follows that the edge $\{v_1, v_3\}$ can be added to G without sacrificing planarity, which is in contradiction to G being maximal planar. Therefore, there is no such face f bounded by four or more vertices. □

The proof of Lemma 2.24 also contains the idea for an algorithm to *topologically triangulate* a plane graph.

Theorem 2.25 *For a given connected plane graph $G = (V, E)$ on n vertices one can compute in $O(n)$ time and space a maximal plane graph $G' = (V, E')$ with $E \subseteq E'$.*

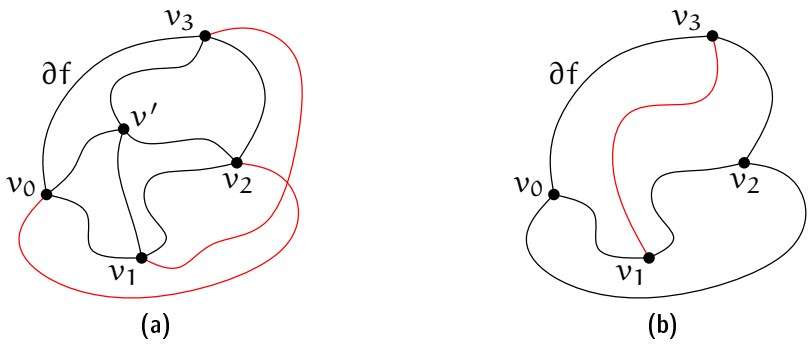


Figure 2.16: *Every face of a maximal planar graph is a topological triangle.*

Proof. Suppose, for instance, that G is represented as a DCEL[2], from which one can easily extract the face boundaries. If some vertex v appears several times along the boundary of a single face, it is a cut-vertex. We fix this by adding an edge between the two neighbors of all but the first occurrence of v. This can easily be done in linear time by maintaining a counter for each vertex on the face boundary. The total number of edges and vertices along the boundary of all faces is proportional to the number of edges in G, which by Corollary 2.5 is linear. Hence we may suppose that all faces of G are bounded by a cycle.

For each face f that is bounded by more than three vertices, select a vertex v_f on its boundary and store with each vertex all faces that select it. Then process each vertex v as follows: First mark all neighbors of v in G. Then process all faces that selected v. For each such face f with $v_f = v$ iterate over the boundary $\partial f = (v, v_1, \dots, v_k)$, where $k \geqslant 3$, of f to test whether there is any marked vertex other than the two neighbors v_1 and v_k of v along ∂f.

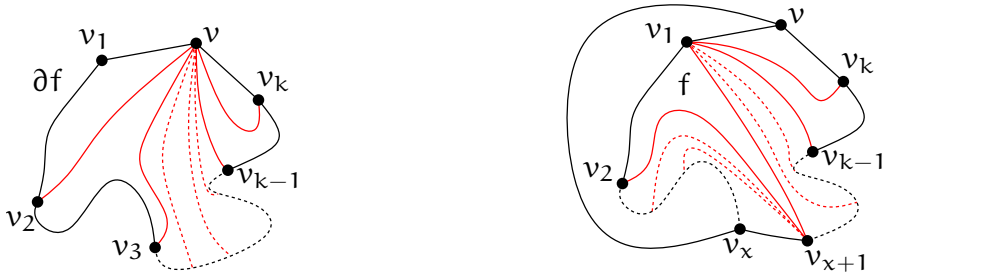
If there is no such vertex, we can safely triangulate f using a star from v, that is, by adding the edges $\{v, v_i\}$, for $i \in \{2, \dots, k-1\}$ (Figure 2.17a).

Otherwise, let v_x be the first marked vertex in the sequence $v_2, \dots, v_{k-1}$. The edge $\{v, v_x\}$ that is embedded as a curve in the exterior of f prevents any vertex from $v_1, \dots, v_{x-1}$ from being connected by an edge in G to any vertex from $v_{x+1}, \dots, v_k$. (This is exactly the argument that we made in the proof of Lemma 2.24 above for the edges $\{v_0, v_2\}$ and $\{v_1, v_3\}$, see Figure 2.16a.) In particular, we can safely triangulate f using a bi-star from v_1 and v_{x+1}, that is, by adding the edges $\{v_1, v_i\}$, for $i \in \{x+1, \dots, k\}$, and $\{v_j, v_{x+1}\}$, for $j \in \{2, \dots, x-1\}$ (Figure 2.17b).

Finally, conclude the processing of v by removing all marks on its neighbors.

Regarding the runtime bound, note that every face is traversed a constant number of times. In this way, each edge is touched a constant number of times, which by Corollary 2.5 uses linear time overall. Similarly, the vertex marking is done at most twice (mark und unmark) per vertex. Therefore, the overall time needed can be bounded by

[2] If you wonder how the—possibly complicated—curves that correspond to edges are represented: they do not need to be, because here we need a representation of the combinatorial embedding only.



(a) Case 1: v does not have any neighbor on ∂f other than v_1 and v_k.

(b) Case 2: v has a neighbor v_x on ∂f other than v_1 and v_k.

Figure 2.17: *Topologically triangulating a plane graph.*

$\sum_{v \in V} \deg_G(v) = 2|E| = O(n)$ by the Handshaking Lemma and Corollary 2.5. □

Theorem 2.26 *A maximal planar graph on $n \geqslant 4$ vertices is 3-connected.*

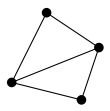
Exercise 2.27 *Prove Theorem 2.26.*

Using any of the standard planarity testing algorithms we can obtain a combinatorial embedding of a planar graph in linear time. Together with Theorem 2.25 this yields the following

Corollary 2.28 *For a given planar graph $G = (V, E)$ on n vertices one can compute in $O(n)$ time and space a maximal planar graph $G' = (V, E')$ with $E \subseteq E'$.* □

The results discussed in this section can serve as a tool to fix the combinatorial embedding for a given graph G: augment G using Theorem 2.25 to a maximal planar graph G′, whose combinatorial embedding is unique by Theorem 2.22.

