GEOMGRAPHS: Algorithms and Combinatorics of Geometric Graphs
MPRI 2025-2026

Arnaud de Mesmay

These are the lecture notes for my half of the course GEOMGRAPHS. The other half is taught by Luca Castelli Aleardi and the slides and exercise sheets for his half are available on the course's webpage.

Some practicalities:

- The course is on Thursdays, from 8:45 to 11:45, in Sophie Germain room 1002.
- There is an exercise session in the middle of each lecture, after the half-time break.
- The class will be graded with a final written exam.
- There will be two optional exercise sheets, with points contributing extra credits for the final grade.

This aim of this course is to provide an overview of the combiantorial, geometric and algorithmic properties of embedded graphs: we start with ***planar graphs*** and then move on to ***surface-embedded graphs***. These are graphs that can be drawn without crossings in the plane or on more complicated surfaces, see Figure 1. Therefore they form a ***combinatorial*** object with a ***topological*** constraint. The objective of the course is to ***explore how topology interacts with combinatorics and algorithms*** on this very natural class of objects. Some questions we will explore are:

1. What are the combinatorial consequences of being planar? Are there combinatorial characterizations?
2. How to test algorithmically whether a graph is planar? If yes, how to draw the graph?
3. Can one exploit planarity to design better algorithms for planar graphs than for general graphs?
4. How do the previous answers generalize to graphs embedded in these more complicated surfaces?
5. How to solve certain topological questions algorithmically on surfaces?

While the focus of the course stays very theoretical (e.g., mostly with a theorem, lemma, proof structure), embedded graphs are of great interest for the practically-oriented mind, as they appear everywhere, for example in road networks (where underpasses and bridges can be modeled using additional topological features), chip design or the meshes that are ubiquitous in computer graphics or computer aided design. In all these applications, there is a strong need

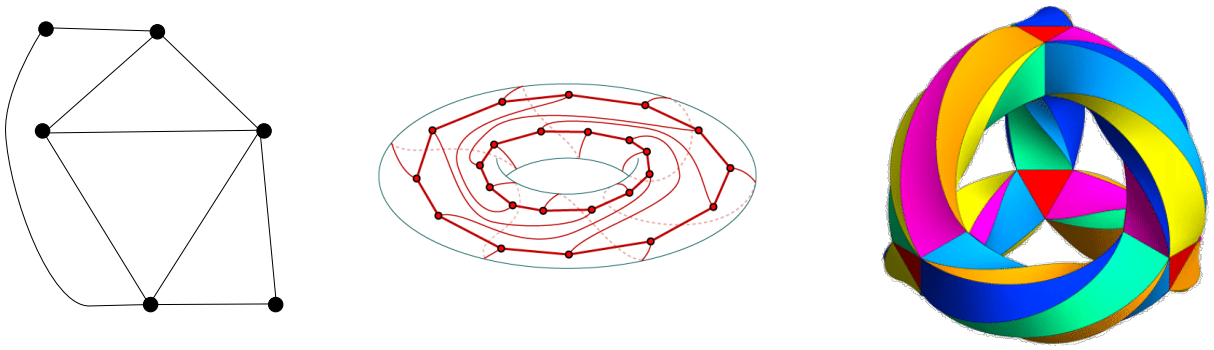

Figure 1: Graphs embedded in the plane, the torus, and the three-holed torus.

for a theoretical understanding of embedded graphs, as well as algorithmic primitives related to their topological features. Additionally, embedded graphs are an important lens to study graphs in general, since any graph can be embedded on some surface. This is especially the case in graph minor theory, where embedded graphs play an absolutely central role (but we barely touch on this topic).

The class is taught alternatively by Luca and myself: his class focuses on combinatorial aspects and graph-drawing questions, while my half covers algorithmic questions and topological aspects of surface-embedded graphs.

These lecture notes follow quite closely the material taught in my half of the class. This is actually their point, as in my experience lecture notes with too much content can easily get overwhelming (especially when one misses a class). Thus we refrain from (too many) digressions and heavy references. The tone aims to be conversational. I will do my best however to add each week the missing details for the proofs which may have been a bit handwavy during the lectures. The content has a strong overlap with the lecture notes of a previous iteration of the class (by Éric Colin de Verdière), and with those for a course on Computational Topology I co-taught with Francis Lazarus in 2016-2018, and we refer to, and strongly recommend those for the missing digressions and references.

1 Planar Graphs

1.1 A partial recap of the first lecture

A ***planar graph*** is a graph that can be embedded, i.e., drawn without crossings, in the plane, or equivalently the sphere. Throughout this course, it is quite important that we allow graphs to have multiple edges and loops, which is handled seamlessly by the previous definition. A graph without multiple edges nor loops is called a ***simple graph***. A ***plane graph*** is an embedding of a planar graph, and two plane graphs are considered equivalent if there is a homeomorphism of the plane sending one to the other one (or intuitively, if one can continuously deform one into the other without adding crossings). Note that a planar graph can correspond to multiple different plane graphs: think for example of a graph with a single vertex and multiple loops.

Compared to just a general graph, a plane graph has an additional combinatorial structure: there are now ***faces***, which are the connected components of the complement $\mathbb{R}^2 \setminus G$. This additional structure interacts with the initial graph in multiple ways:

- The ***Euler characteristic*** stipulates that for any connected plane graph, we have $v - e + f = 2$, where v, e and f are respectively the number of vertices, edges and faces.
- This implies that planar graphs are ***sparse***: for $v \geq 3$, $e \leq 3v - 6$ (and in general $e \leq 3v$. So planar graphs are very particular compared to general graphs.
- To any plane graph G we can associate a ***dual graph*** G^*, whose vertices are the faces of G and whose edges connect adjacent faces of G (with multiplicity and loops if needed). Note that the dual of a simple graph is in general not simple, and that the dual of graph depends on the embedding.
- The combinatorial data of the faces is actually all that is needed to encode a plane graph. There are various data structures one can use to do that. In these notes, we will simply consider that we have such a data structure that allows us to do "intuitive" operations in the natural time: for example moving from an edge incident to a vertex to the next edge in the circular order in constant time, or listing the k edges adjacent to a face in time $O(k)$, etc.

