CS762: Graph-Theoretic Algorithms

Felix Zhou[1]

Spring 2020
University of Waterloo

[1]from Professor Therese Biedl's Lectures

Contents

Chapter 1

Introduction

1.1 Motivation

Consider the minimum weight dominating set problem. It is NP-hard in general. However, we may be able to exploit the properties of certain graph classes and obtain polynomial time algorithms. For example, if the graph is planar, interval, chordal, or a k-partial tree.

1.1.1 Points of Inquisition

What graph problems become easier if the input graph belongs to graph classs $\mathcal{C}$?

How easy it is to recognize graphs in $\mathcal{C}$?

What are properties and/or equivalent characterizations of graphs in class $\mathcal{C}$?

1.2 Graph Assumptions

Unless stated otherwise, we will work with finite, simple, and connected graphs. In addition we can assume there is a sufficiently large lower bound on $n = |V|$.

By default, graphs are undirected. However, we may impose a "direction" for certain algorithms if it helps.

As in normal CS literature, we will let n or $n(G)$ indicate the number of vertices and $m, m(G)$ the number of edges.

We will assume graphs are stored with incidence lists. This allows vertex or edge addition in $O(1)$ time. Edge deletion is also $O(1)$ assuming we know where the edge is. However,

vertex deletion takes $\Theta(\deg(v))$ time.

1.3 Weighted Dominating Set in Paths

As a warm-up, notice that the weighted dominating set in paths is solvable using DP with the following formula

$$\delta(v_n, 1) = \text{weight of best dominating set in } \{v_1, \ldots, v_n\} \text{ including } v_n$$
$$\delta(v_n, 0) = \text{weight of best dominating set in } \{v_1, \ldots, v_n\} \text{ NOT including } v_n$$

The key observation is that if we do not include v_n in the dominating set, v_{n-1} must be included. On the other hand, if v_n is included, we can either include v_{n-1} or not. If we do, we can use $\delta(v_{n-1}, 1)$. Otherwise, since v_{n-1} is covered anyways, we can take one of $\delta(v_{n-2}, i)$ for $i = 0, 1$.

The recursion is given as

$$\delta(v_n, 0) = \begin{cases} \infty, & n = 1 \\ \delta(v_{n-1}, 1), & n \neq 1 \end{cases}$$
$$\delta(v_n, 1) = \begin{cases} w(v_1), & n = 1 \\ w(v_2), & n = 2 \\ w(v_i) + \min\{\delta(v_{n-1}, 1), \delta(v_{n-2}, ?)\} \end{cases}$$
$$\delta(v_n, ?) = \min\{\delta(v_n, 1), \delta(v_n, 0)\}$$

Part I

Planar Graphs

Chapter 2

Planar Graphs

We will explore the following tools which make problems easier in planar graphs:

Graph Properties Structural properties such as the forbidden $K_5, K_{3,3}$-minors

Planar Embeddings We can traverse a fixed embedding in a special way

Dual Graph For a fixed embedding, this gives rise to a dual, planar graph which can be useful

Separator Theorems Any planar graph can be "evenly" divided by removing $O(\sqrt{n})$ vertiecs. This gives rise to divide-and-conquer techniques

2.1 Definitions

2.1.1 Planar Drawing

Definition 2.1.1 (Planar Drawing)
A planar drawing Γ assigns each vertex to a point in $\mathbb{R}^2$ and each edge to a curve in $\mathbb{R}^2$ which begins at the endpoints of the edge. The interiors of the images of edges are pair-wise disjoint.

A planar drawing Γ separates $\mathbb{R}^2$ into regions.

Definition 2.1.2 (Face)
A maximal connected region of $\mathbb{R}^2 \setminus \Gamma(G)$.

Notice that there is exactly one unbounded face as G is finite. This is referred to as the outer face. All other faces are interior faces.

Proposition 2.1.1

1. All subgraphs of a planar graph is planar
2. Adding edges do not preserve planarity in general but if the endpoints are on the boundary of some face, it does
3. Vertex contraction does not necessarily preserve planarity but if both vertices are on one face, it does
4. Edge contraction always preserves planarity

If follows if G is planar, then any minor of G is planar.

For all graphs obtained in the proposition above, we refer to the "natural" new drawing as the planar drawing induced by the planar drawing of G.

2.1.2 Planar Embedding

Drawings are analytic properties and thus not very useful in the algorithmic sense.

Definition 2.1.3 (Rotation System)
Any drawing Γ defines for each vertex the clockwise order of incident edges. The set of all these orders is a rotation system of the graph.

Notice that we can read facial circuits directly from the rotation system. Start at some $e = uv \in E$ and take the next edge $e' = vw$ after e in the edge order at v, then take the edge after e' to be the edge wx after vw in the rotation system of w.

Definition 2.1.4 (Combinatorial Embedding)
A rotation system together with one face indicating the outer face.

Although non-planar graphs can also have combinatorial embeddings, we will refer to planar embeddings and combinatorial embeddings interchangeably since we do not study non-planar embeddings.

2.1.3 Multiple Planar Embeddings

Lemma 2.1.2
Let G be connected, planar with drawing Γ. Let F be one of the faces.
There is a planar drawing of G with the same rotation system as Γ and F being the outer face.

We can prove this the classical way with stereographic projection. However, the following algorithmic proof is less terse.

Proof
Let p be some point in the interior of F and r a ray emanating from p which does not intersect any vertices.

If r crosses no vertices, F is already on the outer face. Otherwise, take e to be some edge on the outer face intersecting r. Since e is on the outer face, we can easily reroute it so it does no cross f in a way which preserves the rotation system.

We can then proceed by induction on the number of edges r crosses.

2.2 Exploting the Planar Embedding

Problem 1 (Menger)
Given directed graph G and vertices s, t, find the maximum set of vertex-disjoint paths from s to t.

The Menger problems can be solved in general with maximum-flow techniques. However, there is faster solutions in planar graphs.

2.2.1 Right-First Search

This is a specific implementation of depth-first search where ties are broken by the planar embedding.

```
Right-First-Search(G, v0, e0):
  discovered = [None for v in V]

  def recurse(v, e):
    discovered[v] = e # discovered coming from e
    for e_i in e_1, ..., e_d:
    # incident edges at v
    # enumerated in ccw order with e_d = e
      v, w = e_i
      if discovered[w] is None:
        recurse(w, e_i)

  recurse(v0, e0)
  return discovered
```

Definition 2.2.1 (Right-Most)
Let π be a path from s to some other vertex x, both of which are on the outer face. The part to the right of π contains everything bounded by π and the counter-clockwise outer face path from s to x. We say π is right-most if any other path $\hat{\pi}$ from s to x is in the left region of π.

Lemma 2.2.1
Let G be plane with vertices s, x on the outer face.
Run right first search starting at s with respect to the outer-face edge of s going the clockwise order. If x is reached during the exploration, the resulting path π is rightmost.

Proof
Let P be any other sx-path and consider the first vertex v where P, π use different outgoing edges e_P, e_π.

Then, consider the counter-clockwise order at v. We must have the incoming edge on both paths, e_π, then e_P. This is purely by the definition of right-first search.

So P diverges "left" from π at v. It may intersect π again at some vertex before x but by the same argument, it can only diverge "left".

2.2.2 The Menger Problem in st-planar graphs

Definition 2.2.2 (st-Planar)
A graph is st planar if it has an embedding in which s, t are both on the outer face.

We will study the Menger problem for st-planar graphs.

Pseudocode

1) Temporarily insert the edge $e_s = ts$ on the outer face
2) Run right-first search starting at s with respect to e_s
3) If this does not reach t, there are no (more) st-paths
4) If it does reach t, output the path it used and remove any vertex which was used not including s, t
5) Repeat 2)

Analysis

Let $\pi_1, \ldots, \pi_\ell$ be the paths at were found by the algorithm. Let $P_1, \ldots, P_k$ be any other set of vertex-disjoint directed st-paths. We show that $k \leq \ell$.

Order $P_1, \ldots, P_k$ such that P_i is in the right region of P_{i+1} for all i. This works since the paths are vertex-disjoint.

Since π_1 is rightmost, deleting π_1 and everything to the right will not delete any vertex of $P_2, \ldots, P_k$. Repeating this argument shows that P_i is to the left of π_i for $i = 1, 2, \ldots$.

In particular, we have $k \leq \ell$ as desired.

Notice that each subsequence call ot right-first search only viists edges that have not previously been visited. It follows that the total run time is $O(m + n)$.

2.3 Dual Graphs

For every graph with a fixed rotation system, we can define a dual graph capturing the emdbedding in a graph structure.

Definition 2.3.1 (Dual Graph)
The dual graph $G^* = (V^*, E^*)$ for a graph $G = (V, E)$ with a fixed embedding is as follows

$$V^* = \{F : F \text{ is a face of } G\}$$
$$E^* = \{v_F v_{F'} : F, F' \text{ is incident to the faces } F, F' \text{ of } G\}$$

Observe that each $v \in V$ becomes a face in the dual. The edges incident with v as a face in G^* rise directly from those incident with v in G.

Lemma 2.3.1
Let G be a connected plane graph.

$$(G^*)^* = G$$

It is not clear what the definition of a dual for a disconnected graph should be. The ambiguity is mosty due to the unbounded face. We can either take the union of duals of each connected component or directly apply the definition using faces.

2.3.1 Computing & Storing G^*

Lemma 2.3.2
The dual graph of any plane connected G can be computed in $O(m + n)$.

Proof
The idea is to have the incidence list simultaneously store incidences for the primal and dual graph.

Start with the incidence list of G. Each vertex has a doubly linked list of references to a doubly linked list of edges. Each edge in turn has two references to the two vertices to which it is incident.

For each $uv \in E(G)$ in the edge list, add two initially null references to vertices in G^* with $v \to w, w \to v$. We impose a direction to indicate that this edge is incident with the face which is to the right as we walk from the head to tail.

Now, while there is an edge of e for which one of the new references are null, say $v \to w$, define a new face F and append it to a list of faces (vertices of G^*). Trace the facial circuit of F by start at the edge vw and tracing the counter-clockwise next edge after vw at w, etc. For each edge e in the facial circuit, we append a new reference to it in the list for F and fill in the null reference in e to F as well.

Notice that each edge is traversed twice so the overall run time is linear.

2.3.2 Algorithmic Implications

Dual graphs are useful for rephrasing a problem in the primal.

Definition 2.3.2 (Polygon Mesh)
A way to approximate surfaces by small polygons.

Notice that if the surface has genus 0, the polygon mesh is a planar graph.

The most common kind of mesh is triangular, but there are some situations where it is helpful to have a quadrangular mesh instead. How can we convert a triangular mesh to quadrangular one?

In other words, given a plane connected G where every face is a triangle, groups faces into pairs that share an edge. In other words, we want a set of edges E' such that every face is incident to exactly one edge in E.

In the dual graph, this is precisely the problem of finding a perfect matching. Maximum matchings can be found efficiently for any graph. Moreoever, in the special case of meshes, there is always a perfec matching whicH can be found in linear time.

2.4 Euler's Formula

Theorem 2.4.1 (Euler's Formula)
Let G be a connected, but not necessarily simple plane graph.
Then

$$n - m + f = 2$$

where f is the number of faces.

Proof
By induction on the number of faces with the case of $f = 1$ implying G is a tree.

Corollary 2.4.1.1
Let G be planar. Any planar drawing of G has the same number of faces.

Corollary 2.4.1.2
Any simple connected planar graph with at most 3 vertices has at most $3n - 6$ edges.

Proof
Every edge is incident with at most 2 edges and every face is incident with at least 3 edges. Applying this with Euler's Formula gives the result.

Corollary 2.4.1.3
Every simply planar graph has a vertex of degree at most 5.

Lemma 2.4.2
Every simple planar triangle-free graph with at least 3 vertices has at most $2n - 4$ edges.

Proof
The proof is identical, except now every face is incident with at least 4 edges.

Corollary 2.4.2.1
$K_5, K_{3,3}$ are not planar.

Theorem 2.4.3
plane G is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$ as a subgraph (topological-minor).

2.4.1 Algorithmic Implications

Dijkstra's algorithm finds shortest path trees in $O(m + n \log n)$ time with Fibonacci heaps and $O((m + n) \log n)$ time if we use binary heaps. However, since $m \in O(n)$ for planar graphs, there is no need to use the more complicated Fibonacci heaps for planar graphs.

Definition 2.4.1 (Minimum-Degree Order)
$v_1, \ldots, v_n$ such that v_i has minimum degree in $G[v_1, \ldots, v_n]$.