Being maximal planar is a property of an abstract graph. In contrast, a geometric graph to which no straight-line edge can be added without introducing a crossing is called a *triangulation*. Not every triangulation is maximal planar, as the example depicted to the right shows.



It is also possible to triangulate a geometric graph in linear time. But this problem is much more involved. Triangulating a single face of a geometric graph amounts to what is called "triangulating a simple polygon". This can be done in near-linear[3] time using standard techniques, and in linear time using Chazelle's famous algorithm, whose description spans a fourty pages paper [6].

Exercise 2.29 *We discussed the DCEL structure to represent plane graphs in Section 2.2.1. An alternative way to represent an embedding of a maximal planar graph is the following: For each triangle, store references to its three vertices and*

[3] $O(n \log n)$ or—using more elaborate tools—$O(n \log^* n)$ time

to its three neighboring triangles. Compare both approaches. Discuss different scenarios where you would prefer one over the other. In particular, analyze the space requirements of both.

Connectivity serves as an important indicator for properties of planar graphs. Another example is the following famous theorem of Tutte that provides a sufficient condition for Hamiltonicity. Its proof is beyond the scope of our lecture.

Theorem 2.30 (Tutte [24]) *Every 4-connected planar graph is Hamiltonian.*

Moreover, for a given 4-connected planar graph a Hamiltonian cycle can also be computed in linear time [7].

2.5 Compact straight-line drawings

As a next step we consider plane embeddings in the geometric setting, where every edge is drawn as a straight-line segment. A classical theorem of Wagner and Fáry states that this is not a restriction in terms of plane embeddability.

Theorem 2.31 (Fáry [9], Wagner [25]) *Every planar graph has a plane straight-line embedding (that is, it is isomorphic to a plane straight-line graph).*

This statement is quite surprising, considering how much more freedom arbitrarily complex Jordan arcs allow compared to line segments, which are completely determined by their endpoints. In order to further increase the level of appreciation, let us note that a similar "straightening" is not possible, when fixing the point set on which the vertices are to be embedded: For a given planar graph $G = (V, E)$ on n vertices and a given set $P \subset \mathbb{R}^2$ of n points, one can always find a plane embedding of G that maps V to P [20]. However, this is not possible in general with a plane *straight-line* embedding. For instance, K_4 does not admit a plane straight-line embedding onto a set of points that form a convex quadrilateral, such as a rectangle. In fact, it is NP-hard to decide whether a given planar graph admits a plane straight-line embedding onto a given point set [5].

Although Fáry-Wagner's theorem has a nice inductive proof, we will not discuss it here. Instead we will prove a stronger statement that implies Theorem 2.31.

A very nice property of straight-line embeddings is that they are easy to represent: We need to store points/coordinates for the vertices only. From an algorithmic and complexity point of view the space needed by such a representation is important, because it appears in the input and output size of algorithms that work on embedded graphs. While the Fáry-Wagner Theorem guarantees the existence of a plane straight-line embedding for every planar graph, it does not provide bounds on the size of the coordinates used in the representation. But the following strengthening provides such bounds, by describing an algorithm that embeds (without crossings) a given planar graph on a linear size integer grid.

Theorem 2.32 (de Fraysseix, Pach, Pollack [11]) *Every planar graph on $n \geqslant 3$ vertices has a plane straight-line drawing on the $(2n-3) \times (n-1)$ integer grid.*

Canonical orderings. The key concept behind the algorithm is the notion of a canonical ordering, which is a vertex order that allows to construct a plane drawing in a natural (hence canonical) way. Reading it backwards one may think of a shelling or peeling order that destructs the graph vertex by vertex from the outside. A canonical ordering also provides a succinct representation for the combinatorial embedding.

Definition 2.33 *A plane graph is* **internally triangulated** *if it is biconnected and every bounded face is a (topological) triangle. Let G be an internally triangulated plane graph and $C_\circ(G)$ its outer cycle. A permutation $\pi = (v_1, v_2, \ldots, v_n)$ of $V(G)$ is a* **canonical ordering** *for G, if*

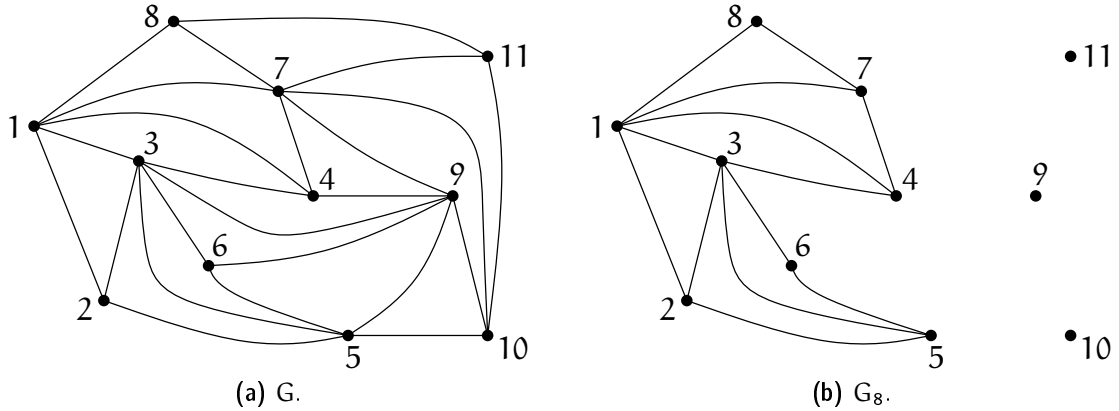
(1) G_k is internally triangulated, for all $k \in \{3, \ldots, n\}$;

(2) v_1v_2 is on the outer cycle $C_\circ(G_k)$ of G_k, for all $k \in \{3, \ldots, n\}$;

(3) v_{k+1} is located in the outer face of G_k and its neighbors appear consecutively along $C_\circ(G_k)$, for all $k \in \{3, \ldots, n-1\}$;

where G_k is the subgraph of G induced by $v_1, \ldots, v_k$.