All of the properties of planar and plane graphs boil down to "intuitive" (but hard to prove) topological properties of the plane: the ***Jordan curve theorem*** shows that any simple (i.e., non self-intersecting) closed curve in the plane separates it into two components, one bounded and one unbounded. The ***Jordan Schoenflies theorem*** shows that the bounded component is homeomorphic to a disk. These theorems are hard to prove because simple closed curves in the plane can be very complicated, as pictured in Figure 2. If one restricts our attention, say, to polygonal curves, then the proofs become much easier. This leaves us with two alternatives for the class: either we define all our graphs to have edges as polygonal segments, and then all the intuitive facts are easy to prove (there is such a proof for the Jordan curve theorem in my older lecture notes), or we take the general definition and then trust these hard theorems that everything works. I leave you to choose your preferred option.

In particular, all the bounded faces of a connected plane graph are (homeomorphic to) a disk. We say that such a graph is ***cellularly embedded*** (this definition is quite useless for plane graphs but will become useful on other surfaces). We will see later on that the failure of such nice topological properties for more complicated surfaces leads to interesting topological questions.

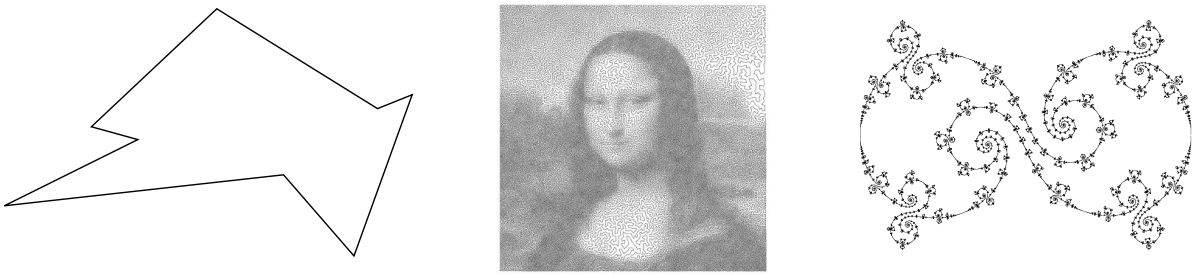

Figure 2: Three simple closed curves in the plane

We also recall the famous theorem of Kuratowski, which will not be proved in this class (nor in Luca's) (you can find a proof in my older lecture notes).

Theorem 1.1 (Kuratowski, 1929)**.** *A graph is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$ as a subgraph.*

Finally, the Fàry theorem shows that every plane graph can be realized with edges drawn as straight lines. This is proved in Luca's half as a corollary of the Tutte embedding theorem (there are also easier direct proofs).

1.2 Coloring

The sparsity has the following nice implication. A k-(vertex) ***coloring*** of a simple graph is an assignment of k colors to the vertices so that no two adjacent vertices share a common color.

Proposition 1.2. *Simple planar graphs are* 6*-colorable.*

Proof. We prove the result by induction on the number of vertices. For low values, this is immediate. For the induction step, pick a vertex z of degree less than 6 which exists because of the sparsity, and color inductively $G \setminus z$. Then the five neighbors have at most 5 different colors, and we can color z with one of the remaining colors. □

The following improvement is on the exercise sheet, and relies on more than just the Euler characteristic and the sparsity.

Exercise 1.3. Prove that simple planar graphs are 5-colorable. Hint: look at paths connecting non-adjacent neighbors of a degree 5 vertex.

As is well-known, planar graphs are actually 4-colorable, and we will not prove this in the course.

1.3 The crossing lemma

A ***drawing*** of a graph if just a continuous map $f : G \to \mathbb{R}^2$, that is, a drawing of the graph on the plane where crossings *are* allowed. We will only consider drawings ***in general position***: that means that vertices must be mapped to distinct points, edges do not intersect vertices except at their endpoints, edges only intersect transversely and at most two edges intersect at each point. Note that such a drawing in general position induces a plane graph, where every crossing has been replaced by a vertex of degree 4. As we said earlier, via Fàry's theorem, such a plane graph can be assumed to have its edges realized as straight lines. So without loss of

generality, we can and we will assume that this is the case in all our drawings: this implies that all the edges are drawn as polygonal segments (which may bend at crossings).

The ***crossing number*** $cr(G)$ of a graph is the minimal number of crossings over all the possible drawings in general position of G. For instance, $cr(G) = 0$ if and only if G is planar. The ***crossing number inequality*** provides the following lower bound on the crossing number.

Theorem 1.4. $cr(G) \geq \frac{|E|^3}{64|V|^2}$ *if* $|E| \geq 4|V|$.

The proof is a surprising application of (basic) probabilistic tools. There is a nice and much more indepth discussion of this proof available on Terry Tao's blog.