Lemma 2.4.4
Every planar graph has a vertex order $v_1, \ldots, v_n$ such that each v_i has at most 5 predecessors. It can be found in linear time.

Proof
Let $v_1, \ldots, v_n$ denote a minimum degree order. This clearly suffices since any subgraph of G is also planar and thus has minimum degree at most 5.

Colouring

Using the minimum degree order of a planar graph, the greedy colouring algorithm uses at most 6 colours.

Some additional work garanteeds a linear time 5-coloring algorithm.

Theorem 2.4.5 (Four-Colour)
Every planar graph has a vertex-coloruing with at most 4 colours.

Testing Adjacency

Recall that if a graph is stored with incidence lists, then adjacency testing between $v \neq w \in V$ takes

$$\Theta(\min\{\deg(v), \deg(w)\})$$

time.

Theorem 2.4.6
Any planar graph can be stored using $O(n)$ space, so adjacency testing is $O(1)$.

Proof
Compute a minimum-degree order and store the predecessors and index in the ordering for each vertex.

To compare v_i, v_j simply look at the constant number of predecessors of $v_{\max(i,j)}$.

Clique

For each integer $k = 5, 4, \ldots, 2$, traverse the minimum-degree ordering in reverse. For each v_i and each possible subset $C \subseteq \text{pred}(v)$ such that $|C| = k - 1$, test if C is a clique through brute force $O(1)$ adjacency testing. Stop immediately upon finding a clique and output $C \cup \{v_i\}$ as a maximal clique.

Chapter 3

Problems in Planar Graphs

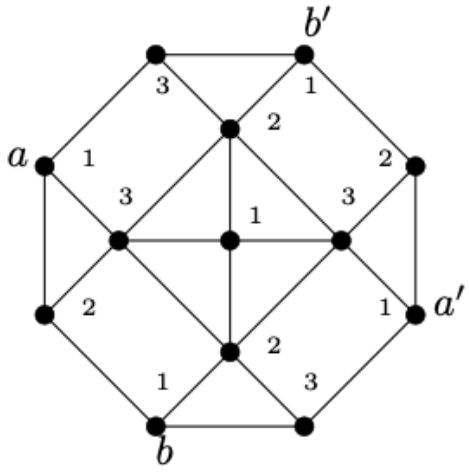
3.1 NP-Hard Problems in Planar Graphs

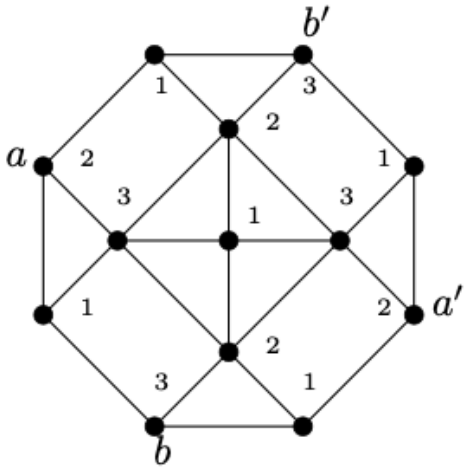
We will introduce soem problems which are NP-hard even in planar graphs. The proofs of NP-hardness typically involve one of two technique. The first technique involves developing a crossing gadget which shows that problems in general graphs reduce to planar ones. On the other hand, if the proof of NP-hardness in general graphs relies on a reduction from an NP-hard problem in planar graphs, the reduction preserves planarity, then the same problem is NP-hard for planar graphs.

3.1.1 Coloring

We first show that 3-coloring is NP-hard.

Lemma 3.1.1
In any 3-coloring of the gadget, $c(a) = c(a')$ and $c(b) = c(b')$, where c denotes the coloring function.
Moreoever, there is a 3-coloring of the gadget where $c(a) = c(b)$, and there exists a 3-coloring of the gadget where $c(a) \neq c(b)$.





Proof
We may assume without loss of generality that the center vertex is colored 1. Its neighbours must then be colored 2, 3 alternatingly. The colors of the outer face vertices are then also fixed between one of two states illustrated above.

3-Coloring Reduction

We want to show

$$\text{3-coloring} \preceq \text{3-coloring in planar graphs}$$

Let G be an arbitrary graph and create a drawing (with crossings) that is x-monotone. In other words, G the vertices are all on the x-axis with edges being half-circles. This is so we can classify each endpoint of edges as "left" and "right" endpoint.

Create a new graph G_1 by replacing crossings with the gadget defined above.

Proposition 3.1.2
G is 3-colorable if and only if G_1 is 3-colorable.

Proof
Apply the lemma for each gadget subgraph and see that a 3-coloring is transferable between G, G_1.

3.1.2 Planar 3-SAT

Now, we try our hand at the other method of proving a problem is NP-hard in planar graphs.

Definition 3.1.1 (3-SAT Graph)
Given an instance of 3-SAT, the corresponding SAT-graph is defined as follows

1. create one vertex for every variable x_i
2. create one vertex for every clause c_j
3. create an edge x_i, c_j if and only if c_j contains the literal x_i or $\bar{x}_i$
4. create a cycle $x_1, x_2, \ldots, x_n, x_1$ of edges

Problem 2 (Planar 3-SAT)
An instance of 3-SAT wher the associated 3-SAT graph is planar.

Theorem 3.1.3
3-SAT reduces to Planar 3-SAT, thus Planar 3-SAT is NP-hard.

3.1.3 Independent Set

Assuming the NP-hardness of Planar 3-SAT, it is straightforward to prove that Vertex Cover, Independent Set, and Hamiltonian Cycle are all NP-hard in planar graphs by inspecting the original reduction from 3-SAT and showing (with easy modifications) that it preserves planarity.

Theorem 3.1.4
3-SAT reduces polynomially to Independent Set.

Proof
Let $\mathcal{I}$ be an instance of 3-SAT with N variables and M clauses. Define an instance $\mathcal{I}'$ of independent set.

For any variable x_i define the variable cycle consisting of $2 \deg x_i$ vertices (number of times it appears in a clause). Label the vertices of this cycle alternatingly as $x_i, \bar{x}_i$. For every clause c_j, define a clause-triangle consisting of three vertices forming a triangle. Label the three vertices with the three literals in c_j, and connect them to an OPPOSITE literal in the corresponding variable-cycle.

Clearly, G' is polynomial in n, m.

Observe that any independent set I in G' contains at most one vertex per clause triangle and at most half the vertices of each variable cycle. We surely have $|I| \leq M + \sum_i \deg x_i$.

We claim the 3-SAT instance $\mathcal{I}$ is satisfiable if and only if G' contains an independent set of size at least $M + \sum_i \deg x_i$.

Assume that $\mathcal{I}$ is satisfiable and define I as follows. For each x_i, add $\deg(x_i)$ vertices from the variable-cycle to i by adding all x_i vertices if x_i is TRUE. Otherwise, add $\bar{x_i}$ vertices. For each clause c_j, at least one vertex in the clause cycle ℓ_j is true. Add to I the vertex corresponding to ℓ_j. By construction, its neigher in the variable clause x_{ℓ_j} cannot be in I since $\bar{x_{\ell_j}}$ is added.

Now assume we have an independent set of the specified size. I necessarily uses one vertex from each clause-triangle and $\deg x_i$ vertices from the variable-cycles. This induces an assignment satisfying the instance $\mathcal{I}$ of 3-SAT.

Planar Reduction

Take an instance of Planar 3-SAT and apply the exact same reduction.

Proposition 3.1.5
The instance of Independent Set consists of a planar graph.
Specifically

$$\text{Planar 3-SAT} \preceq \text{Independent Set in planar graphs}$$

Proof
Fix a planar embedding of G_{SAT}, the 3-SAT graph corresponding to an instance $\mathcal{I}$ of Planar 3-SAT.

Modify G_{SAT} by vertex-capping: replace a vertex of degree e by a d-cycle of vertices of degree 3, and redistribute the incident edges to the vertex of the d-cycle. Apply the same technique to the clauses. Finally, achieve the labelling in the general reduction by deleting the edges of the cycle of variables and subdividing each each in some variable-cycle. With some possible rerouting of clause-variable edges to the newly created vertices, we have the exact reduction from above.

Since vertex-capping, edge deleting, and edge subdivision preserves planarity, we are done.

Corollary 3.1.5.1
Independent Set is NP-hard even in planar graphs with maximum degree 3.

Proof
Notice that the graph used in the reduction above has maximum degree 3.

We also immediately have that Vertex Cover is NP-hard in planar graphs of maximum degree 3. This is due to the fact that C is a vertex cover if and only if $V - C$ is an independent set.

Another construction starting from Planar 3-SAT but using a different gadget shows that Hamiltonian cycle is NP-hard in planar graphs.

3.2 Maximum-Flow

Definition 3.2.1 (Flow Network)
A simple graph G with two distinct vertices s, t, and a capacity function $c : E \to \mathbb{R}^+$.

We say the flow network is undirected if $c(i, j) = c(j, i)$ for all $ij \in E$ and directed otherwise.

Definition 3.2.2 (Network Flow)
A function $x : V \times V \to \mathbb{R}^+$ such that

(i) $x(ij) > 0 \implies ij \in E$

(ii) $x(ij) - x(ji) \leq c(ij)$ (capacity constraint)

(iii) $x(\delta(v)) = x(\delta(\bar{v}))$ for all $v \neq s, t$ (balance constraint)

One should think of an edge as two directed edges in opposing directions. The capacity constraint can be thought of as "net" flow on that edge since we can have units of flow in both the forward and reverse directions.

Definition 3.2.3 (Flow Value)
The value of a flow x is

$$x(\delta(s)) - x(\delta(\bar{s}))$$

Notice that the value of a flow is precisely the units flowing out of s and never returning.

3.2.1 *st*-Cuts

Definition 3.2.4 (*st*-Cut)
A partition $(S, \bar{S})$ of the vertices such that

$$s \in S, t \notin S$$

Definition 3.2.5 (Cut Value)
The value of a cut $(S, \bar{S})$ is

$$x(\delta(S))$$

Lemma 3.2.1
If x is a flow and $(S, \bar{S})$ is an st-cut then

$$x(\delta(s)) - x(\delta(\bar{s})) \leq x(\delta(S))$$

Proof
Unwrap the definitions.

Theorem 3.2.2 (Maximum-Flow Minimum-Cut)
For any network, the value of the maximum st-flow is equal to the value of the maximum st-cut.

3.2.2 Undirected Flow in st-Planar Graphs

Ford-Fulkerson solves maximum flow in pseudo-polynomial time. There are polynomial time algorithms, but can we do better?

Intuitively for planar graphs, a minimum st-cut is a simple, closed, Jordan curve which separates s, t and does not cross vertices. We can nicely redefine such curves by observing that they correspond to cycles in the dual graph.

Consider G^* for plane G and define for every dual edge weight $w(e^*) = c(e)$, the capacity of the corresponding edge in G.

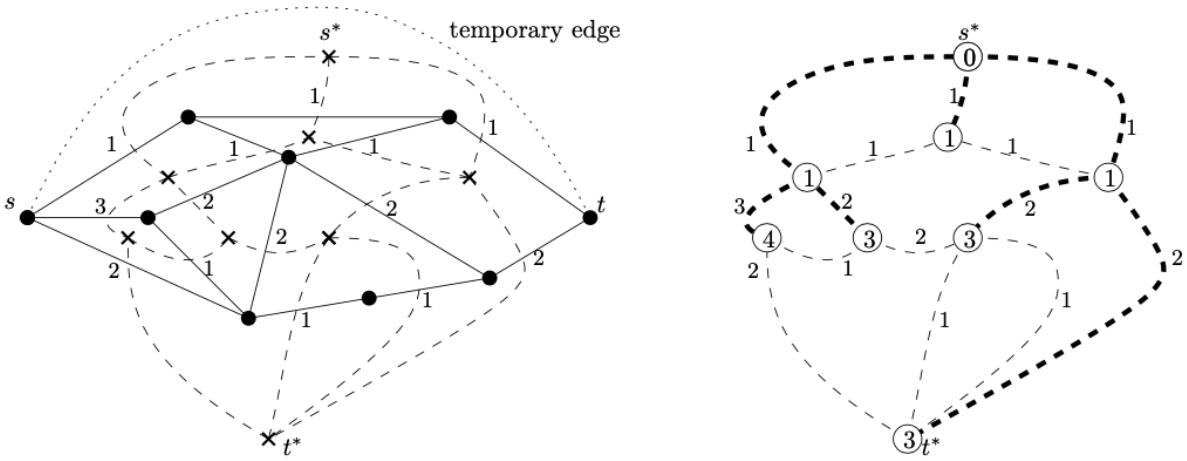
Lemma 3.2.3
A minimum cut in an indirected planar network (G, s, t) corresponds to a shortest-weight cycle in the dual graph which separates s, t.