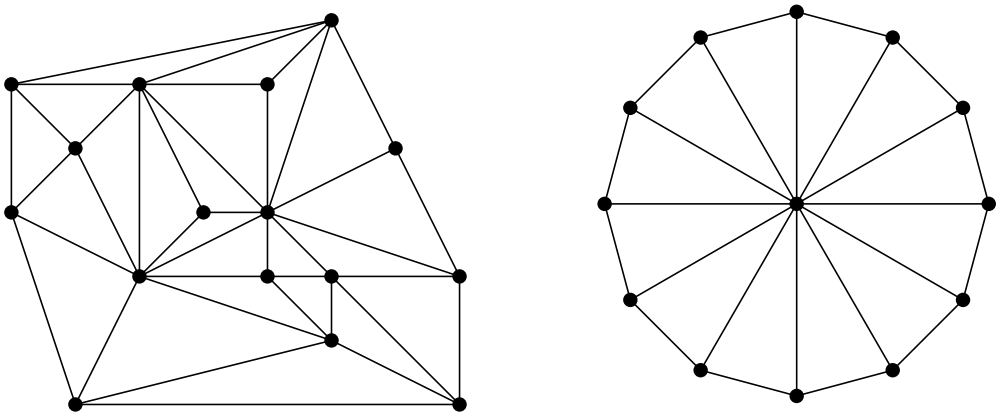
Figure 2.18 shows an example. Note that there are permutations that do not correspond to a canonical order: for instance, when choosing the vertex 4 as the eighth vertex instead of 8 in Figure 2.18b the graph G_8 is not biconnected (1 is a cut-vertex).



(a) G. (b) G_8.

Figure 2.18: *An internally triangulated plane graph and a canonical order for it.*

Exercise 2.34 *(a) Compute a canonical ordering for the following internally triangulated plane graphs:*



(b) Give a family of internally triangulated plane graphs G_n *on* $n = 2k$ *vertices with at least* $(n/2)!$ *canonical orderings.*

Exercise 2.35 *(a) Describe a plane graph* G *with* n *vertices that can be embedded (while preserving the outer face) on a grid of size* $(2n/3 - 1) \times (2n/3 - 1)$ *but not on a smaller grid.*

(b) Can you draw G *on a smaller grid if you are allowed to change the embedding?*

Theorem 2.36 *For every internally triangulated plane graph* G *and every edge* $\{v_1, v_2\}$ *on its outer face, there exists a canonical ordering for* G *that starts with* v_1, v_2*. Moreover, such an ordering can be computed in linear time.*

Proof. Induction on n, the number of vertices. For a triangle, any order suffices and so the statement holds. Hence consider an internally triangulated plane graph $G = (V, E)$ on $n \geqslant 4$ vertices. We claim that it is enough to select a vertex $v_n \notin \{v_1, v_2\}$ on $C_\circ(G)$ that is not incident to a chord of $C_\circ(G)$.

First observe that G is plane and $v_n \in C_\circ(G)$ and so all neighbors of v_n in G must appear on the outer face of $G_{n-1} = G \setminus \{v_n\}$. Consider the circular sequence of neighbors around v_n in G and break it into a linear sequence $u_1, \ldots, u_m$, for some $m \geqslant 2$, that starts and ends with the neighbors of v_n in $C_\circ(G)$. As G is internally triangulated, each of the bounded faces spanned by v_n, u_i, u_{i+1}, for $i \in \{i, \ldots, m-1\}$, is a triangle and hence $\{u_i, u_{i+1}\} \in E$. This implies *(3)* for $k = n$. Properties *(1)* and *(2)* hold trivially (by assumption) in that case. In order to complete the ordering inductively we need to show that G_{n-1} is also internally triangulated.

The way G_{n-1} is obtained from G, every bounded face f of G_{n-1} also appears as a bounded face of G. As G is internally triangulated, f is a triangle. It remains to show that G_{n-1} is biconnected. The outer cycle $C_\circ(G_{n-1})$ of G_{n-1} is obtained from $C_\circ(G)$ by removing v_n and replacing it with the (possibly empty) sequence $u_2, \ldots, u_{m-1}$. As v_n is not incident to a chord of $C_\circ(G)$ (and so neither of $u_2, \ldots, u_{m-1}$ appeared along

$C_\circ(G)$ already), the resulting sequence forms a cycle, indeed. Add a new vertex v in the outer face of G_{n-1} and connect v to every vertex of $C_\circ(G_{n-1})$ to obtain a maximal planar graph $H \supset G_{n-1}$. By Theorem 2.26 H is 3-connected and so G_{n-1} is biconnected, as desired. This also completes the proof of the initial claim.