Proof. Starting with a drawing of G with the minimal number of crossings, define a new graph G' obtained by removing one edge for each crossing. This graph is planar since we removed all the crossings, and it has at least $|E| - cr(G)$ edges, so we obtain that $|E| - cr(G) \leq 3|V|$ (Note that we removed the -6 to obtain an inequality valid for any number of vertices). This gives in turn

$$cr(G) \geq |E| - 3|V|.$$

This can be amplified in the following way. Starting from G, define another graph by removing vertices (and the edges adjacent to them) at random with some probability $1 - p < 1$, and denote by G'' the resulting graph. Taking the previous inequality with expectations, we obtain $\mathbb{E}(cr(G'')) \geq \mathbb{E}(|E''|) - 3\mathbb{E}(|V''|)$. Since vertices are removed with probability $1 - p$, we have $\mathbb{E}(|V''|) = p|V|$. An edge survives if and only if both its endpoints survives, so $\mathbb{E}(|E''|) = p^2|E|$. Finally a crossing survives if an only if the four adjacent vertices survive[1]. The resulting drawing might not be crossing-minimal but it does imply that $\mathbb{E}(cr(G'') \leq p^4 cr(G)$, which is the inequality we will need. So we obtain

$$cr(G) \geq p^{-2}|E| - 3p^{-3}|V|,$$

and taking $p = 4|V|/|E|$ (which is less than 1 if $|E| \geq 4|V|$) gives the result. □

In particular, applying this inequality to dense graphs, and in particular to complete graphs K_n shows that any drawing of the complete graph K_n has $\Omega(n^4)$ crossings. Finding the correct constants is notoriously difficult though, even for that very specific-looking case of complete graphs: the Hill conjecture posits that

$$cr(K_n) = \frac{1}{4}\lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor\lfloor\frac{n-2}{2}\rfloor\lfloor\frac{n-3}{2}\rfloor \sim \frac{1}{64}n^4,$$

but the best known lower bound is quite far off[2], at about $cr(K_n) \geq 0.8594\frac{1}{64}n^4$ for sufficiently large n. Likewise, the correct constant is unknown for complete bipartite graphs, and in general there are tons of simple-looking open problems on crossing numbers (see this exhaustive survey by Marcus Schaefer).

[1] There may be less than four adjacent vertices in general, but not in the drawing minimizing the crossing number, since drawings where there is a crossing forming an α shape can be simplified by removing one crossing. So in a crossing-minimal drawing, they do not happen

[2] Note that the fact that 1/64 also appears in Theorem 1.4 is a red herring: the number of edges in a complete graph is $\binom{n}{2}$, so a blind application of the crossing lemma only gives $cr(K_n) \geq \frac{1}{64 \cdot 2^3}n^4$.

1.4 The Hanani-Tutte theorem

Another interesting result of planar drawings is the following surprising (to me at least) theorem. Two edges of a graph are ***independent*** if they are not incident to a common vertex.

Theorem 1.5 (Strong Hanani-Tutte theorem, 1934)**.** *Any drawing in general position of a non-planar graph contains two independent edges that cross an odd number of times.*

Conversely, if one can draw a graph so that all independent edges cross an even number of times, then the graph is planar (!)

Proof. We first claim that it suffices to prove the theorem for K_5 and $K_{3,3}$. Indeed, let G be a non-planar graph. By Kuratowski's theorem, it contains a subdivision of K_5 or $K_{3,3}$. If the theorem is proved for K_5 and $K_{3,3}$, then we can find a pair of disjoint paths in our graph G that cross an odd number of times. This means that there exists a pair of independent edges that cross an odd number of times. Indeed, otherwise, summing all the even number of crossings among all pairs of edges in the pair of paths would give an even number of crossings for the pair of paths.

In order to prove the theorem for K_5 and $K_{3,3}$, we prove the stronger result that in any drawing of these two graphs, the sum of the number of crossings over all independent pairs of edges is odd. The reason is that this quantity mod 2 does not actually depend on the drawing:

Claim 1.6. *For $G = K_5$ or $K_{3,3}$ and any two drawings G_1 and G_2 in general position of the graph G, the quantity*

$$\sum_{\substack{e,e' \\ \text{independent edges}}} cr(e,e') \mod 2$$

is the same.

Proof. We first describe a general way to deform any G_1 into any G_2, and then prove that the target quantity is invariant throughout this deformation.

As we said earlier, we can assume that in the drawings G_1 and G_2, the edges are polygonal segments. We first move the vertices of G_1 one by one so that they coincide with the respectives vertices of G_2. This can be done by choosing, for every vertex v in G, a polyogonal path p_v between its position v_1 in G_1 and its position v_2 in G_2. This path can clearly be chosen so that it avoids vertices of G_1 and G_2 and so that it intersects their edges transversely and outside of existing crossings. Then we move v_1 along p, dragging all the incident edges along the way, as pictured in Figure 3, top.

In a second step, we inductively straigthen the edges in G_1 and in G_2: we focus on G_1, the case of G_2 being identical. Each edge of G_1 is a polygonal path $e = (q_1, \dots, q_k)$, where the points q_i and q_{i+1} are connected with straight segments. We can straighten such a polygonal path by replacing the triangle q_1, q_2, q_3 by the straight segment q_1q_3: this is done by moving one tip of the triangle to the opposite edge. Here again, one can do so along a path that avoids all the vertices of G_1 and G_2 and intersects their edges transversely and outside of existing crossings, see Figure 3, middle.. We then go on inductively so that every edge in G_1 is a straight line between its endpoints, and likewise in G_2. Since the vertices coincide, the drawings are now identical.

Let us now analyze how the crossings evolve as we do these deformations. By our choice of deformation paths, such evolutions only happen at discrete moments: in the first step this will be when a vertex passes through an edge (Figure 3, (a)), and in the second step this will be when a point q_i crosses an edge (Figure 3, (b)), when one of the two incident straight lines