Proof
We prove this by finding a flow whose value is the same as the weight of the cycle.

Restricting to st-Planar Graphs

Let us further assume that s, t lie on the same face. By our work earlier, s, t might as well be on the outer face.

Definition 3.2.6 (Dual Network for *st*-Planar Graph)
Add an edge st in the primal network to obtain G'. Take the dual graph of G' and delete the edge between the two faces s^*, t^* incident to the new edge st. Finally, for any other dual edge e^*, set $w(e^*) = c(e)$.



Lemma 3.2.4
The length of shortest s^*t^*-path is the value of maximum flow in the primal network.

Proof
We first construct a flow candidate from the shortest paths. Then we show it is indeed a flow. Finally, we conclude by finding a cut of the same value. By the max-flow min-cut theorem, this suffices to show the result.

Flow Construction Assume we know have the shortest path tree rooted at s^*. For $a^* \in V(G^*)$, denote by $d(a^*)$ the distance from s^* to a^*.

Assume $ij \in E(G)$. Let ℓ^* be the face that is to the LEFT of $i \to j$. Let r^* be the face to the RIGHT of $i \to j$.

Define the flow for edge $i \to j$ to be

$$x(ij) = \max(0, d(r^*) - d(\ell^*))$$

and observe that by definition

$$x(ji) = \max(0, d(\ell^*) - d(r^*))$$

We now claim x is a valid flow.

Capacity Constraints Fix an edge ij and let ℓ^*, r^* be as above for $i \to j$. It must be true that

$$d(\ell^*) + w(\ell^*, r^*) \geq d(r^*)$$

This implies that

$$d(r^*) - d(\ell^*) \leq w(\ell^*, r^*) = c(ij)$$

We also know $0 \leq c(ij)$ and thus

$$x(ij) = \max(0, d(r^*) - d(\ell^*)) \leq c(ij)$$

as desired.

Balance Constraints Fix $i \neq s, t$ and let $j_0, \ldots, j_{k-1}, j_k = j_0$ be its neighbours in clockwise order. Let f_ℓ be the face to the left of $i \to j_\ell$, so the face to the right is $f_{\ell+1}$ (modulo addition k).

By construction $x(i, j_\ell) = \max(0, d(f_{\ell+1}) - d(f_\ell))$. We have

$$\begin{aligned} x(\delta(i)) - x(\delta(\bar{i})) &= \sum_{\ell=1}^{k} \max(0, d(f_{\ell+1} - d(f_\ell))) - \max(0, d(f_\ell) - d(f_{\ell+1})) \\ &= \sum_{\ell=1}^{k} d(f_{\ell+1}) - d(f_\ell) \\ &= 0 \end{aligned}$$

A similar calculation shows that the value of x is

$$d(t^*) - d(s^*) = d(^*)$$

as $d(s^*) = 0$. This is due to the fact that we deleted the edge s^*t^* and thus there is a single equation which does not cancel out

Finding the Cut Let P be the shortest path from s^* to t^*. Its length is $d(t^*)$. Clearly

$$P \cup \{s^*t^*\}$$

would give us a cycle separating s, t.

Let S be all the vertices which are inside this cycle. Thus $(S, \bar{S})$ forms a cut. Consider E_S, the edges of G with exactly one endpoint in S. Notice that these are precisely the dual edges of P.

The value of this cut is precisely $d(t^*)$. So we have found a valid flow x and a corresponding cut $(S, \bar{S})$ which the same values, concluding the proof.

To summarize, we compute the dual network (linear), then run a shortest path algorithm (linear, complicated).

Theorem 3.2.5
A maximum flow can be found in $O(n)$ time.

Chapter 4

Planarity Testing

In all the algorithms for planar graphs so far, we assumed we had a planar embedding. To have any hope to putting them to use, we need a way to recognize planar graphs.

4.1 Bush Forms

Lemma 4.1.1
Let G be planar and $(V', \bar{V}')$ be a cut for which $G[V \setminus V']$ is connected. Then $G[V']$ has a planar embedding where all endpoints of edges in the cut are on the outer face.

Proof
Contract all edges within $V \setminus V'$. This shrinks V' to a single vertex c. Bring this vertex to the outer face and observe that this forces the neighbours of c to the outer face of $G[V']$.

Rather than deal with cuts directly, we work with bush forms. These add parts of the edges of a cut to the graph.

Definition 4.1.1 (Bush Form)
The bush form $B(V')$ of $V' \subseteq V$ consists of the graph $G[V']$ as well as one leaf vertex for every edge e in the cut

$$(C', \bar{V}')$$

adjacent to the endpoint of e in V'.

It is natural to label these vertices with the other endpoints of e that is not in V'.

We say a bush form is valid if all leaves are on the outer-face.

Lemma 4.1.2
If G is planar and $G[V - V']$ is connected, then $B(V')$ has a valid embedding.

This follows directly from the previous lemma.

4.2 The Algorithm by Haeupler & Tarjan

This method is based on a depth-first search traversal.

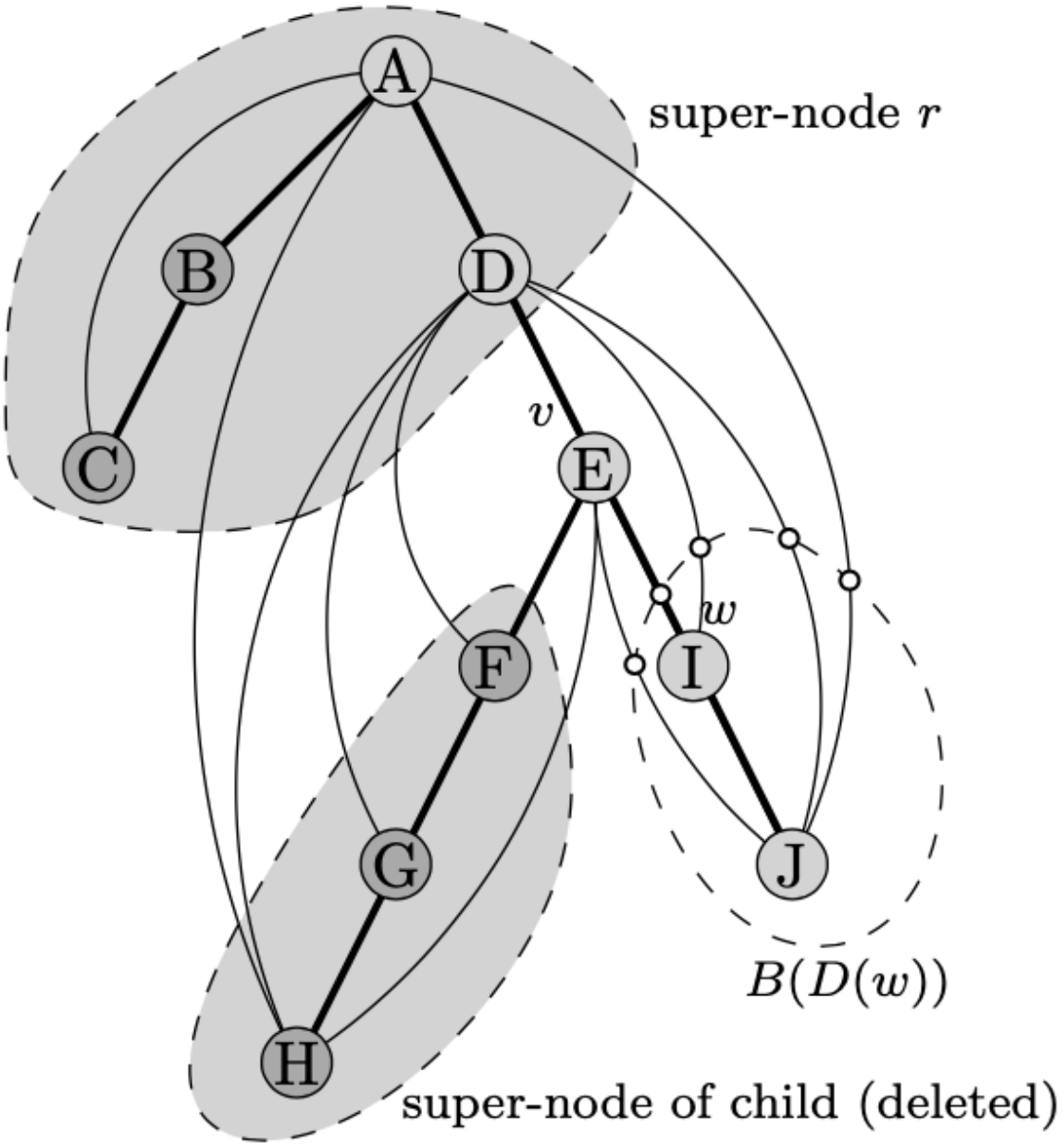
4.2.1 Depth-First Search & Bush Forms

Recall that a rooted DFS-tree T partitions the edges of G into either tree-edges or an edge connecting an ancestor with a descendant in the tree. We write $D(v)$ to denote the descendants of v, with the convention of including v.

Lemma 4.2.1
If G is planar, then for any tree edge vw with v the parent, the bush form

$$B(D(w))$$

has a valid embedding where the leaves L_v that are labelled v are consecutive on the outer-face.



Proof

Observe that $B(D(w))$ has valid embeddings since $V \setminus D(w)$ is connected via the DFS-tree.

Fix a planar drawing of G and contract all tree-edges except those within the subgraph induced by $\{u\} \cup D(w)$.

If v is not the root, then this creates one super-node r with the root. It also creates one super-node for each child $c \neq w$ of v.

Delete any super-node such a child as this does not affect edges relevant for the bush form $B(D(w))$. This is because there are no edges between $D(c), D(w)$.

Modify the resulting planar drawing to bring edge rv to the outer-face (or just v if it is the root). This means that the only edges in the cut $(D(w), \overline{D(w)})$ have endpoints in r or v. By planarity, their other endpoints in $D(w)$ must then be consecutive as desired.

After vertex-capping both r, v and deleting the edges in the cycle created by the vertex-cappings (and some more), this gives the desired drawing of $B(D(w))$.

4.2.2 High-Level Idea

We run a DFS and maintain a number of data structures that help determine if $B(D(w))$ satisfies the conclusion of the lemma. Along the way, we implicitly build a planar embedding of $B(D(w))$ so that if the lemma holds at the end, we have constructed a planar embedding of the entire graph.

Since the appropriate data structures are built while traversing, we index them with a time stamp $t \geq 1$. Time advances when we explore or retreat from an edge.

At any time a vertex is one of three stages: Undiscovered, Finished (retreated from it), and Active (neither of the previous two). The actice vertex always form a path from the root to some vertex, so any vertex in T has at most 1 active vertex at any given time.

Let $D(w, t)$ be the descendants of w visited by time t. Notice that $D(w, t) = D(w)$ if t is the time when we retreat from to from w to v, its parent. So the all-important lemma restricts $B(D(t))$ at this time.

Testing whether $B(D(w, t))$ has a suitable embedding is the key ingredient. For this, we implicitly store all possible planar embeddings. This is done using the PQ-tree data structure.

4.2.3 PQ-Trees

The goal is to store permissible permutations of a finite set X. A PQ-Tree is a rooted tree with elements of X at the leaves. A permissible permutation is obtained by listing the leaves in order from left to right.

We are able to reorganize the leaf-order with the following rules: Interior nodes are either P or Q nodes. A Q node indicates that its entire subtree can be reversed, while a P node indicates that its children can be placed in any order.

Reductions

PQ-trees have only one non-trivial operations. Given a PQ-tree T and a constraint $I \subseteq X$, decide if there exists a permutation expressed by T in which the items in I are consecutive. If there is one, return a PQ-tree which expresses all such permutations.

Lemma 4.2.2
PQ-trees can be implemented such that a reduction for constraint I takes $O(|I|)$ amortized time.

Lemma 4.2.3
Let X be a finite set and $\mathcal{I}$ a finite set of consecutive constraints. We can test whether there is a permutation of X such that for any $I \in \mathcal{I}$, the elements are consecutive in

$$O\left(|X| + \sum_{i \in \mathcal{I}} |I|\right)$$

time.

Proof
Initialize a PQ-tree to be a P-node with X as leaves. Initialize $\mathcal{I}' = \varnothing$.

Perform reductions for each $I \in \mathcal{I}$ one by one and add to $\mathcal{I}'$. Either until we have exhausted all of $\mathcal{I}$ or report that no suitable permutation exists.