It remains to show that we can always find such a vertex $v_n \notin \{v_1, v_2\}$ on $C_\circ(G)$ that is not incident to a chord of $C_\circ(G)$. If $C_\circ(G)$ does not have any chord, this is obvious, because every cycle has at least three vertices, one of which is neither v_1 nor v_2. So suppose that $C_\circ(G)$ has a chord c. The endpoints of c split $C_\circ(G)$ into two paths, one of which does not have v_1 nor v_2 as an internal vertex. Among all possible chords of $C_\circ(G)$ select c such that this path has minimal length. (It has always at least two edges, because there is always at least one vertex "behind" a chord.) Then by definition of c this path is an induced path in G and none of its (at least one) interior vertices is incident to a chord of $C_\circ(G)$, because such a chord would cross c. So we can select v_n from these vertices. By the way the path is selected with respect to c, this procedure does not select v_1 nor v_2.

Regarding the runtime bound, we maintain the following information for each vertex v: whether it has been chosen already, whether it is on the outer face of the current graph, and the number of incident chords with respect to the current outer cycle. Given a combinatorial embedding of G, it is straighforward to initialize this information in linear time. (Every edge is considered at most twice, once for each endpoint on the outer face.)

When removing a vertex, there are two cases: Either v_n has two neighbors u_1 and u_2 only (Figure 2.19a), in which case the edge u_1u_2 ceases to be a chord. Thus, the chord count for u_1 and u_2 has to be decremented by one. Otherwise, there are $m \geqslant 3$ neighbors $u_1, \ldots, n_m$ (Figure 2.19b) and (1) all vertices $u_2, \ldots, u_{m-1}$ are new on the outer cycle, and (2) every edge incident to u_i, for $i \in \{2, \ldots, k-1\}$, and some other vertex on the outer cycle other than u_{i-1} or u_{i+1} is a new chord (and the corresponding counters at the endpoints have to by incremented by one).

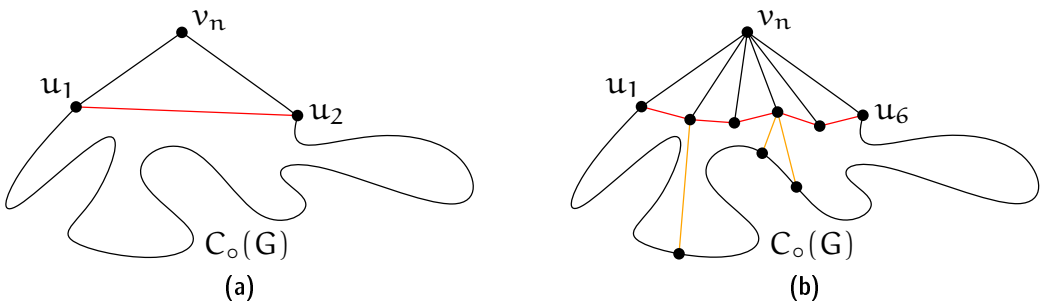


Figure 2.19: *Processing a vertex when computing a canonical ordering.*

During the course of the algorithm every vertex appears once as a new vertex on the outer face. At this point all incident edges are examined. Overall, every edge is inspected at most twice—once for each endpoint—which takes linear time by Corollary 2.5. □

Using one of the linear time planarity testing algorithms, we can obtain a combinatorial embedding for a given maximal planar graph G. As every maximal plane graph is internally triangulated, we can then use Theorem 2.36 to provide us with a canonical ordering for G, in overall linear time.

Corollary 2.37 *Every maximal planar graph admits a canonical ordering. Moreover, such an ordering can be computed in linear time.* □

As simple as they may appear, canonical orderings are a powerful and versatile tool to work with plane graphs. As an example, consider the following partitioning theorem.

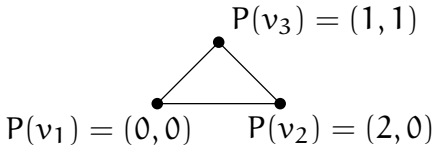
Theorem 2.38 (Schnyder [22]) *For every maximal planar graph* G *on at least three vertices and every fixed face* f *of* G*, the multigraph obtained from* G *by doubling the (three) edges of* f *can be partitioned into three spanning trees.*

Exercise 2.39 *Prove Theorem 2.38.* Hint: *Take a canonical ordering and build one tree by taking for every vertex* v_k *the edge to its first neighbor on the outer cycle* $C_\circ(G_{k-1})$.

Of a similar flavor is the following open problem, for which only partial answers for specific types of point sets are known [3, 1].

Problem 2.40 (In memoriam Ferran Hurtado (1951–2014))
Can every complete geometric graph on $n = 2k$ vertices (the complete straight line graph on a set of n points in general position) be partitioned into k plane spanning trees?

The shift-algorithm. Let $(v_1, \ldots, v_n)$ be a canonical ordering. The general plan is to construct an embedding by inserting vertices in this order, starting from the triangle $P(v_1) = (0,0)$, $P(v_3) = (1,1)$, $P(v_2) = (2,0)$. At each step, some vertices will be shifted to the right, making room for the edges to the freshly inserted vertex. For each vertex v_i already embedded, maintain a set $L(v_i)$ of vertices that move rigidly together with v_i. Initially, $L(v_i) = \{v_i\}$, for $1 \leqslant i \leqslant 3$.



Ensure that the following invariants hold after Step k (that is, after v_k has been inserted):

(i) $P(v_1) = (0,0)$, $P(v_2) = (2k-4, 0)$;

(ii) The x-coordinates of the points on $C_\circ(G_k) = (w_1, \ldots, w_t)$ are strictly increasing (in this order)[4];

[4]The notation is a bit sloppy here because both t and the w_i in general depend on k. So in principle we should write w_i^k instead of w_i. But as the k would just make a constant appearance throughout, we omit it to avoid index clutter.

(iii) each edge of $C_\circ(G_k)$ is drawn as a straight-line segment with slope ± 1.

Clearly these invariants hold for G_3, embedded as described above. Invariant (i) implies that after Step n we have $P(v_2) = (2n-4, 0)$, while (iii) implies that the Manhattan distance[5] between any two points on $C_\circ(G_k)$ is even.