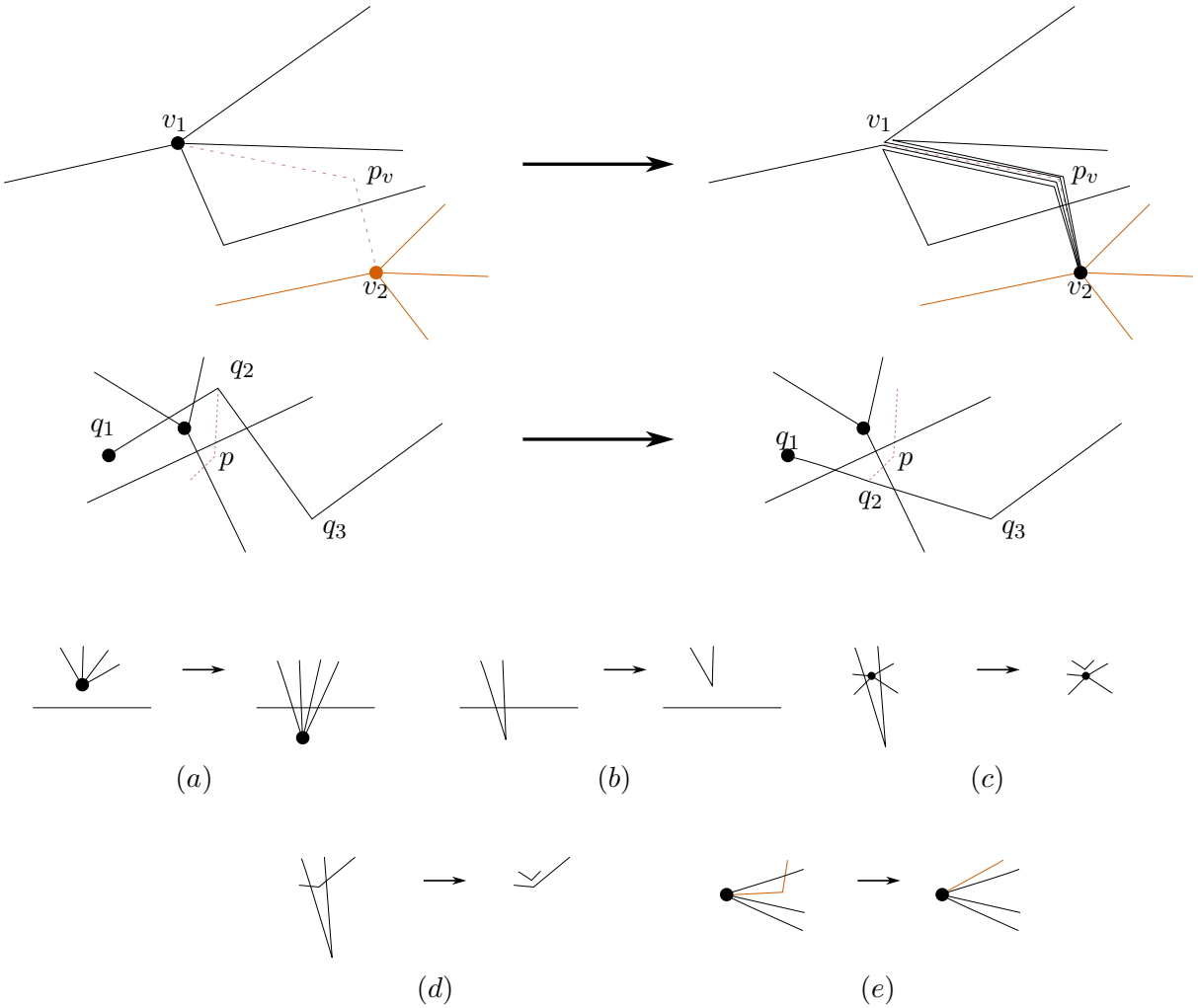

Figure 3: Deforming drawings.

passes through a vertex or a crossing (Figure 3, (c) and (d)), or when two edges switch their order around a vertex (Figure 3, (e)). Note that outside of these discrete events, the crossings do not change: actually the homeomorphism class of the drawing (viewed as a plane graph) does not change. Also note that the events of type (b), (d) and (e) do change the drawing but do not change the parity of crossings of independent edges.

There remains events (a) and (c) which both feature, in slightly different ways, a vertex v passing through an edge e. If v is incident to e , the changes in crossings only involve non-independent edges and are thus irrelevant to our count. If $G = K_{3,3}$, the degree of every vertex is 3, but for any vertex v not incident to e, exactly two of the edges incident to v are independent with e (look at Figure 4). So the count of independent crossings changes by 2, an even amount. If $G = K_5$, every vertex v in K_5 has degree exactly four, and for any fixed e not incident to v, exactly two of the edges incident to v are independent with e (look again at Figure 4). So here again the count of independent crossings changes by 2, an even amount. ⌟

We conclude the proof by exhbiting drawings of K_5 and $K_{3,3}$ where this sum is odd, as done in Figure 4 (note that the proof implies that *any* drawing will work). □

This theorem and its proof provide a polynomial-time algorithm to test whether a graph is planar. It is not as efficient as other more standard approaches, but is simple conceptually and, with a lot of work, can be generalized to higher dimensions. Let G be a graph of which

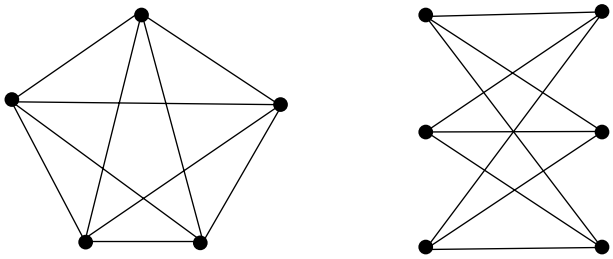

Figure 4: The graphs K_5 and $K_{3,3}$.

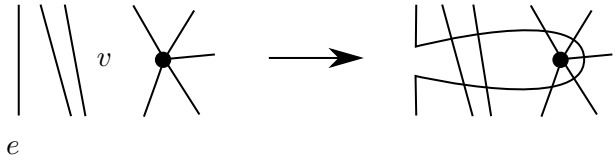

Figure 5: A finger move for a pair $p = (e, v)$.

we want to test planarity. Let us consider the vector space S over the field with 2 elements $\mathbb{F}_2$ whose basis consists of pairs of independent edges (e, f) in G.