4.2.4 Data Structures

We now return to the idea of performing a depth-first search while storing information to construct bush forms $B(D(w,t))$ at the time when we retreat from w.

To save space, we store bush forms (PQ-trees) only at active nodes v. Furthermore, the PQ-tree for v only includes the finished descendants of v.

These PQ-trees consist of non-leaf vertices which are v or finished descendants of v and discovered leaf vertices.

At the active child w of v, there may be other parts of the bush form $B(D(v,t))$ but these are NOT stored with v and instead stored with at some vertex in $D(w)$. When we later retreat from w, the bush form of w will be merged into the bush form of v, hence completing it by the time we retreat from v and actually need $B(D(v))$.

Descendants Consisting of Finished Children

Let $c_1, \ldots, c_d$ be the children of v that are finished at time t. So $D(c_i, t) = D(c_i)$ for $i \in [d]$. Define

$$V'(v,t) := \{v\} \cup \bigcup_{i=1}^{d} D(c_i)$$

Let $PQ(v,t)$ be a PQ-tree the encodes all valid embeddings of the bush form $B(V'(v,t))$. We label the leaves of $PQ(v,t)$ with the corresponding edges in the leaf-edges of $B(V'(v,t))$.

The bush form $B(V'(v,t))$ always includes the edge vp among its leaf edges if v is not the root and has parent p. Among the multiple PQ-trees that could represent $B(V'(v,t))$, we additionally impose that the leaf for vp is a child of the root, and the root-node is a P-node.

The Active Child

Recall that v has at most one active child w.

We store a list $S(v,t)$ of edges (v,x) where the other endpoint x is in $D(w,t)$ (a visited descendant of the active child w of v). List $S(v,t)$ is empty if v currently has no active child.

$S(v,t)$ helps us reformulate the previous lemma in terms of PQ-trees and these data structures.

Proposition 4.2.4
Consider the time t when we retreat from w to its parent v. If G is planar, then a reduction of $PQ(w,t)$ with respect to constraint $I := S(v)$ is successful.

Proof
Since we are about to finish w, all children are finished and $V'(w,t) = D(w)$. As $D(w,t) = D(w)$ as well, $S(v,t)$ contains all edges incident with v with the other endpoint in $D(w)$.

The claim follows from our previous lemma as $PQ(w,t)$ stores $B(V'(w,t)) = V(D(w))$.

4.2.5 Summary

Initialization

Initially only the root r has been visited. It has no active child, so $S(r) = \varnothing$. It also has no finished children, so $V'(r) = \{r\}$. Then $PQ(r)$ has no leaves.

We create a single root-node in $PQ(r)$ as a placeholder so we can add leaves as need later. Notice we do not specify $t = 1$.

There are 3 kinds of updates: advancing along a tree-edge (discover new vertex), traversing a non-tree-edge (vertex already discovered), and retreating along a tree-edge (vertex is finished).

Tree Edge Update

Suppose we advance along the tree edge vw where w was discovered.

Initialize $PQ(w)$ to be a single P-node with a single child labeled with leaf edge vw.

v had no actice child and now has one. Set $S(v) = \{vw\}$. Finally, as w has been visited, the edge vw must be considered for the bush form $B(V'(v))$, thus add it as a child of the root-node of $PQ(v)$ (for consistency but not strictly necessary).

None-Tree Edge Update

Suppose we explore a non-tree edge wv, where both w, v are active and w was a descendant of v.

Since w is a descendant of v and both are active, w is a proper descendant of the active child of v. Add the newly discovered edge vw to $S(v)$.

Likewise, vw is a leaf edge of $V(V'(v))$ and of $V(V'(w))$, so in both $PQ(v)$ and $PQ(w)$, add a leaf labelled vw and make it a child of the root (for consistency but not strictly necessary).

Tree Edge Retreat Update

Suppose we retreat along a tree edge vw with v being the parent. So w was active and now becomes finished. $V'(v)$ now includes $D(w)$.

First, let I be the leaves in $PQ(w)$ that are labelled with entries from $S(v)$. Since edges are cross-linked, we can find these in $O(|S(v)|)$ time. Reset $S(v) = \varnothing$ as we no longer need it.

Do a PQ-tree reduction at $PQ(w)$ with respect to I. By our previous proposition, if this fails then G cannot be planar.

Assuming this reduction is succesful, let $\hat{PQ}(w)$ be the resulting PQ-tree. Delete all leaves of $I - vw$ from $\hat{PQ}(w)$ and rearrange $\hat{PQ}(v)$ so that the leaf vw is again a child of the root-node. Call this result $PQ'(w)$.

These operations are not standard for PQ-trees but are easy to implement.

Finally, merge $PQ'(w)$ into $PQ(v)$ by making its root a child of the leaf vw which exists in $PQ(v)$. Notice the leaves labelled $S(v)$ in $PQ(v)$ are no longer leaves so we delete them.

Moreover, any node of $PQ(v)$ with only one child should be contracted into this child (ie the now-non-leaf node vw should be contracted into the previous-root node $PQ'(w)$).

Theorem 4.2.5
Planarity testing can be done in $O(m)$ time.

Proof
The key observation is that each edge vw is in $S(v)$ only once, so even though reductions are expensive, the amortized time is linear.

4.2.6 Final Thoughts

The key insigh of Haeupler and Tarjan was to add vertices in finishing order. This means the rest of the graph is connected.

Chapter 5

Triangulated Graphs

Recall that every simple planar graph has at most $3n - 6$ edges. We now wish to study the case when this is tight.

5.1 Maximal Planar Graphs

Definition 5.1.1 (Maximal Planar Graph)
A simple planar graph to which we cannot add more edges while staying planar.

Definition 5.1.2 (Triangulated Graph)
A simple planar graph with $n \geq 3$ that has a planar embedding for which every face is a triangle.

Lemma 5.1.1
The following are equivalent for a simple planar graph with $n \geq 3$.

1. G is triangulated
2. G has $3n - 6$ edges
3. G is maximal planar.

Proof
(1) $\implies$ (2) The proof that the number of edges is bounded by $3n - 6$ can be modified if the faces have exactly 3 incident edges.

(2) $\implies$ (3) Trivial.

$\neg(3) \implies \neg(1)$ Assume G is not triangulated. so some face F has at least 4 vertices incident to it. There must be vertices v, w in F which are not incident, lest we add a vertex to F ajacent with all of its vertices and get K_5.

Add the edge vw preserves planarity and thus G was not maximal planar.

Lemma 5.1.2
Any triangulated graph with $n \geq 3$ has $2n - 4$ faces.

Proof
Euler's Formula.

Lemma 5.1.3
Any simple triangulated graph is 3-connected.

Proof
If it is not, we could add an edge and remain simple, planar. so it could not be maximal planar.

So every triangulated graph has a unique planar embedding by Whitney's Theorem.

Lemma 5.1.4
Any simple planar graph can be made triangulated by adding edges.

Proof
If it is not triangulated, it is not maximal planar. Thus we can add edges until it is.

5.2 Related Graph Classes

We use the term triangulated graph only for simple planar graphs.

Definition 5.2.1 (Inner Triangulated Graph)
The underlying graph constructed by a set of points and straight-line edges such that all interior faces are triangles.

Definition 5.2.2 (Triangulated Disk)
Planar graph with a planar drawing that is inner triangulated with the outer face a simple cycle.

Lemma 5.2.1
Any triangulated disk G is 2-connected. Furthermore, any cutting pair $\{v, w\}$ has both v, w on the outer face connected by a chord of the outer face.

Proof
Let G^+ be obtained from G by adding a vertex u inside the outer face and making it adjacent to all vertices on F. Clearly G^+ is planar and triangulated. It is also simple. So G^+ is 3-connected and $G = G^+ - u$ is 2-connected.

For any cutting pair $\{v, w\}$, there are two faces F, F' that contains them both. At least one of them, say F, is not the outer face and so exists in G^+. If vw does not exist then we could add it inside F, contradicting maximality of G^+. So vw exists.

This means there are actually three faces F, F', F'' which contain both v, w. If neither is the outer face, then all three also exist in G^+, making $\{v, w\}$ a cutting pair in G^+, a contradiction.

So it must be that vw is a chord of the outer face as desired.

5.3 Canonical Ordering

This is a useful tool for graph drawing, planar encoding, and proving graph properties. The order exists for any 3-connected planar graph but we will focus on triangulated ones for simplicity sake.

In the following, whenever we speak of a triangulated graph, we assume that the unique rotational sytem is fixed, and an outer-face has been chosen.

Definition 5.3.1 (Canonical Ordering)
A vertex order

$$v_1, \ldots, v_n$$

of a triangulated planar graph such that v_1, v_2, v_n is the outer-face and any vertex v_k for $3 \leq k \leq n-1$ has at least two predecessors and at least one successor.

5.3.1 Properties

Given a canonical order, we write $G_k := G[v_1, \ldots, v_k]$ and assume its planar embedding is the one induced by G.

Lemma 5.3.1
Let G be a triangulated graph with canonical order

$$v_1, \ldots, v_n$$

Then for any $3 \leq k \leq n-1$

1. The outer face of G_k contains the edge $v_1 v_2$
2. v_k is on the outer-face of G_k
3. Every $v_j, j > k$ is in the outer-face of G_k
4. Every internal face of G_k is a face of G In particular, G_k is internally triangulated.
5. The predecessors of v_{k+1} form a a consecutive set of vertices (interval) on the outer-face of G_k
6. The outer-face of G_k is a simple cycle
7. G_k is a triangulated disk and in particular is 2-connected

Proof

1. G_k is an induced subgraph
2. reverse induction on k; we obtain G_k by deleting all $v_\ell, \ell > k$ which in particular includes the successor of G_k, bringing v_k to the outer face
3. v_j is on the outer face of G_j for all $j > k$.
4. all vertices in $G - G_k$ belong to the outer-face of G_k and hence all internal faces of G_k are also faces in G
5. since v_{k+1} is in the outer-face of G_k all its predecessors must be on the outer-face of G_k by planarity. When we add edges from v_k to predecessors, we create internal faces and these are necessarily triangles. Hence when we traverse the precessors of v_{k+1} in clockwise order, any predecessors that are consecutive must be adjacent as desired
6. follows by induction on k using the observation that $\text{indeg}(v_k) \geq 2$ means we v_k and at least 2 edges every time
7. follows from the previous property

5.3.2 Existence of the Canonical Order

Lemma 5.3.2
Let G be a triangulated disk. Let ab be an edge on the outer-face.
Then G has a vertex order

$$v_1 = a, v_2 = b, v_3, \ldots, v_n$$

such that every v_i for $i \geq 3$ has at least two precedecessors.
Moreover, any v_i for $i \leq n$ that is not on the outer-face has at least one successor.

Proof
We argue by induction on n.

for $n = 3$, G is necessarily a triangle, and the only possible vertex order satisfies all conditions.

Presume $n \geq 4$. The aim is to choose a vertex $v_n \neq a, b$ so that $G - v_n$ is still a triangulated disk. We do not have to worry about internal faces of $G - v_n$ so the only way $G - v_n$ is not a triangulated disk is that the outer face is NOT a simple cycle. In other words, we do not want $G - v_n$ to have a cut vertex v on the outer face. Observe that this happens if and only if $\{v, v_n\}$ is a cutting pair.

By our previous work lemma, vv_n is a chord. So we are actuallly looking for v_n on the outer face which is not incident to a chord.

If there are no chord, this is trivial. Suppose there are. Let

$$a = c_1, c_2, \ldots, c_p = b$$

be the outer face of G.

Pick $c_i c_j$ to be the chord which minimizes $j - i$. By construction, there are no OTHER chords in $\{c_i, c_{i+1}, \ldots, c_j\}$. Since the graph is planar, there is no chord between c_{i+1} and any vertex in $\{c_1, \ldots, c_{i-1}\}$ or $\{c_{j+1}, \ldots, c_p\}$.

Furthermore, we know

$$1 \leq i < i + 1 < j \leq p$$

so $c_{i+1} \neq a, b$ and we may choose v_n.

Then $G - v_n$ has the desired ordering by induction. Moreover, v_n has at least two predecessors (neighbours on the simple cycle which was the outer face of G). Finally, any vertex that is not on the outer-face of G has at least successor. This is due to the fact that it was either not on the outer-face of $G - v_n$, or it was adjacent to v_n and hence v_n is a successor.

Theorem 5.3.3
Let G be a triangulated planar graph with $\{a, b, t\}$ be the outer-face of G. There is a canonical order of G such that

$$v_1 = a, v_2 = b, v_n = t$$

Proof
By the lemma.