Idea: put v_{k+1} at $\mu(w_p, w_q)$, where $w_p, \ldots, w_q$ are its neighbors on $C_\circ(G_k)$ (recall that they appear consecutively along $C_\circ(G_k)$ by definition of a canonical ordering), where

$$\mu((x_p, y_p), (x_q, y_q)) = \frac{1}{2}(x_p - y_p + x_q + y_q, -x_p + y_p + x_q + y_q)$$

is the point of intersection between the line $\ell_1 : y = x - x_p + y_p$ of slope 1 through $w_p = (x_p, y_p)$ and the line $\ell_2 : y = x_q - x + y_q$ of slope -1 through $w_q = (x_q, y_q)$.

Proposition 2.41 *If the Manhattan distance between w_p and w_q is even, then $\mu(w_p, w_q)$ is on the integer grid.*

Proof. By Invariant (ii) we know that $x_p < x_q$. Suppose without loss of generality that $y_p \leqslant y_q$. The Manhattan distance d of w_p and w_q is $x_q - x_p + y_q - y_p$, which by assumption is an even number. Adding the even number $2x_p$ to d yields the even number $x_q + x_p + y_q - y_p$, half of which is the x-coordinate of $\mu((x_p, y_p), (x_q, y_q))$. Adding the even number $2y_p$ to d yields the even number $x_q - x_p + y_q + y_p$, half of which is the y-coordinate of $\mu((x_p, y_p), (x_q, y_q))$. □

After Step n we have $P(v_n) = (n-2, n-2)$, because v_n is a neighbor of both v_1 and v_2. However, $P(v_{k+1})$ may not "see" all of $w_p, \ldots, w_q$, in case that the slope of $w_p w_{p+1}$ is 1 and/or the slope of $w_{q-1} w_q$ is -1 (Figure 2.20).

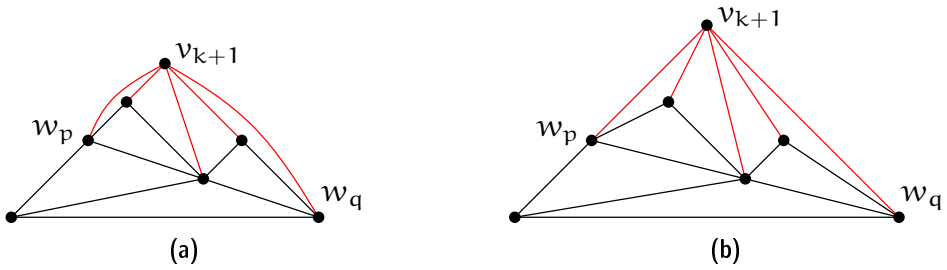


Figure 2.20: *(a) The new vertex v_{k+1} is adjacent to all of $w_p, \ldots, w_q$. If we place v_{k+1} at $\mu(w_p, w_q)$, then some edges may overlap, in case that w_{p+1} lies on the line of slope 1 through w_p or w_{q-1} lies on the line of slope -1 through w_q; (b) shifting $w_{p+1}, \ldots, w_{q-1}$ by one and $w_q, \ldots, w_t$ by two units to the right solves the problem.*

In order to resolve these problems we shift some points around so that after the shift w_{p+1} does not lie on the line of slope 1 through w_p and w_{q-1} does not lie on the line of slope -1 through w_q. The process of inserting v_{k+1} then looks as follows.

[5] The *Manhattan distance* of two points (x_1, y_1) and (x_2, y_2) is $|x_2 - x_1| + |y_2 - y_1|$.

1. Shift $\bigcup_{i=p+1}^{q-1} L(w_i)$ to the right by one unit.
2. Shift $\bigcup_{i=q}^{t} L(w_i)$ to the right by two units.
3. $P(v_{k+1}) := \mu(w_p, w_q)$.
4. $L(v_{k+1}) := \{v_{k+1}\} \cup \bigcup_{i=p+1}^{q-1} L(w_i)$.

Observe that the Manhattan distance between w_p and w_q remains even, because the shift increases their x-difference by two and leaves the y-coordinates unchanged. Therefore by Proposition 2.41 the vertex v_{k+1} is embedded on the integer grid.

The slopes of the edges $w_p w_{p+1}$ and $w_{q-1} w_q$ (might be just a single edge, in case that $p+1 = q$) become < 1 in absolute value, whereas the slopes of all other edges along the outer cycle remain ± 1. As all edges from v_{k+1} to $w_{p+1}, \ldots, w_{q-1}$ have slope > 1 in absolute value, and the edges $v_{k+1} w_p$ and $v_{k+1} w_q$ have slope ± 1, each edge $v_{k+1} w_i$, for $i \in \{p, \ldots, q\}$ intersects the outer cycle in exactly one point, which is w_i. In other words, adding all edges from v_{k+1} to its neighbors in G_k as straight-line segments results in a plane drawing.

Next we argue that the invariants (i)–(iii) are maintained. For (i) note that we start shifting with w_{p+1} only so that even in case that v_1 is a neighbor of v_{k+1}, it is never shifted. On the other hand, v_2 is always shifted by two, because we shift every vertex starting from (and including) w_q. Clearly both the shifts and the insertion of v_{k+1} maintain the strict order along the outer cycle, and so (ii) continues to hold. Finally, regarding (iii) note that the edges $w_p w_{p+1}$ and $w_{q-1} w_q$ (possibly this is just a single edge) are the only edges on the outer cycle whose slope is changed by the shift. But these edges do not appear on $C_\circ(G_{k+1})$ anymore. The two edges $v_{k+1} w_p$ and $v_{k+1} w_q$ incident to the new vertex v_{k+1} that appear on $C_\circ(G_{k+1})$ have slope 1 and -1, respectively. So all of (i)–(iii) are invariants of the algorithm, indeed.