1. Start with *any* drawing of the graph G. Write down the vector $x(D) \in S$ of the number of crossings mod 2 of independent edges.
2. For every pair $p = (v, e)$ in the graph G, define a vector $u(p) = \sum_{f \text{ incident to } v}(f, e) \in S$: it contains a 1 for each pair of edges (f, e) such that f is incident to e, and 0 otherwise.
3. Denote by U the subspace spanned by the family $u(p)$. Test whether $x(D)$ belongs to U. If yes, output that the graph is planar, otherwise, output that it is not planar.

Lemma 1.7. *This algorithm is correct and has polynomial complexity.*

Proof. The reason why this algorithm is correct follows from (the proof of) Theorem 1.5. Indeed, starting from any drawing D of G, one can change its vector $x(D)$ by exactly $u(p)$ by applying a ***finger move*** (see Figure 5) from the edge e around the vertex p. If $x(D)$ belongs to U, there exists a sequence of finger moves that brings the vector of crossings $x(D)$ to zero mod 2. Then by Theorem 1.5, the graph is planar. Conversely, if the graph is planar, starting from any drawing we can deform it similarly to the proof of Theorem 1.5 until there are no crossings remaining, and this morphing gives a sequence of finger moves, and thus a family of vectors $u(p)$ showing that $x(D)$ belongs to U.

The third step can be solved by Gaussian elimination in cubic time (or there are faster algorithms, even more so since the underlying field is $\mathbb{F}_2$). Since S has size $O(|E|^2) = O(|V|^2)$ (if the graph is not sparse, we can reject it straight away), this yields a complexity $O(|V|^6)$. □

For the algebraic-minded reader, this algorithm actually amounts to computing an obstruction class in the equivariant cohomology of the deleted product (ask me in class if you are interested about what these words mean).

1.5 Efficient algorithms for planar graphs

Many algorithmic problems can be solved faster when the input graph is planar. This includes some problems which are NP-hard in general but can be solved in polynomial time in the planar

case: for example MAX-CUT, (uniform) SPARSEST CUT, FEEDBACK ARC SET, or computing the BRANCH-WIDTH of a graph. Similarly, testing for GRAPH ISOMORPHISM is (probably) not NP-hard but no polynomial-time algorithm is known in general, while it can be solved efficiently on planar graphs (take a look at Exercise 3 in the Exercise Sheet accompanying Lecture 4). We will not study any of those (choices have to be made). Instead, we will look at a few problems where planarity allows for faster and conceptually simpler algorithms than in the general case. At the risk of overly simplifying things, there are in my opinion three main reasons as to why planar graphs tend to be easier to handle algorithmically than general graphs. The first reason is *sparsity*, which has strong combinatorial and algorithmic consequences, as we have seen earlier for colouring problems. The second reason is *duality*: sometimes a problem that is not easy to solve in the primal graph becomes much easier to solve in the dual graph. Sometimes juggling between both the primal and the dual setting allows to make good progress. The third reason is the existence of *small separators*, which will be proved in Lucas's half, and which allows for efficient divide and conquer approaches for a lot of problems. Famously, a clever use of (iterated) small separators yields the following theorem, showing that shortest paths, which are arguably the most important algorithmic primitive, can be computed in linear time on planar graphs. This is to be compared with Dijkstra's algorithm which runs in time $O(n \log n)$ in sparse graphs. (But note that for unweighted graphs, a breadth-first search tree from a vertex is a shortest path tree, so in that case, we can also compute shortest paths in time $O(n)$).

Theorem 1.8 (Henzinger, Klein, Rao, Subramanian 1997)**.** *On an edge-weighted planar graph, a shortest path tree from any given vertex can be computed in linear time.*

1.5.1 Minimum spanning trees

Let G be a graph with a weight function $w : E \to \mathbb{R}^+$. A ***minimum spanning tree*** of a graph $G = (V, E)$ is a tree $T = (V, E')$ with $E' \subseteq E$ that spans all the vertices of G (this was actually already implied by the notations of the vertices) and such that it has minimal total weight $\sum_{e \in E'} w(e)$ among all the spanning trees. Computing a minimum spanning tree is a fundamental primitive in algorithm design, and also an important practical problem in its own right: think about an electric company wanting to wire all the houses in a neighborhood at a minimal cost. For general graphs with n vertices and m edges, classical algorithms (e.g., Prim's or Kruskal's) run in time $O(m \log n)$ (note that in the sparse case, $m = O(n)$, but this is still not linear). Using some very fancy data structures, one can do better, but there is no known deterministic algorithm running in linear time.

In contrast, for planar graphs, one can compute a minimum spanning tree very easily in linear time. The key is the use of duality:

Theorem 1.9. *If G is a planar graph with n vertices, one can compute a minimum spanning tree in time $O(n)$.*

We start with exploring how spanning trees interact with duality, as illustrated in Figure 6.

Lemma 1.10. *Let G be a planar graph and G^* be its dual graph. Then if $T = (V, E')$ and $E' \subseteq E$ is a spanning tree, then $T^* = (F^*, (E \setminus E')^*)$ is also a spanning tree, called the co-tree. If one is minimal, the other is maximal.*

Proof. If T is a spanning tree, it does not contain any cycle, which happens if and only if its complement $\mathbb{R}^2 \setminus E'$ is connected, by the Jordan curve theorem. But $\mathbb{R}^2 \setminus E'$ is connected if and only if T^* is connected. Indeed, any two points in $\mathbb{R}^2 \setminus E'$ are in faces of G, and thus can be connected without crossing edges of E' if and only if those faces can be connected in the dual