Lemma 5.3.4
A canonical order of a triangulated planar graph can be found in linear time.

5.3.3 Splitting into Trees

Theorem 5.3.5
The edge set of any triangulated simple G can be partitioned into 3 edge-disjoint trees.

Proof
Assume that we have the canonical ordering $v_1, \ldots, v_n$ of G. Label the edge v_1v_2 1.

Fix some $k \geq 2$ and enumerate the outer face of G_k

$$v_1 = c_1, c_2, \ldots, c_p = v_2$$

Let ℓ, r be the minimal and maximal indices where c_ℓ, c_r are predecessors of v_{k+1}. Notice that $\ell < r$ since $\text{indeg}(v_{k+1}) \geq 2$.

Assign 1 to edge $c_\ell v_{k+1}$ and 2 to the edge $c_r v_{k+1}$. Finally, assign 3 to all other edges from v_{k+1} to a predecessor. There must be a 1 and 2 edge but not necessarily a 3 edge.

For each $i \in [3]$, let T_i be the graph formed by the edges labeled i. We claim T_i is a tree.

Direct each edge from the lower-indexed to the higher-indexed endpoint.

Each vertex has at most one incoming edge labelled 1. This implies that the edges labeled 1 are a forest. Since every vertex except v_1 has an incoming 1 edge, T_1 has $n-1$ edges and is in fact a spanning tree of G.

Similarly, the edges labeled 2 are a forest. Observe that all vertices except v_1, v_2 have exactly one incoming 2 edge and v_1 is not incident to any edge labeled 2. It follows that T has $n-2$ edges an reaches $n-1$ vertices. Thus it is a spanning tree of $V - v_1$.

Finally, a vertex may have many incoming edges labelled 3 but it has at most one outgoing edge labeled 3. a vertex v_i has an outgoing 3 edge only if it disappears from the outer face in $G[v_{i+1}]$. Vertices v_1, v_2, v_n neveer disappear from the outer face and have no outgoing edges labeled 3. Furthermore, v_1, v_2 have no incident edges labeled 3. Therefore T_3 forms a spanning tree of $V - v_1 - v_2$.

Definition 5.3.2 (Schnyder Wood)
The tree trees obtained from a canonical order.

Arboricity

Definition 5.3.3 (Arboricity)
A graph G is said to have arboricity

$$a(G)$$

if there is a partition of the edges into at most $a(G)$ forests.

Corollary 5.3.5.1
Every planar graph has arboricity at most 3.

Proof
Add edges until we have a triangulated graph G'. From Schnyder Wood, we know $a(G') \leq 3$.

Since edge deletion does not increase arboricity

$$a(G) \leq 3$$

as desired.

5.3.4 Visibility Representation

Definition 5.3.4 (Bar Visibility Representation)
Each vertex is a horizontal line segment. Each edge is a vertical line segment connected the bars of its endpoints and intersecting no other bars.

Definition 5.3.5 (Strong Visibility Representation)
A bar visibility representation where edges between two vertices exist if and only if there is some vertical line intersecting both horizontal vertex lines and nothing else.

Definition 5.3.6 (Weak Visibility Representation)
A bar vibility representation where the rule for strong model does not necessarily apply.

Theorem 5.3.6
Every triangulated graph has a strong visibility representation.

Proof
Let $v_1, \ldots, v_n$ be the canonical ordering of a triangulated graph G. We create for $k \geq 2$ a visibility representation of G_k with the following invariant: For each outer face vertex c in clockwise order from v_1 to v_2, there is an interval where the bar of c can "see" upward to infinity.

This is trivial for G_2. Assume G_k has been drawn with the invariant holding. To add v_{k+1}, we create a new segment above the current visibility representation.

Let c_i, c_j be the left most and rightmost predecessors of v_{k+1}. Place the bar for v_{k+1} such that it covers partially the intervals for c_i, c_j and all intervals in between. Then since all $v_{k=1}$ is adjacent to all vertices between c_i, c_j on the outer face of G_k, all visibility lines from v_{k+1} aorrespond to edges.

In addition, the invariant continues to hold, so all visibility lines from v_{k+1} correspond to edges and c_i, c_j continue being on the outer face.

The result follows by induction.

Corollary 5.3.6.1
Every planar graph has a weak visibility representation.

Proof
Triangulate the graph and apply the theorem.

Corollary 5.3.6.2
Any planar graph has a weak visibility representation for which endpoints are grid points in

$$[1, 3n-6] \times [1, n]$$

Proof
There are n vertices and thus we need only n different y-coordinates to place vertex intervals. There are at most $3n - 6$ edges and thus we need only $3n - 6$ different x-coordinates to place edge lines.

Theorem 5.3.7
Every planar graph is the minor of a $k \times k$-grid for $k \leq 3n$.

Proof
Use the weak visibility representation Γ to obtain a $k \times k$-grid. That is, overlay Γ with a $k \times k$-grid H. Delete from H all vertices and edges that do not belong to a vertex bar or edge segment.

Finally, contract all edges along vertex bars and all but one edge of H along each edge segment. This precisely gives G and thus shows that H is a minor of H.

5.3.5 Straight-Line Embeddings

Theorem 5.3.8
Every simple planar graph has a straight-line embedding.

Proof
It suffices to show this for triangulated graphs.

Let $v_1, \ldots, v_n$ be a canonical ordering of G. We claim for $2 \leq k \leq n$:

G_k has a straight-lint drawing such that

$$x(c_1) < x(c_2) < \cdots < x(c_p)$$

where the outer face of G_k consists of $v_1 = c_1, c_2, \ldots, c_p = v_2$ in clockwise order. Here $x(v)$ denotes the x-coordinate of vertex v.

This holds for $k = 2$ by drawing v_1v_2 as a horizontal line segment. Assume we have such a drawing for G_k. We show how to add v_{k+1} appropriately.

Assume v_{k+1} is adjacent to

$$c_\ell, \ldots, c_r$$

on the otuer face of G_k with $\ell < r$. Let $x^* \in \mathbb{R}$ be such that

$$x(c_\ell) < x^* < x(c_r)$$

If we place v_{k+1} which this x-coordinate, the invariant will be satisfied.

> Choosing the y coordinate requires care. Since every x^* differs from the x-coordinate of its predecessors, there is a line from each c_i to the horizontal line $x = x^*$. This is due to the fact that there are only finite points which we need to worry about and they all have differing x-coordinates by induction.

Notice that the y-coordinates can easily explode with our construction. If we are more careful with choosing x-coordinates, we can obtain smaller y-coordinates and a smaller graph drawing overall.

Chapter 6

Friends of Planar Graphs

6.1 Super Classes of Planar Graphs

6.1.1 Graphs in 3D

Theorem 6.1.1
Any graph can be drawn straight-line without crossing in 3D.

Proof
Draw vertices on the moment-curve

$$\{(x, x^2, x^3) : x \in \mathbb{R}^+\}$$

with edges as straight lines.

If any edges cross, the 4 vertices on the ends of the crossing edges lie on one plane, so the span of those 4 points is 2-dimensional.

But any 3 distinct points on the moment curve are linearly independent by Vandermonde determinants, thus this never happens.

Definition 6.1.1 (Knot-Less Graph)
Can be drawn in $3D$ such that any cycle is homeomorphic to the unit circle.

Definition 6.1.2 (Link-Less Graph)
Can be drawn in $3D$ such that no two disjoint cycles form curves that are linked.

6.1.2 Graphs of Bounded Genus

These are graphs embeddable on the surfaces of topological objects with constant genus (holes).

A specific embedding in a surface can be described using a rotation system. From this embedding we can hence construct a dual graph.

Moreoever, Euler's formula holds in the generalized form

$$n - m + f = \chi$$

where χ is the Euler characteristic which for orientable surfaces is

$$\chi = 2 - 2g$$

where g is the genus and

$$\chi = 1$$

for the projective plane.

So we can bound the number of edges, the minimum degree, and the number of predecessors in a minimum degree order.

Some results include that any graph of genus g can be colored with

$$\left\lfloor \frac{7 + \sqrt{1 + 48g}}{2} \right\rfloor$$

colors.

Thus clique-size in such a graph is bounded and brute force solves the clique problem in polynomial time. Maximum cut is polynomial for graphs of bounded genus, leveraging the dual graph again. Though the same ideas do not transfer for maximum flow, it can still be solved in near linear time for graphs of bounded genus.

Unfortunately testing whether a graph has genus g is NP-hard.

6.1.3 Near Planar Graphs

Definition 6.1.3 (Bounded Crossing Number)
A graph G has crossing number k if there is a drawing of G in the plane that has k crossings.

Definition 6.1.4 (Bounded Skew Number)
A graph has skew number k if there exists a drawing of G in the plane with at most k edges and k is minimal.

Definition 6.1.5 (k-Planar Graphs)
Can be drawn such that any edge crosses at most k other edges.

1-planar graphs arise naturally from taking the union of a plane graph, its dual, and adding all edges for face-vertex incidences.

6.2 Subclasses of Planar Graphs

6.2.1 Trees

Self-explanatory.

6.2.2 Outer-Planar Graphs

Definition 6.2.1 (Outer-Planar)
A graph with a planar embedding such that all vertices are on the outer face.

Lemma 6.2.1
A graph G is outer-planar if and only if we can add a universal vertex u to G and the resulting graph is planar.

Proof
$(\implies)$ Trivial.

$(\impliedby)$ Assume u is on the outer face and delete it.

Corollary 6.2.1.1
Outer-Planar testing can be done in $O(n)$ time.

Maximal Outer-Planar Graphs

We are interested in outer-planar graphs which do not remain outer-planar with the addition of any edge.

Theorem 6.2.2
Any maximal outer planar graph G with $n \geq 3$ vertices satisfy

1. Every interior face is a triangle
2. The outer face is simple
3. G is a triangulated disk
4. G has a Hamiltonian cycle
5. G has exactly $n-2$ interior faces
6. G has exactly $2n-3$ edges

Proof
The only non-trivial observation is that a straight-line drawing of G contains an n-gon, which can be split into $n-2$ interior faces.

Lemma 6.2.3
Let G be a maximal outer-planar graph.
$G^* = \{v^*\} \cup T$ where v^* corresponds to the outer face and T is a tree with maximum degree 3.

Proof
Since G is 2-connected, so must G^* (by ear-decomposition for example).

G has $2n-3$ edges and $n-1$ faces. So G^* has G^* has $n-1$ vertices and $2n-3$ edges.

There are n edges incident to v^* thus $G^* - v^*$ has $n-2$ vertices with $n-3$ edges. This must be a tree! Since all faces except the outer face are triangles, all vertices except v^* have degree 3.

Lemma 6.2.4
Every maximal outer-planar graph has a vertex ordering $v_1, \ldots, v_n$ such that v_1v_2 is an edge on the outer face and $v_i, i \geq 3$ has exactly two predecessors that are adjacent ot each other.

Proof
The canonical order satisfies every condition except we permit more than two predecessors.

This is resolved by noting that $\text{indeg}(v_2) = 1$ and $\text{indeg}(v_i) = 2$ gives $2(n-2)+1 = 2n-3$ edges and anymore would violate our theorem above.

Problems

It is obvious that we can always find a 3-coloring using a greedy algorithm on the vertex-order.

6.2.3 k-Outer-Planar Graphs

Definition 6.2.2 (k-Outer-Planar Graphs)
Can be embedded within the plane where all vertices are removed by removing the outer face k times.

Clearly, any planar graph has outer-planarity at most $\frac{n}{3}$. It is not known whether this is tight.

Theorem 6.2.5
Let G be a planar graph that has a planar grid-drawing in a $w \times h$-grid. Then the outer-planarity of G is at most

$$\min\left\{\left\lceil\frac{w}{2}\right\rceil, \left\lceil\frac{h}{2}\right\rceil\right\}$$

Proof
Assume $h \leq w$ so we must know G is $\lceil\frac{h}{2}\rceil$-outer-planar. We prove this by induction on h.

In the base case $h = 1, 2$ the claim clearly holds.

Assume $h \geq 2$ and we have a drawing of height h. Then the vertices placed in the top most and bottommost row have to be on the outer face. Thus the outermost onion peel contains all these vertices (maybe more).

Remove the onion peel. The remaining graph is drawn on a grid of heigh $h-2$ and thus what remains is

$$\left\lceil\frac{h-2}{2}\right\rceil$$

outer-planar.

k-outer-planarity testing can be done in $O(k^3n^2)$ or even $O(n^2)$ time!