So far we have argued about the shift with respect to vertices on the outer cycle of G_k only. To complete the proof of Theorem 2.32 it remains to show that the drawing remains plane under shifts also in its interior part.

Lemma 2.42 *Let G_k, $3 \leqslant k \leqslant n$, be straight-line grid embedded as described, $C_\circ(G_k) = (w_1, \ldots, w_t)$, and let $\delta_1 \leqslant \ldots \leqslant \delta_t$ be non-negative integers. If for each i, we shift $L(w_i)$ by δ_i to the right, then the resulting straight-line drawing is plane.*

Proof. Induction on k. For G_3 this is obvious. Let $v_k = w_\ell$, for some $1 < \ell < t$. Construct a delta sequence Δ for G_{k-1} as follows. If v_k has only two neighbors in G_k, then $C_\circ(G_{k-1}) = (w_1, \ldots, w_{\ell-1}, w_{\ell+1}, \ldots, w_t)$ and we set $\Delta = \delta_1, \ldots, \delta_{\ell-1}, \delta_{\ell+1}, \ldots, \delta_t$. Otherwise, $C_\circ(G_{k-1}) = (w_1, \ldots, w_{\ell-1} = u_1, \ldots, u_m = w_{\ell+1}, \ldots, w_t)$, where $u_1, \ldots, u_m$ are the $m \geqslant 3$ neighbors of v_k in G_k. In this case we set

$$\Delta = \delta_1, \ldots, \delta_{\ell-1}, \underbrace{\delta_\ell, \ldots, \delta_\ell}_{m-2 \text{ times}}, \delta_{\ell+1}, \ldots, \delta_t \,.$$

Clearly, Δ is monotonely increasing and by the inductive assumption a correspondingly shifted drawing of G_{k-1} is plane. When adding v_k and its incident edges back, the drawing remains plane: All vertices $u_2, \ldots, u_{m-1}$ (possibly none) move rigidly with (by exactly the same amount as) v_k by construction. Stretching the edges of the chain $w_{\ell-1}, w_\ell, w_{\ell+1}$ by moving $w_{\ell-1}$ to the left and/or $w_{\ell+1}$ to the right cannot create any crossings. □

Linear time. *(This part was not covered in the lecture.)* The challenge in implementing the shift algorithm efficiently lies in the eponymous shift operations, which modify the x-coordinates of potentially many vertices. In fact, it is not hard to see that a naive implementation—which keeps track of all coordinates explicitly—may use quadratic time. De Fraysseix et al. described an implementation of the shift algorithm that uses $O(n \log n)$ time. Then Chrobak and Payne [8] observed how to improve the runtime to linear, using the following ideas.

Recall that $P(v_{k+1}) = (x_{k+1}, y_{k+1})$, where

$$\begin{aligned} x_{k+1} &= \frac{1}{2}(x_p - y_p + x_q + y_q) \text{ and} \\ y_{k+1} &= \frac{1}{2}(-x_p + y_p + x_q + y_q) = \frac{1}{2}((x_q - x_p) + y_p + y_q)\,. \end{aligned} \tag{2.43}$$

Thus,

$$x_{k+1} - x_p = \frac{1}{2}((x_q - x_p) + y_q - y_p)\,. \tag{2.44}$$

$\Rightarrow$ We need the y-coordinates of w_p and w_q together with the relative x-position (offset) of w_p and w_q only in to determine the y-coordinate of v_{k+1} and its offset to w_p.

Maintain the outer cycle as a rooted binary tree T, with root v_1. For each node v of T, the *left child* is the first vertex covered by insertion of v (if any), that is, w_{p+1} in the terminology from above (if $p + 1 \neq q$), whereas the *right child* of v is the next node along the outer cycle (if any; either along the current outer cycle or along the one at the point where both points were covered together). See Figure 2.21 for an example.

At each node v of T we also store its x-offset $dx(v)$ with respect to the parent node. For the root v_1 of the tree set $dx(v_1) = 0$. In this way, a whole subtree (and, thus, a whole set $L(\cdot)$) can be shifted by changing a single offset entry at its root.

Initially, $dx(v_1) = 0$, $dx(v_2) = dx(v_3) = 1$, $y(v_1) = y(v_2) = 0$, $y(v_3) = 1$, $\text{left}(v_1) = \text{left}(v_2) = \text{left}(v_3) = 0$, $\text{right}(v_1) = v_3$, $\text{right}(v_2) = 0$, and $\text{right}(v_3) = v_2$.

Inserting a vertex v_{k+1} works as follows. As before, let $w_1, \ldots, w_t$ denote the vertices on the outer cycle $C_\circ(G_k)$ and $w_p, \ldots, w_q$ be the neighbors of v_{k+1}.