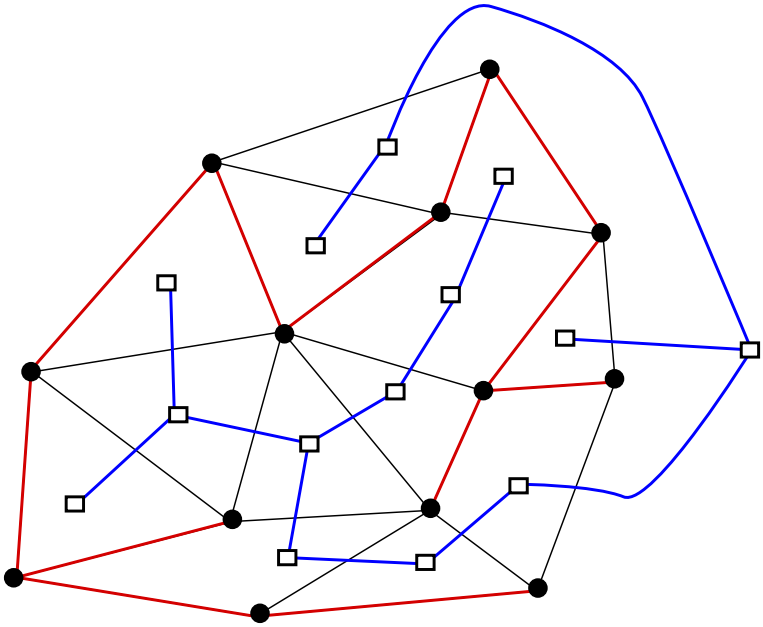

Figure 6: A spanning tree and the spanning co-tree. These are sometimes called ***interdigitating*** trees.

graph where one removed the edges dual to those of E'. Since T and T' are both connected, they are both acyclic and thus are both trees. If one of them, say T, is minimal, then T^* is maximal, since otherwise one could increase the weight of T^* which would mechanically decrease the weight of T. □

Observe that when we *contract* an edge in a graph, the corresponding edge in the dual graph gets *removed*, and vice-versa.

The following is an easy consequence of Euler's formula.

Lemma 1.11. *In a planar graph G, there is either a vertex of degree at most 3, or a face of degree at most 3.*

Proof. Assume otherwise. Then we can double count edges in two different ways and get $4v \leq 2e$, and $4f \leq 2e$. Then $v - e + f \leq 0$ contradicting Euler's formula. □

We now have all the tools to describe our algorithm.

Proof of Theorem 1.9. The algorithm is based on alternating actions between the primal and the dual graph. We initialize the set of edges E' that we pick at the empty set, and:

- Let v be a vertex of the graph G. If v is only surrounded by loops, then the solution is the trivial empty tree. Otherwise, at least one non-loop edge is adjacent to v. Among all these non-loop edges, one of minimal weight e necessarily belongs to the minimal spanning tree: otherwise, one could add it and remove another edge. So we can take it, add it to E' and recurse on G/e.
- Let f be a face of the graph G, and thus a vertex of the dual graph G^*. If all the edges adjacent to f in G^* are loops, then G^* has a single face, thus G is a tree, and the spanning tree is G itself. Otherwise, the dual of an edge e of maximal weight incident to

f necessarily belongs to the maximal spanning co-tree, and thus e does not belong to the minimum spanning tree. So we can recurse on $G \setminus e$.

Each of these two actions removes an edge in one way or another from the graph that we consider, so the number of recursions is $O(n)$. For each of the two actions, the cost of finding which edge to contract or remove is of the order of the degree of the vertex or the face that we consider. So if we always pick a vertex or an edge of degree at most 3 (provided by Lemma 1.11), this will take constant time. Note that contracting (respectively removing) an edge adjacent to a vertex (respectively a face) of constant degree means updating $O(1)$ flags in the representation, so the recursive call can be made in constant time.

So there remains to explain how to find the vertex or face of low degree provided by Lemma 1.11 in constant time. We will *amortize* this search, i.e., we will prove that the total time spent looking for these is $O(n)$, and thus it is $O(1)$ in average per round. In order to do that, we first compute a list L containing all the vertices and the faces of degree 3 of the initial graph. This takes linear time by traversing both the primal and the dual. The list L will be updated throughout the algorithm so that it always contains the list of vertices or faces that *may* have degree at most 3. When contracting or removing an edge, the degree of the four adjacent vertices and faces can change, so we add them all to the list L. Since there are $O(n)$ iterations, this process adds $O(n)$ elements to the list L throughout the algorithm.

Now, whenever we want to take an action, we look at the first element of the list L. If it does not exist anymore, or if it has degree more than 3, we remove it from the list, and continue. By Lemma 1.11, we always end up finding a vertex or a face. Since $O(n)$ vertices and faces were added to the list, we remove $O(n)$ of those throughout the algorithm. So in total, the search procedure takes $O(n)$ time, which proves our amortized complexity and finishes the proof of the algorithm. □

Note that this algorithm crucially relies on the fact that we work with non-simple graphs (even if the initial graph is simple, it becomes non-simple after contractions). This is why we rely on Lemma 1.11 and duality and not merely sparsity.

1.6 Minimum cut

Our second foray into algorithms for planar graphs concerns the computation of a minimum cut: given two vertices s and t, called ***terminals***, on a planar graph $G = (V, E)$, we want to compute the minimum set of edges X so that removing X from E separates s and t.

Theorem 1.12. *Let $G = (V, E)$ be a connected edge-weighted planar graph, and s and t be two distinct vertices of G. Then the problem of computing a minimum $s - t$-cut of G can be solved in $O(n \log^2 n)$ time, and even $O(n \log n)$ time if one uses Theorem 1.8.*

The basic idea of an efficient algorithm for min-cut on planar graphs is to look at it through the lens of duality: a cut on the primal graph separating s and t dualizes to a cycle in the dual graph separating the faces dual to s and t.

Proposition 1.13. *$X \subseteq E$ is an (s, t)-cut in G if and only if X^* contains the edge set of some cycle of G^* separating s and t.*

This proposition is considered obvious pretty much anywhere, but we will prove it, if only to emphasize that it does not hold on other surfaces (as usual, we will use the Jordan curve theorem).