6.2.4 Series-Parallel Graphs

6.2.5 2-Terminal Series-Parallel Graphs

Definition 6.2.3 (2-Terminal SP-Graph)
A graph with two terminals s, t obtained through:

base case A single edge st

series combination If G_1, G_2 are 2-terminal SP-graphs, then identifying t_1, s_2 gives a 2-terminal SP-graph

parallel conbination Identifying s_1, s_2 and t_1, t_2 is a 2-terminal SP-graph

We will see $K_{1,3}$ is not 2-terminal SP and thus not even trees are 2-terminal SP-graphs. Moreover, $K_{1,3}$ appears as an induced subgraph of a 2-terminal SP-graph so this class is NOT classed under taking minors.

The SP-Tree

We can capture the sequence of conbinations used to construct a 2-terminal SP-graph G with a binary tree T.

base case If G is an edge st, then T is a single node labelled st

combination If G is the combination of G_1, G_2, then T has a node labelled S or P. This node is labelled G and its children are the SP-trees for G_1, G_2.

Lemma 6.2.6
Let G be a 2-terminal SP-graph with vw and edge.
The following are also 2-terminal SP-graphs

1. The graph obtained by duplicating vw
2. The graph obtained by subdividing the edge vw with a new vertex x

Proof
Get the leaf of the SP-tree representing vw. Replace this with a P node with two children representing vw and its duplicate.

For the second claim, replace this node with an S node with two children that represent vx and xw.

Lemma 6.2.7
Let G be a 2-terminal SP-graph with $n \geq 3$ and source terminals s, t.

1. G has at most $2n - 3$ edges
2. G is either a single edge or contains a vertex of degree 2
3. G is planar and has a drawing with the terminals on the outer face
4. G has a 3-coloring where the terminals hae different colour
5. G an acyclic edge orientation such that s is the only source and t is the only target

Proof
Structural induction on the SP-tree.

SP-Graphs

Definition 6.2.4 (Series-Parallel Graph)
A graph is called a series-parallel graph if it is a spanning subgraph of a 2-terminal series-parallel graph.

6.2.6 Apollonion Networks

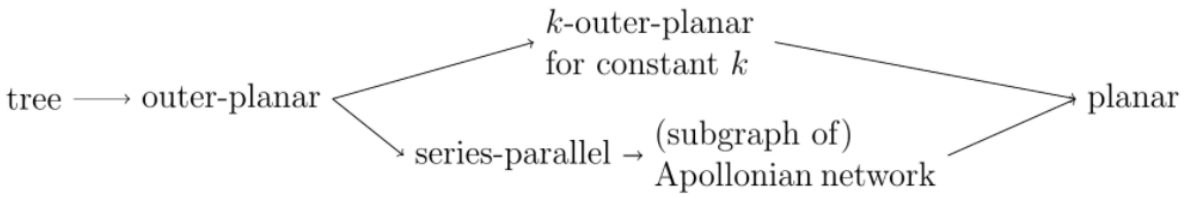
Definition 6.2.5 (Apollonion Network)
A triangulated plane graph obtained as follows

base case A triggle is an Apollonion network

Face Subdivision If G is an Apollonion network and T is a triangular inner face of G, the graph obtained from G by inserting a new vertex inside T and connecting it to T's vertices is also an Apollonion network.

Lemma 6.2.8
Any Apollonion network on $n \geq 4$ vertices has a vertex order $v_1, \ldots, v_n$ where v_1, v_2, v_3 form a triangle and for $i \geq 4$, the vertex v_i has exactly three predecessors that form a triangle.

6.2.7 Relationships between Subclasses of Planar Graphs



Part II

From Interval Graphs to Treewidth

Chapter 7

Interval Graphs & Friends

7.1 Interval Graphs

Definition 7.1.1 (Intersection Graph of Intervals)
Given a set of intervals, we define a graph with a vertex v for every interval I_v, and an edge vw if and only if the two corresponding intervals intersect.

Definition 7.1.2 (Interval Graph)
G is an interval graph if it is the intersection graph of some set of intervals.

Observe that we may assume without loss of generality that all endpoints are distinct in the range $[2n]$.

Every complete graph is an interval graph. Simply take the intervals to be copies of the same interval.

Proposition 7.1.1
A cycle of length $k \geq 4$ cannot be an interval graph.

Proof
Assume otherwise and let c_j be the vertices/intervals.

Pick c_i to be the one whose left endpoint ℓ_i is maximal. Then the interval of c_{i-1} start to the left of c_i. But $c_{i-1}c_i$ is an edge so its right endpoint r_{i-1} is to the right of ℓ_i. The same holds for c_{i+1}. In particular, c_{i-1}, c_{i+1} both share the point ℓ_i and they must have an edge.

Definition 7.1.3 (Induced-Hereditary)
Closed under taking induced subgraphs.

Interval graphs are induced-hereditary.

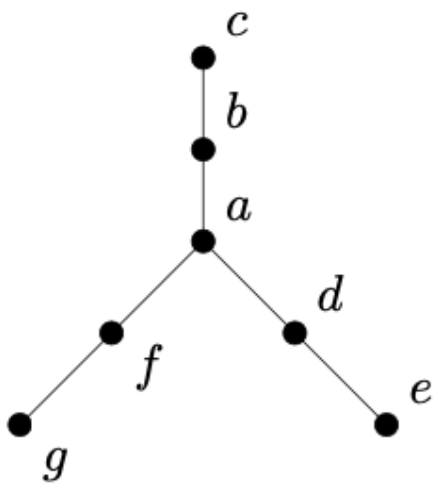
Definition 7.1.4 (Hereditary)
Closed under taking subgraphs.

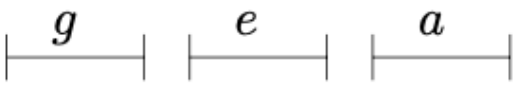
Interval graphs are NOT hereditary.

7.2 Chordal Graphs

Definition 7.2.1 (Chordal Graph)
A graph G is chordal if it does not contain an induced k-cycle for $k \geq 4$.

Observe that every interval graph is chordal. Moreoever, every tree is chordal. However, not all trees are interval graphs





7.3 Perfect Elimination Order

Observe that sorting the intervals of an interval graph by the left endpoint yields a vertex order. Consider some predecessor v_h of v_i in this order. Since v_h comes earlier, $\ell(v_h) < \ell(v_i)$.

Since it is a neighbour, $r(v_h) > \ell(v_i)$. So the set of predecessors and v_i,

$$\text{pred}(v_i) \cup \{v_i\}$$

form a clique!

Definition 7.3.1 (Perfect Elimination Order)
A vertex order $v_1, \ldots, v_n$ such that

$$\text{pred}(v_i)$$

is a clique for all $1 \leq i \leq n$.

Observe that the described vertex order of intervals graphs is a p.e.o. Futhermore, a pre-order traversal of a tree yields a p.e.o. Thus not every graph with a vertex elimination order is an interval graph.

Theorem 7.3.1
If G has a perfect elimination order, G is chordal.

Proof
Consider any k-cycle of C of G for some $k \geq 4$. Enumerate

$$C = c_0, \ldots, c_{k-1}$$

and pick the vertex in the cycle that appears last among the cycles in p.e.o.

c_i must have two predecessors and thus they must be adjacent. So C has a chord!

The converse actually holds as well but we defer the proof.

7.4 Problems in Chordal Graphs

7.4.1 Coloring

Theorem 7.4.1
The greedy algorithm yields the optimal number of colors on a p.e.o.

Proof
Assume the greedy algorithm yields a k-coloring. Any clique of size k requires k colors.

Any vertex requiring the use of the k-th color has $k-1$ predecessors. Thus

$$k \geq \chi(G) \geq \omega(g) \geq k$$

and we have equality throughout.

7.4.2 Clique

Corollary 7.4.1.1
let G be a graph with p.e.o. $v_1, \ldots, v_n$. Then

$$\omega(G) = \chi(G) = \max_i \{\text{indeg}(v_i) + 1\}$$

7.4.3 Independent Set

Definition 7.4.1 (Clique Cover)
A partition of $V(G)$ so that each class form a clique in G.

Theorem 7.4.2
The greedy algorithm on a p.e.o. $v_1, \ldots, v_n$ yields a maximum independent set.

Proof
View the greedy algorithm from n to 1. If v_i has not been marked yet, add v_i to the independent set and mark its predecessors.

Observe that marked sets form a clique cover. Suppose our independent set uses k vertices, one from each clique.

But obviously an independent set can have at most 1 vertex per clique. Thus k is optimal.

7.4.4 Dominating Set

Theorem 7.4.3
Dominating Set is NP-hard in chordal graphs.

Proof
We show a reduction from Vertex Cover in a general graph to Dominating Set in chordal

graphs.

Let $G = (V, E)$ and an integer k. Create a new graph G^+ as follows.

Subdivide every edge $e \in E$ with w_e being the identified vertex.

Add all edges between the original vertices of G.

We claim that G has a vertex cover C of size k if and only if G^+ has a dominating set S of size k.

Given a vertex cover of C, use the same vertex set $S = C$ in G^+. Since every $e \in E$ has an endpoint v in C, the subdivision vertex w_e in G^+ is dominated by v. Also, all vertices in $V \setminus S$ are dominated by any vertex in S since the vertices of V form a clique in G^+.

Now, fix some dominating set S in G^+. If S contains any subdivision vertices w_e for $e = uv$, we may replace w_e by v in S, as w_e dominates only u, v while v dominates at least u, w_e. So we may assume S only contains original vertex of V. With this, for any $w_e \in G^+$, there is a vertex $v \in s$ adjacent to w_e. This means that v is an endpoint of the edge e, and hence the same set S is also a vertex cover in G.

This shows the reduction. We need only show G^+ is chordal. We can easily get a p.e.o. of G^+ by enumerating the vertices in V (form a clique), and then the vertices w_e (both predecessors are adjacent) as required.

7.5 Friends of Interval Graphs

7.5.1 Intersection Graphs

Definition 7.5.1 (Intersection Graph)
A graph G is an intersection graph if there is a universe $\mathcal{X}$ such that every vertex v corresponds to a set $S_v \subseteq \mathcal{X}$ of objects of the universe.
Moreoever, the edge uv exists if and only if S_v and S_w intersect.

Theorem 7.5.1
Every graph is an intersection graph.

Proof
Let $\mathcal{X}$ be the set of edges and each vertex v, let

$$S_v := \{e = vw : e \in E\}$$

Given two vertex v, w, then S_v, S_w intersect if and only if there is an edge incident to both

v, w. This holds if and only if v, w are adjacent.

We can restrict the universe to make intersection graphs more interesting. Examples such as the intersection graph of d-dimensional boxes or unit discs.

Another example is the touching graphs of discs. This means the geometric objects are interior-disjoint and there exists and edge in the graph if and only if two corresponding objects share a boundary point. These graphs turn out precisely to be the planar graph.

7.5.2 $\mathcal{H}$-Free Graphs

Definition 7.5.2 (Induced $\mathcal{H}$-Free Graph)
Let $\mathcal{H}$ be an arbitrary collection of graphs. The graph clas of induced-$\mathcal{H}$-free graphs are the ones whose induced subgraph NEVER belong to $\mathcal{H}$

Some examples include the P_4-free graphs (path with 4 vertices), AT-free graphs, and Odd-hole (Odd-anti-hold) -free graphs.

A hole is a cycle $C_k, k \geq 4$ and an anti-hole is the complement of such a cycle.

7.5.3 Perfect Graphs

Definition 7.5.3 (Perfect Graph)
A graph G is perfect if

$$\omega(H) = \chi(H)$$

for every (not necessarily proper) induced subgraph H of G

Theorem 7.5.2 (Strong Perfect Graph Theorem)
A graph is perfect if and only if it does not have an odd hole or an odd anti-hole as induced subgraph.

Chapter 8

Recognizing Chordal Graphs & Interval Graphs

8.1 Finding a Perfect Elimination Order

8.1.1 Finding Simplicial Vertices

We need an alternative characterization of p.e.o.'s.

Definition 8.1.1 (Simplicial Vertex)
$v \in V(G)$ such that the neighbours of v form a clique in G.

Lemma 8.1.1
A graph G has a p.e.o. if and only if every induced subgraph of G has a simplicial vertex.

Proof
If G has a perfect elimination order and H is an induced subgraph, then the last vertex of H in the p.e.o. is simplicial in H, as all its neighbours in H are predecessors in the p.e.o. and so form a clique.

Vice versa, if all induced subgraphs of G contain a simplicial vertex, let v_n be a simplicial vertex of G. By induction on the number of vertices $G - v_n$ has a perfect elimination order, and appending v_n to it gives a p.e.o. of G.