1. Increment $dx(w_{p+1})$ and $dx(w_q)$ by one. *(This implements the shift.)*
2. Compute $\Delta_{pq} = \sum_{i=p+1}^{q} dx(w_i)$. *(This is the total offset between w_p and w_q.)*

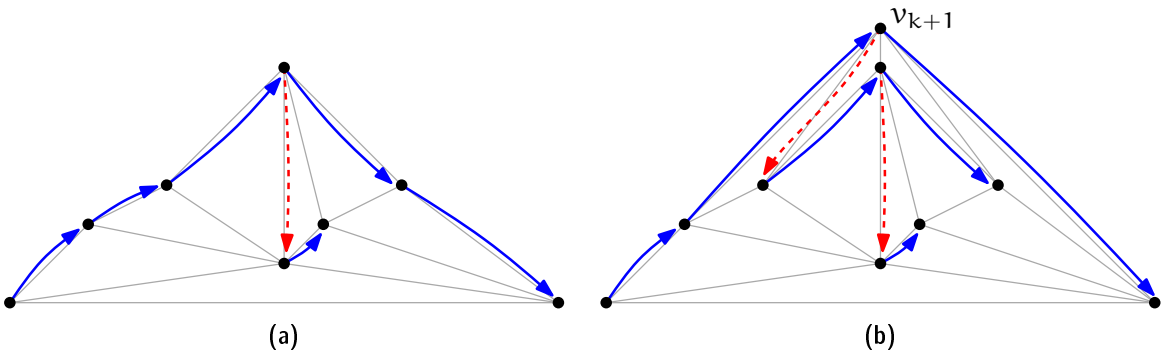


Figure 2.21: *Maintaining the binary tree representation when inserting a new vertex v_{k+1}. Red (dashed) arrows point to left children, blue (solid) arrows point to right children.*

3. Set $\mathrm{dx}(v_k) \leftarrow \frac{1}{2}(\Delta_{pq} + y(w_q) - y(w_p))$ and $y(v_k) \leftarrow \frac{1}{2}(\Delta_{pq} + y(w_q) + y(w_p))$. *(This is exactly what we derived in (2.43) and (2.44).)*

4. Set $\mathrm{right}(w_p) \leftarrow v_k$ and $\mathrm{right}(v_k) \leftarrow w_q$. *(Update the current outer cycle.)*

5. If $p+1 = q$, then set $\mathrm{left}(v_k) \leftarrow 0$; else set $\mathrm{left}(v_k) \leftarrow w_{p+1}$ and $\mathrm{right}(w_{q-1}) \leftarrow 0$. *(Update the part that is covered by insertion of v_{k+1}.)*

6. Set $\mathrm{dx}(w_q) \leftarrow \Delta_{pq} - \mathrm{dx}(v_k)$ and—unless $p+1 = q$—set $\mathrm{dx}(w_{p+1}) \leftarrow \mathrm{dx}(w_{p+1}) - \mathrm{dx}(v_k)$. *(Update the offsets according to the changes in the previous two steps.)*

Observe that the only step that possibly cannot be executed in constant time is Step 2. But all vertices but the last vertex w_q for which we sum the offsets are covered by the insertion of v_{k+1}. As every vertex can be covered at most once, the overall complexity of this step during the algorithm is linear. Therefore, this first phase of the algorithm can be completed in linear time.

In a second phase, the final x-coordinates can be computed from the offsets by a single recursive pre-order traversal of the tree. The (pseudo–)code given below is to be called with the root vertex v_1 and an offset of zero. Clearly this yields a linear time algorithm overall.

```
compute_coordinate(Vertex v, Offset d) {
  if (v == 0) return;
  x(v) = dx(v) + d;
  compute_coordinate(left(v), x(v));
  compute_coordinate(right(v), x(v));
}
```

Remarks. From a geometric complexity point of view, Theorem 2.32 provides very good news for planar graphs in a similar way that the Euler Formula does from a combinatorial

complexity point of view. Euler's Formular tells us that we can obtain a combinatorial representation (for instance, as a DCEL) of any plane graph using $O(n)$ space, where n is the number of vertices.

Now the shift algorithm tells us that for any planar graph we can even find a geometric plane (straight-line) representation using $O(n)$ space. In addition to the combinatorial information, we only have to store $2n$ numbers from the range $\{0, 1, \ldots, 2n-4\}$.

When we make such claims regarding space complexity we implicitly assume the so-called *word RAM model*. In this model each address in memory contains a *word* of b bits, which means that it can be used to represent any integer from $\{0, \ldots, 2^b - 1\}$. One also assumes that b is sufficiently large, for instance, in our case $b \geqslant \log n$.

There are also different models such as the *bit complexity model*, where one is charged for every bit used to store information. In our case that would already incur an additional factor of $\log n$ for the combinatorial representation: for instance, for each halfedge we store its endpoint, which is an index from $\{1, \ldots, n\}$.

Questions

1. *What is an embedding? What is a planar/plane graph?* Give the definitions and explain the difference between planar and plane.

2. *How many edges can a planar graph have? What is the average vertex degree in a planar graph?* Explain Euler's formula and derive your answers from it.

3. *How can plane graphs be represented on a computer?* Explain the DCEL data structure and how to work with it.

4. *How can a given plane graph be (topologically) triangulated efficiently?* Explain what it is, including the difference between topological and geometric triangulation. Give a linear time algorithm, for instance, as in Theorem 2.25.

5. *What is a combinatorial embedding? When are two combinatorial embeddings equivalent? Which graphs have a unique combinatorial plane embedding?* Give the definitions, explain and prove Whitney's Theorem.

6. *What is a canonical ordering and which graphs admit such an ordering? For a given graph, how can one find a canonical ordering efficiently?* Give the definition. State and prove Theorem 2.36.

7. *Which graphs admit a plane embedding using straight line edges? Can one bound the size of the coordinates in such a representation?* State and prove Theorem 2.32.

References

[1] Oswin Aichholzer, Thomas Hackl, Matias Korman, Marc van Kreveld, Maarten Löffler, Alexander Pilz, Bettina Speckmann, and Emo Welzl, Packing plane spanning trees and paths in complete geometric graphs. In *Proc. 26th Canad. Conf. Comput. Geom.*, pp. 233–238, 2014, URL http://cccg.ca/proceedings/2014/papers/paper34.pdf.

[2] Bruce G. Baumgart, A polyhedron representation for computer vision. In *Proc. AFIPS Natl. Comput. Conf.*, vol. 44, pp. 589–596, AFIPS Press, Alrington, Va., 1975, URL http://dx.doi.org/10.1145/1499949.1500071.