Proof. The reverse direction is straightforward: if X^* contains a cycle of G^* separating s and t, then any path in G between s and t must cross this cycle, and thus X is an (s,t)-cut.

For the forward direction, we take X an (s,t)-cut in G, and choose C to be an inclusionwise minimal subset of X that is also an (s,t)-cut in G. We show that C^* is a cycle separating s and t. By minimality of C, each vertex of G can be connected to either s or t without taking edges of C, and the two cases are exclusive. We label vertices with "S" or "T" depending on which one they are connected to. Moreover, for any edge in C, its two endpoints cannot have the same label, and for any other edge, its endpoints have the same label. So if we look at a face f of G adjacent to an edge of C, the labels on the facial walk on the face alternate only when the edge is in C, and thus there is an even number of edges of C adjacent to f. So C^* is an ***Eulerian*** subgraph of G^*, that is, a subgraph where each vertex has even degree. Pick any cycle in that subgraph. By the Jordan curve theorem, it is separating, and since the faces on each side of each edge are labelled "S" and "T", it separates s and t. By minimality, C^* is that cycle. □

This proposition transforms a *combinatorial* problem into a *topological* one, namely, finding in the dual graph the shortest cycle enclosing a given face and not another given one. Even more topologically, if we *remove* the faces s^* and t^* from the dual graph, we obtain a surface homeomorphic to an annulus, and we look for the shortest cycle that goes around this annulus. However, this runs into an interesting technical issue: is s^* and t^* are adjacent, then removing s^* and t^* does not actually yield an annulus. Morally, this should not pose a problem: we just want to add an infinitesimally small buffer between s^* and t^* and we will be fine. One way to do this is to enlarge a tiny bit the set of curves that we look at. When working on the primal graph, we generally work with walks on the primal graph. When working on the dual graph, we work with walks on the dual graph, which correspond by duality to closed curves that are in ***general position*** with respect to G, i.e., they do not meet the vertices of G and cross the edges of G transversely. So our solution is to directly work in this setting of curves in general position with respect to G. The length of such a curve is defined to be the number of edges of G that it crosses. Note that this is a bit more general than just looking at walks on the dual graph: now we can define a pair of small curves in general position around s and t that are disjoint, even if s and t are adjacent in G. Yet from an algorithmic perspective, all the curves in general position can be pushed on the dual graph in a way that does not change the length, so any computation, for example shortest paths, can be made in the dual graph. In this new setting, Proposition 1.13 becomes:

Proposition 1.14. *Let γ be a simple closed curve in general position with respect to G, that separates s from t and that has minimal length among all such curves. Then the set of edges crossed by γ is a minimum (s,t)-cut in G.*

Recall that a simple closed curve on a sphere is an injective map $\gamma : \mathbb{S}^1 \to \mathbb{S}^2$..

Proof. Any path connecting s to t in G crosses γ, thus the set of edges crossed by γ is an (s,t)-cut. Conversely, a minimum (s,t)-cut dualizes to a cycle in G^* separating s and t, which corresponds to a simple closed curve γ in general position with respect to G, that separates s from t. □

Now, we remove a small disk around s and t, getting an annulus A, and a portion of the graph G embedded on this annulus as in Figure 7. How do we compute a shortest closed curve that goes around the annulus A? We can fix one point on each boundary, and compute a shortest path between these two points, and call it p. Then we show that some shortest closed curve does not cross p more than once, and thus can be found by a shortest path computation.

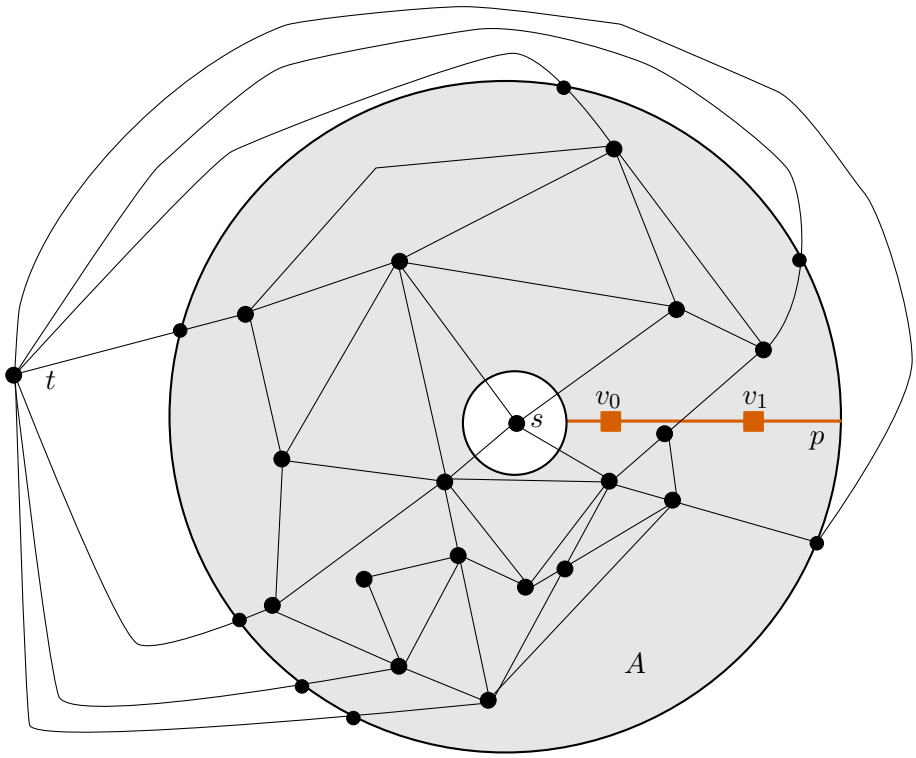

Figure 7: The annulus obtained after removing a small disk around s and t.