This lemma implies a simple polynomial time algorithm to test whether a graph has a p.e.o. Search for a simplicial vertex v_n. If we find one, append it is to a recusively obtained order

of $G - v_n$.

8.1.2 Maximum Cardinality Search

This is a much faster algorithm which works by repeatedly choosing the vertex for which the maximum number of neighbours has already been chosen. Ties are broken arbitrarily.

```
def MaxCardSearch:
  for v in V:
    chosen[v] = False
    p[v] = 0

  for i=1, ..., n:
    _, v_i = max((p(v), v) for v in V)
    chosen[v_i] = True
    for w in adj(v_i):
      if not chosen[w]:
        p[v] += 1

        return v_1, ..., v_n
```

the run time is dominated by the time to find the vertex maximizing $p(v)$ among remainig vertices. This can be done easily with a priority queue. But since $0 \leq p(v) \leq n-1$ for all v and we only increment elements by at most 1, we can actually do this with a suitable bucket struct of linked-lists such that the time to find v_i or update $p(w)$ is constant.

So the run time can be reduced to

$$O(m+n)$$

Lemma 8.1.2
Let $C = c_0, \ldots, v_{k-1}$ be a cycle in a chordal graph. Then for any $0 \leq i < k$, either $c_{i-1}c_{i+1}$ is an edge, or c_i is incident to a chord of C.

Proof
We argue by induction on k. The case $k = 3$ is trivial.

Assume $k \geq 4$ and c_i is not incident to a chord of C. Since G is chordal, there must exist some chord $c_j c_\ell$ in C. This chord splits C into two parts

$$C', C''$$

One of these parts, say C' contains c_i. If C' had a chord incident to c_i, then this would be a chord of C which is a contradiction. So by induction on C', the edge $c_{i-1}c_{i+1}$ exists.

To see that MCS works, we show that it maintains the following: Any set that separates an undiscovered connected component from the discovered part must be a clique. We need the following notation: For any vertex-set S, let $N_i(S)$ be the set of neighbours of S in $\{v_1, \ldots, v_n\}$.

Lemma 8.1.3
Assume $v_1, \ldots, v_n$ was the output of MCS. Let C be a component of $G \setminus \{v_1, \ldots, v_i\}$, for some $1 \leq i < n$.
Then $N_i(C)$ is a clique.

Proof
We proceed using induction on i. The base case $i = 1$ is obvious as $N(C) \subseteq \{v_1\}$ which is a trivial clique.

Consider the time when we add vertex v as v_{i+1} to the order. Fix an arbitrary component C' of $G \setminus \{v_1, \ldots, v_{i+1}\}$. We know that $N_i(C')$ is a clique by induction, and want to show that $N_{i+1}(C')$ is also a clique.

This can be violated if and only if $v \in N_{i+1}(C')$, and some vertex in $N_i(C')$ is NOT adjacent to v.

Assume for a contradiction that v has a neighbour y in C', and there is x in C' (possibly $x = y$) with a neighbour $p_x \in N_i(C')$ such that $p_x v \notin E$.

Let

$$x = x_1, x_2, \ldots, x_d = y$$

be a path from x to y within C'. By re-defining x if necessary, we may assume p_x is NOT adjacent to $x_i, i \geq 2$.

We claim that v has a neighbour p_v in $\{v_1, \ldots, v_i\}$ that is not adjacent to x. this holds because we picked v rather than x in MCS, so

$$|N_i(v)| \geq |N_i(x)|$$

We have $p_x \in N_i(x) \setminus N_i(v)$ so there must be at least one vertex p_v in $N_i(v) \setminus N_i(x)$, or else we would have taken x.

Observe that both v, x belonged to one connected componenet C_v of $G \setminus \{v_1, \ldots, v_i\}$, due to the path $x_1, \ldots, x_d, v$. Therefore p_v, p_x both belong to $N_i(C_v)$, and since this is a clique by induction we know $p_v p_x$ is an edge.

Now apply our lemma to the cycle

$$v, p_v, p_x, x_1, \ldots, x_d, v$$

using vertex p_x.

This states that either we have the edge $p_v x$, or the edge $p_x v$, or some edge $p_x x_i$ for $i > 1$.

All of these cases have been ruled out above which is a contradiction. We conclude $N(C')$ is a clique as required.

Theorem 8.1.4
Let G be a chordal graph. Then the order $v_1, \ldots, v_n$ returned by MCS is a perfect elimination order, regardless of how ties are broken.

Proof
We must show that $\text{pred}(v_i)$ is a clique for $i = 1, \ldots, n$. To this end, consider the subgraph G_i induced by $v_1, \ldots, v_n$, and observe that it is chordal and $v_1, \ldots, v_i$ has been obtained by runnning MCS on G_i.

Vertex v_i is a component of $G_i \setminus \{v_1, \ldots, v_{i-1}\}$ and its neighbourhood

$$N_{i-1}(v_i) = \text{pred}(v_i)$$

is a clique by our lemma.

Observe that this finishes the characterization of chordal graphs as those which have a perfect elimination order.

8.1.3 Lexicographic BFS

MCS was improved later to lexicographic BFS. This has many properties which make it useful for other applications. For example, when a vertex can be the last vertex.

However, it is harder to see why lexicographic BFS runs in linear time. Since we do not need the algorithm further, we omit the details.

8.2 Testing a Putative Perfect Elmination Order

We know if G is chordal, then the order $v_1, \ldots, v_n$ computed by MCS is a p.e.o. Thus to test if G is chordal, we only need a way to test a vertex ordering if is a p.e.o.

8.2.1 An Idea

Assume we already know that

$$v_1, \ldots, v_{i-1}$$

is a p.e.o.

Let v_h be the last predecessor of v_i (the one maximizing h).

For each $u \in \text{pred}(v_i) - v_h$, test whether u is adjacent to v_h. If this fails at some u, $\text{pred}(v_i)$ cannot be a clique. If they are all successful then

$$\text{pred}(v_i) \subseteq \text{pred}(v_h) \cup \{v_h\}$$

is a clique by induction.

Notice that this particular idea takes $O(m)$ adjacency queries. This could take more than linear time since adjacency-queries may take more than constant time.

Rather than testing immediately, we can gather all such pairs inot one multi-set E of pairs of vertices and test them all at once in the end. This shifts the burden of work to the last step.

We can bucket sort E and the edge-list L of G, and check if $E \subseteq L$. Each vertex adds $\text{indeg}(v) - 1$ pairs to E so the total time taken is

$$O(m + n + |E|) = O(m + n)$$

Theorem 8.2.1
Chordal graphs can be recognized in linear time.

8.3 Recognizing Interval Graphs

Lemma 8.3.1
Let G be a graph with p.e.o.

$$v_1, \ldots, v_n$$

Theb any clique C is a subset of

$$\{v_i\} \cup \text{pred}(v_i)$$

for some i.
If C is a maximal clique, then

$$C = \{v_i\} \cup \text{pred}(v_i)$$

Proof
Let i be the maximum index of a vertex in C. Then all other vertices in C are predecessors

of v_i, and

$$C \subseteq \{v_i\} \cup \text{pred}(v_i)$$

If C is maximal, v_i cannot have more predecessors, or else we can easily increase the size of the clique.

Theorem 8.3.2
G is an interval graph if and only if the maximal cliques of G can be ordered consecutively.
In other words we can listed them as $C_1, \ldots, C_k$ such that if $v \in C_i, C_j$

$$\forall i < h < j, v \in C_h$$

Proof
$(\Longleftarrow)$ If $C_1, \ldots, C_k$ is a consecutive ordering of the maximal cliques, set

$$I_v := [\min\{i, v \in C_i\}, \max\{j : v \in C_j\}]$$

To see that this is indeed an interval representation, assume $vw \in E$. There is some maximal clique C_h which contains this edge. Thus $v, w \in C_h$. By the definition of I_v, I_w, both intervals contain h and hence they intersect.

On the other hand, Suppose I_v, I_w intersect, say at h. Since the clique ordering is consecutive, this implies that both v, w belong to C_h and $vw \in E$ as desired.

$(\Longrightarrow)$ Assume that G is an interval graph with an interval representation with endpoints in $[2n]$. Now for $i = 1, \ldots, 2n-1$, let C_i be the clique formed by the intervals that intersect the open range

$$(i, i+1)$$

We claim $C_1, \ldots, C_{2n-1}$ includes all maximal cliques. To see this let $v_1, \ldots, v_n$ be the perfect elimination order obtained by sorting vertices by the left endpoint. By the lemma, every maximal clique C has the form

$$\text{pred}(v_i) \cup \{v_i\}$$

where v_i is the vertex with maximal index.

Thus all vertices in $\text{pred}(v_i)$ intersect the opern interval

$$(\ell(v_i), \ell(v_i) + 1)$$

where $\ell(v_i)$ denotes the left endpoint of v_i.

In the proposed clique order, every vertex appears in a consecutive set of cliques, since it belongs to an interval. Removing from this order all cliques that were not maximal hence gives the desired order.

So to find all maximal cliques, we start with a candidate set

$$\mathcal{C} = \{\{v_i\} \cup \text{pred}(v_i)\}$$

Proposition 8.3.3
Fix a p.e.o.

$$v_1, \ldots, v_n$$

and set $C_i := \{v_i\} \cup \text{pred}(v_i)$.
Then C_i is not a maximal clique if and only if there is an index $j > i$ such that v_i is the last predecessor of v_j and

$$\text{indeg}(v_j) = \text{indeg}(v_i) + 1$$

Proof
$(\implies)$ If C_i is not a maximal clique, there is a vertex $v_j \notin C_i$ adjacent to all of C_i. We cannot have $j < i$, thus choosing the smallest such j, we get that v_i is the last predecessor of v_j.

In this case

$$\text{pred}(v_j) \subseteq \text{pred}(v_i) \cup \{v_i\} \wedge C_i \subseteq \text{pred}(v_j)$$

so in fact $\text{pred}(v_j) = C_i$ and

$$\text{indeg}(v_j) = |C_i| = \text{indeg}(v_i) + 1$$

$(\impliedby)$ Since v_i is the last predecessor of v_j

$$\text{pred}(v_j) \subseteq \text{pred}(v_i) \cup \{v_i\} = C_i$$

But then by cardinality

$$\text{pred}(v_j) = C_i$$

so $C_i \subset C_j$ and C_i is not maximal.

Observe that finding maximal cliques in a chordal graph now amounts to scanning all vertices and comparing in-degrees. This takes $O(\deg(v))$ time per vertex, hence the time to find the maximal cliques is linear.

```
def max_cliques_in_chordal(G);
  v_1, ..., v_n = peo(G)
  to_delete = [False for v_i in G]

  for j=1, ..., n:
    # O(deg(v)) per vertex
    cache[v_j] = (indeg(v_j), last_pred(v_j))
```

```
for i=1, ..., n:
  for v_j in succ(v_i):
    indeg_v_j, last_pred_v_j = cache[v_j]
    indeg_v_i, _ = cache[v_i]

    if v_i == last_pred_v_j and indeg_v_j == indeg_v_i + 1:
      to_delete[v_i] = true
      break
```

8.3.1 PQ-Trees

No we want to bring maximal cliques into consecutive order. To do so we once again leverage PQ-trees.

1) Test if G is chordal. If not, G cannot be an interval graph. If so, compute a p.e.o. of G.
2) Compute all maximal cliques $\mathcal{C}$ of G.
3) Define the X (universe of PQ-tree) to be $\mathcal{C}$
4) For each $v \in V$, add a constraint I_v that all cliques containing v must be consecutive
5) Test if there is a permutation π of X that all constraints are consecutive
6) If no such permutation exists, G is not an interval graph. Otherwise, we can extract an interval representation

Theorem 8.3.4
Testing whether a graph is an interval graph can be done in linear time.

Proof
We argue that neither $X, \mathcal{I}$ (set of constraints) is too big.

Each clique in $\mathcal{C}$ has the form $\text{pred}(v_i)$ for some vertex i. So

$$|X| \leq |\mathcal{C}| = n$$

For any $v \in V$, if $v \in C_i$, then either $v = v_i$ or v is a predecessor of v_i.

Hence v belongs to at most $\text{outdeg}(v) + 1$ maximal cliques, and

$$\sum_{I_v \in \mathcal{I}} |I_v| \in O(m + n)$$

The result following from the running time of PQ-trees.