[3] Prosenjit Bose, Ferran Hurtado, Eduardo Rivera-Campo, and David R. Wood, Partitions of complete geometric graphs into plane trees. *Comput. Geom. Theory Appl.*, **34**, 2, (2006), 116–125, URL http://dx.doi.org/10.1016/j.comgeo.2005.08.006.

[4] John M. Boyer and Wendy J. Myrvold, On the cutting edge: simplified O(n) planarity by edge addition. *J. Graph Algorithms Appl.*, **8**, 3, (2004), 241–273, URL http://jgaa.info/accepted/2004/BoyerMyrvold2004.8.3.pdf.

[5] Sergio Cabello, Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *J. Graph Algorithms Appl.*, **10**, 2, (2006), 353–363, URL http://jgaa.info/accepted/2006/Cabello2006.10.2.pdf.

[6] Bernard Chazelle, Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, **6**, 5, (1991), 485–524, URL http://dx.doi.org/10.1007/BF02574703.

[7] Norishige Chiba and Takao Nishizeki, The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *J. Algorithms*, **10**, 2, (1989), 187–211, URL http://dx.doi.org/10.1016/0196-6774(89)90012-6.

[8] Marek Chrobak and Thomas H. Payne, A linear-time algorithm for drawing a planar graph on a grid. *Inform. Process. Lett.*, **54**, (1995), 241–246, URL http://dx.doi.org/10.1016/0020-0190(95)00020-D.

[9] István Fáry, On straight lines representation of planar graphs. *Acta Sci. Math. Szeged*, **11**, (1948), 229–233.

[10] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl, Trémaux Trees and Planarity. *Internat. J. Found. Comput. Sci.*, **17**, 5, (2006), 1017—-1030, URL http://dx.doi.org/10.1142/S0129054106004248.

[11] Hubert de Fraysseix, János Pach, and Richard Pollack, How to Draw a Planar Graph on a Grid. *Combinatorica*, **10**, 1, (1990), 41–51, URL http://dx.doi.org/10.1007/BF02122694.

[12] Leonidas J. Guibas and Jorge Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, **4**, 2, (1985), 74–123, URL http://dx.doi.org/10.1145/282918.282923.

[13] John Hopcroft and Robert E. Tarjan, Efficient planarity testing. *J. ACM*, **21**, 4, (1974), 549–568, URL http://dx.doi.org/10.1145/321850.321852.

[14] Ken ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed, The disjoint paths problem in quadratic time. *J. Combin. Theory Ser. B*, **102**, 2, (2012), 424–435, URL http://dx.doi.org/10.1016/j.jctb.2011.07.004.

[15] Lutz Kettner, *Software design in computational geometry and contour-edge based polyhedron visualization*. Ph.D. thesis, ETH Zürich, Zürich, Switzerland, 1999, URL http://dx.doi.org/10.3929/ethz-a-003861002.

[16] Kazimierz Kuratowski, Sur le problème des courbes gauches en topologie. *Fund. Math.*, **15**, 1, (1930), 271–283, URL http://matwbn.icm.edu.pl/ksiazki/fm/fm15/fm15126.pdf.

[17] László Lovász, Graph Minor Theory. *Bull. Amer. Math. Soc.*, **43**, 1, (2005), 75–86, URL http://dx.doi.org/10.1090/S0273-0979-05-01088-8.

[18] Bojan Mohar and Carsten Thomassen, *Graphs on surfaces*. Johns Hopkins University Press, Baltimore, 2001.

[19] David E. Muller and Franco P. Preparata, Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, **7**, (1978), 217–236, URL http://dx.doi.org/10.1016/0304-3975(78)90051-8.

[20] János Pach and Rephael Wenger, Embedding planar graphs at fixed vertex locations. *Graphs Combin.*, **17**, (2001), 717–728, URL http://dx.doi.org/10.1007/PL00007258.

[21] Neil Robertson and Paul Seymour, Graph Minors. XX. Wagner's Conjecture. *J. Combin. Theory Ser. B*, **92**, 2, (2004), 325–357, URL http://dx.doi.org/10.1016/j.jctb.2004.08.001.

[22] Walter Schnyder, Planar Graphs and Poset Dimension. *Order*, **5**, (1989), 323–343, URL http://dx.doi.org/10.1007/BF00353652.

[23] Carsten Thomassen, Kuratowski's Theorem. *J. Graph Theory*, **5**, 3, (1981), 225–241, URL http://dx.doi.org/10.1002/jgt.3190050304.

[24] William T. Tutte, A theorem on planar graphs. *Trans. Amer. Math. Soc.*, **82**, 1, (1956), 99–116, URL http://dx.doi.org/10.1090/S0002-9947-1956-0081471-8.

[25] Klaus Wagner, Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, **46**, (1936), 26–32, URL http://eudml.org/doc/146109.

[26] Klaus Wagner, Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, **114**, 1, (1937), 570–590, URL http://dx.doi.org/10.1007/BF01594196.

[27] Kevin Weiler, Edge-based data structures for solid modeling in a curved surface environment. *IEEE Comput. Graph. Appl.*, **5**, 1, (1985), 21–40, URL http://dx.doi.org/10.1109/MCG.1985.276271.

[28] Hassler Whitney, Congruent Graphs and the Connectivity of Graphs. *Amer. J. Math.*, **54**, 1, (1932), 150–168, URL http://www.jstor.org/stable/2371086.

[29] Wikipedia, Planarity testing — Wikipedia, The Free Encyclopedia. 2013, URL http://en.wikipedia.org/wiki/Planarity_testing, [Online; accessed 19-Feb-2013].