Lemma 1.15. *Some shortest closed curve separating the two boundaries of A is simple and crosses p exactly once.*

Proof. This is illustrated in the left of Figure 8. Let γ be such a curve. If we cut A along p, we obtain a topological disk B, where the path p got cut into two paths p' and p'' on the boundary. The curve γ, once cut on B, contains a simple path q connecting p' to p'', otherwise γ would not separate s from t. Now we reglue B along p, and connect the two endpoints q_1 and q_2 of q by running parallel to p. We obtain a closed curve that is simple, crosses p exactly once and is no longer than γ since γ was connecting q_1 to q_2 as well. □

From this we get a naive quadratic algorithm. We compute the shortest path p, which has some length k. We pick points $v_0, \dots, v_k$ on this path such that the subpath $[v_i, v_{i+1}]$ has length one. Then, cutting along p, each vertex v_i gets duplicated into v_i' and v_i'', and we compute all shortest paths between each v_i' and v_i''. Following all the lemmas, one of them is the dual of a minimal cut. Since shortest paths in the dual graph can be computed in time $O(n \log n)$, this takes $O(n^2 \log n)$ time.

We can speed this up doing some divide-and-conquering.

Lemma 1.16. *Let (x, y, z) be points on p, appearing in this order. When cutting A along p, these get duplicated into (x', y', z') and (x'', y'', z''). If γ_x and γ_z are disjoint shortest paths between x' and x'', respectively z' and z'', then some shortest path between y and y' does not cross γ_x nor γ_z.*

Proof. This is illustrated in the right of Figure 8. Say that γ_y crosses γ_x, then it crosses it in at least two points. Let a and b be the first and the last crossing points when going from y' to y''. Then we can replace whatever γ_y was doing between a and b with the subpath of γ_x between a and b (or more precisely, some shortest paths infinitesimally close to it). Since γ_x is

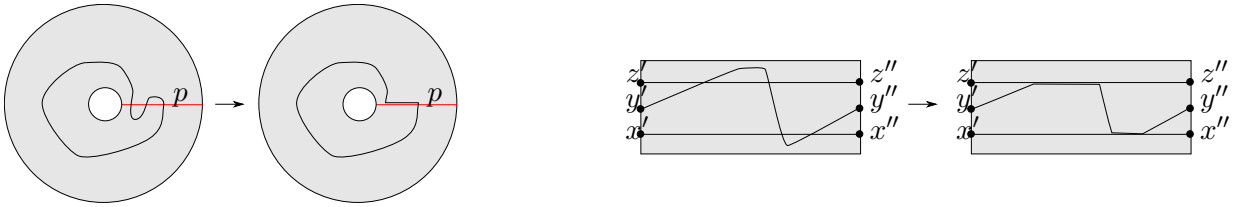

Figure 8: The two shortcutting arguments of Lemma 1.15 and Lemma 1.16.

a shortest path, the new curve is at most as along as γ_y. Doing the same for the crossings with γ_z concludes the proof. □

This suggest the following recursive approach. Having computed our shortest path p of length k between the two boundaries of A, if $k > 2$,

1. we pick a vertex $v := v_{\lfloor k/2 \rfloor}$, cut along p and compute on B a shortest path p between the two vertices v' and v'' corresponding to v,
2. we reglue B into A, and the shortest path p is a simple closed curve γ. We cut A along γ and get two annuli A_1 and A_2.
3. we recurse on A_1 and A_2 and output the shortest of the two solutions.

We stop the recursion when we reach one of the following two base cases for the recursion: (1) when $k \leq 2$, we can compute the shortest closed curve by brute forcing the problem in $O(n)$ time as in the quadratic algorithm, and (2) if there exists a face adjacent to both boundaries, we can compute a shortest cycle going through that face directly in time $O(n \log n)$.

Here again, this algorithm would be quite a bit more annoying to describe purely in the dual graph, as the shortest path p might be following the boundary of an annulus, and thus when cutting along γ in step 2 we do not obtain annuli. This can be dealt with by appropriately subdividing in the correct places, which, when thinking about it, is exactly what this algorithm does – but I believe that the description using curves in general position is more transparent (this "cross-metric" perspective is directly taken from Éric Colin de Verdière's notes).

To conclude the proof of the theorem, we establish the correctness and the complexity analysis:

Proof of Theorem 1.12. The algorithm terminates in $O(\log n)$ recursion levels since the length of the path p in the recursive calls shrinks by a half at each recursive call. The correctness follows from Proposition 1.14, Lemmas 1.15 and 1.16, since they prove that some minimal cut will be dual to the shortest cycle that our recursive calls will find.

The proof of correctness is not as immediate as one could expect, as recursive calls share a lot of structure with their parent: for example it looks like the same edge of G might be cut several times by the shortest paths in the recursion, and thus appear in several of the annuli. But note that if an edge e is cut into subedges $e_1, \ldots, e_\ell$, then in all the recursive calls involving the annulus between e_i and e_{i+1}, the recursion stops directly, since there is a face adjacent to both sides of the annulus. Therefore, only e_1 and e_ℓ actually lead to recursive subcalls, and thus at each level of the recursion, throughout all branches of the recursion tree, each edge only appears a constant number of times.

Therefore, at each level of the recursion, the annulus A is cut into k subannuli $A_1 \cup \ldots \cup A_k$ of total complexity $O(n)$, and thus solving the shortest path computations in all of them takes $O(n \log n)$ time (because the map $x \mapsto x \log x$ is concave).

Each computation in steps 1 and 2 of the algorithm takes time linear in the complexity of the annulus at this stage. There are $O(\log n)$ levels, and by the previous observations, each them costs $O(n \log n)$ time in total, so the total complexity is $O(n \log^2 n)$. Using linear-time shortest paths, this improve to $O(n \log n)$. □