8.3.2 Other Algorithms

The algorithm presented was the first linear time algorithm for recognizing interval graphs. Subsequent algorithms have been developped avoiding the use of PQ-trees. A simple one runs lexBFS 5 times. The algorithm is simple but the proof is quite involved.

Chapter 9

Tree Decompositions

9.1 Strong Path Decomposition

We showed that a graph is an interval graph if and only if the maximal cliques can be listed in a "path".

Definition 9.1.1 (Strong Path Decomposition)
A strong path decomposition $\mathcal{P}$ of a graph $G = (V, E)$ consists of a path P with nodes I and an assignment $X : I \to 2^V$ to each node of P such that

(i) Every vertex of G belongs to at least one bag

(ii) For every $v \in V$, the bags containing V are consecutive along P

(iii) For any pair of vertices v, w, $vw \in E$ if and only if there is some bag X_i containing v, w.

Observe that every bag X_i in such a decomposition forms a clique. This is due to (iii).

While bags are cliques, they do not have to be maximal. Moreoever, duplicate bags are perfect acceptable. In particular, strong path decompositions are NOT unique!

Corollary 9.1.0.1
A graph G is an interval graph if and only if it has a strong path decomposition.

9.2 Strong Tree Decomposition

We can generalize this idea to chordal graphs.

Definition 9.2.1 (Strong Tree Decomposition)
A strong tree decomposition $\mathcal{T}$ of a graph $G = (V, E)$ consists of a tree T with nodes I and an assignment $X : I \to 2^V$ to each node of T such that

(i) Every vertex of G belongs to at least one bag

(ii) For every $v \in V$, the bags containing V are connnected ie are connected (connectivity condition)

(iii) For any pair of vertices v, w, $vw \in E$ if and only if there is some bag X_i containing v, w.

Notice that we use the term strong tree decomposition to emphasize that an edge exists in the graph if and onloy if the two endpoints are in a bag. We can weaker this to an only if condition and get a weak tree decomposition, where not all pairs in a bag have an edge.

To avoid confusion, we will talk about the nodes or arcs (links) of the "host tree" T and vertices or edges of the original graph G.

There is a different way to view a strong tree decomposition.

Definition 9.2.2 (Intersection Graph of Subtrees)
Consider a tree T. Let $T_1, \ldots, T_n$ be some subtrees of T.
We can then define a graph with vertices

$$v_1, \ldots, v_n$$

that has an edge $v_i v_j$ if and only if the subtrees T_i, T_j have a node of T in common.

Proposition 9.2.1
A graph is an intersection graph of subtrees of a tree if and only if it has a strong tree decomposition.

Proof
Assign a vertex to a bag if and only if its subtree contains that node, and vice versa.

9.2.1 Perfect Elimination Orders & Tree Decompositions

Lemma 9.2.2
Assume G has a strong tree decomposition. Then G has a perfect elimination order

$$v_1, \ldots, v_n$$

Furthermore, for $j = 1, \ldots, n$, there exists a bag that contains

$$\{v_j\} \cup \text{pred}(v_j)$$

Proof
Traverse the tree in pre-order to enumerate bags

$$X_1, \ldots, X_k$$

For $i = 1, \ldots, k$, let the graph G_i be the graph for which $X_1 \cup \cdots \cup X_i$ is a strong tree decomposition.

Now we build by induction on i a p.e.o. of G_i.

This is easy for $i = 1$ as the vertices form a clique. Any ordering $v_1, v_2, \ldots$ will do.

For $i > 1$, assume we have a p.e.o. of G_{i-1}. Let X_h be the parent of X_i in the traversal, and $y_1, \ldots, y_\ell$ be the vertices that are in $X_i \setminus X_h$. By the connectivity condition and since X_h is the parent of X_i. These vertices do not appear in $X_1, \ldots, X_{i-1}$.

If $\ell = 0$, then $G_i = G_{i-1}$ and we are done. Assume otherwise that $\ell > 0$. Any neighbour of $y_1, \ldots, y_\ell$ in G_i must be in X_i, as none of $X_1, \ldots, X_{i-1}$ contains $y_1, \ldots, y_\ell$.

Since we have a strong tree decomposition, X_i induces a clique. So $y_1, \ldots, y_\ell$ are all simplicial in G_i, and appending them in arbitrary order to the p.e.o. of G_{i-1} gives the result.

Corollary 9.2.2.1
Let $\mathcal{T}$ be a strong tree decomposition of a graph G. Then for any clique C, some bag of $\mathcal{T}$ contains all vertices of C.

Proof
Obtain the p.e.o. $v_1, \ldots, v_n$ from $\mathcal{T}$ as above. By our previous work

$$C \subseteq \{v_j\} \cup \text{pred}(v_j)$$

for some v_j, and we know that this set appears within one bag.

Lemma 9.2.3
If G has a p.e.o. $v_1, \ldots, v_n$, then it has a strong tree decomposition. Moreover, every bag has the form

$$\{v_i\} \cup \text{pred}(v_i)$$

for some i, and for every i there exists such a bag.

Proof
We prove the claim for

$$G_i := G[v_1, \ldots, v_i]$$

by induction on i.

We start by putting v_1 into one bag X_1 of the tree decomposition. Clearly this satisfies all conditions.

Now consider vertex $v_i, i > 1$. Let v_h be its last predecessor, and X_h be the bag

$$X_h = \{v_h\} \cup \text{pred}(v_h)$$

. Add a new bag

$$X_i := \{v_i\} \cup \text{pred}(v_i)$$

to the decomposition, and make it adjacent to X_h.

Clearly every vertex appears in one bag and every edge is covered by the bag of its endpoint with bigger index.

The connectivity condition held within $X_1, \ldots, X_{i-1}$ by induction, so we need only check vertices in X_i. Indeed, v_i only appears in X_i so the condition holds. for any $w \neq v_i \in X_i$, we know

$$w \in \text{pred}(v_i) \subseteq \{v_h\} \cup \text{pred}(v_h) = X_h$$

so the new bag containing w is adjacent to a previous bag containing w and the connectivity condition holds for w.

Theorem 9.2.4
A graph G is chordal if and only if it has a strong tree decomposition $\mathcal{T}$. Furthermore, the clique size $\omega(G)$ satisfies

$$\omega(G) \leq k + 1$$

if and only if all bags of $\mathcal{T}$ have size at most $k + 1$.

Proof
Given G, find a perfect elimination order $v_1, \ldots, v_n$ and build $\mathcal{T}$ using our lemma. We

know $\omega(G) = \max_i\{\text{indeg}(v_i)+1\}$, hence the bag-size is upper-bounded by the clique size.

Vice versa, given $\mathcal{T}$, extract a p.e.o. $v_1, \ldots, v_n$ using another lemma. Since the maximum clique occurs in some bag, $\omega(G)$ is upper-bounded by the maximum clique size.

Chapter 10

Treewidth

10.1 k-Trees

Lemma 10.1.1
The following are equivalent for graph with $n \geq 2$.

1. G is a tree
2. G has a p.e.o $v_1, \ldots, v_n$ such that $\text{indeg}(v_i) = 1$ for $i > 1$.
3. G is a chordal graph with maximum clique-size $\omega(G) = 2$ and $n - 1$ edges.

Proof
(3) $\implies$ (1) Trivial.

(2) $\implies$ (3) Trivial.

$\neg$(1) $\implies$ $\neg$(3) Suppose G is not a tree but it is chordal and has $n - 1$ edges.

This implies G has a cycle and since it is chordal, a cycle of size 3. So $\omega(G) \geq 3$.

Definition 10.1.1 (k-Tree)
A graph G with $m \geq k + 1$ vertices is a k-tree if G has a p.e.o. such that

$$\text{indeg}(v_i) = k$$

for all $k + 1 \leq i \leq n$.

10.1.1 Properties of k-Trees

Observe that the first $k+1$ vertices of $v_1, \ldots, v_{k+1}$ must form a clique since $\text{indeg}(v_{k+1}) = k$.

The maximum clique is has size

$$\omega(G) = \max_i \{\text{indeg}(v_i)\} + 1 = k + 1$$

Furthermore, the number of edges is

$$\sum_i \text{indeg}(v_i) = 1 + 2 + \ldots, (k-1) + \sum_{i>k} k = kn - \frac{k^2}{2} - \frac{k}{2}$$

and in particular $m \in O(kn)$ which is linear for k constant.

Theorem 10.1.2
Let G be a graph with $n \geq k+1$. Then G is a k-tree if and only if

(i) G is chordal

(ii) $\omega(G) = k+1$

(iii) $m = kn - \frac{k^2}{2} - \frac{k}{2}$

Proof
Necessity is proven.

Suppose now that the three conditions hold.

We have $\text{indeg}(v_i) \leq k$ for all i, but the first $i \leq k$ vertices has $\text{indeg}(v_i) \leq i - 1$. Thus the number of edges is AT MOST

$$kn - \frac{k^2}{2} - \frac{k}{2}$$

but we have equality so the indegrees are actually exactly as described.

So G is a k-tree.

10.1.2 Planar k-Trees

Every tree is a 1-tree. In fact this is an if and only if relation.

Every maximal outer-planar graph is a 2-tree, since the canonical ordering satisfies the properties of the theorem. However, not every 2-tree is maximal outer-planar. Indeed $K_{2,3}$ is a subgraph of a 2-tree which cannot be a 2-tree.

Every Apollonion network is a 3-tree, as its constructive vertex ordering satisfies the required conditions. On the other hand, not every 3-tree is an Apollonion network. $K_{3,3}$ is a subgraph of an Apollonion network but it is not even planar.

Lemma 10.1.3
Every 2-tree is a 2-terminal SP-graph.

Proof
This holds for $n = 2$ since the graph is just an edge. For $n \geq 3$, let $v_1, \ldots, v_n$ be a p.e.o. of G such that $\text{indeg}(v_i) = 2$ for all $i \geq 3$.

The graph induced by $v_1, \ldots, v_{n-1}$ is a 2-tree, and thus by induction is a 2-terminal SP-graph.

Let v_h, v_j be the predecessors of v_n. Replace the leaf node of the SP-tree holding $v_h v_j$ with the graph obtained by a series connection

$$v_h v_n, v_n v_j$$

and a parallel connection

$$v_h v_j, v_h v_n v_j$$

While the converse does not hold, every 2-terminal SP-graph is a subgraph of a 2-tree.

10.2 Partial k-Trees

Definition 10.2.1 (Partial k-Tree)
A spanning subgraph of a k-tree.

Partial 1-trees are spanning subgraphs of trees, i.e. forests.

Every outer-planar graph is a partial 2-tree, as it is a spanning subgraph of a maximal outer-planar graph, which is a 2-tree.

A partial 2-tree is an SP-graph, as it is a spanning subgraph of a 2-tree, which we have shown to be a 2-terminal SP-graph.

$$\text{outer-planar graphs} \subseteq \text{partial 2-trees} \subseteq \text{SP-graphs}$$

It can be shown that every SP-graph is a partial 2-tree.

All partial Apollonion networks are partial 3-trees, since every Apollonion network is a partial 3-tree.

Lemma 10.2.1
Any chordal graph G with $\omega(G) \leq k+1$ is a partial k-tree.

Proof
If $n \leq k+1$, simply add edges to make it into a clique. So assume $n > k+1$.

Fix a p.e.o. of G. We know

$$\text{indeg}(v_i) \leq \omega(G) - 1 \leq k$$

for all i. We show how to add edges so that $\text{indeg}(v_i) = k$ for $i > k$.

Initially add edges so the first k vertices form a clique.

For $k+1 \leq i \leq n$ let v_h be the last predecessor of v_i. If $h \leq k$, then add all edges from v_i to

$$\{v_1, \ldots, v_k\} = C_k$$

We then have $\text{pred}(v_i) = C_k$ and they form a clique.

If $k < h < i$, then by induction $\text{indeg}(v_h) = k$ and

$$C_h := \{v_h\} \cup \text{pred}(v_h)$$

is a clique of size $k = 1$. We also know the predecessors of v_i are a subset of C_h. If v_i currently has fewer than k predecessors, add edges to the vertices of C_h until it has exactly k. This does not affect the validity of the p.e.o.

Notice that this shows us every partial k-tree is a partial ℓ-tree for $\ell > k$.

Corollary 10.2.1.1
If G is a subgraph of a k-tree H, G is a partial k-tree.

Proof
We know that H is chordal with $\omega(G) \leq k+1$. Let H^- be the induced subgraph of vertices in G. Then H^- is still chordal and the upper bound on clique sizes still holds.

Thus H^- is again a partial k-tree, and so is its spanning subgraph G.

10.3 Treewidth

Definition 10.3.1 (Tree Decomposition)
A (weak) tree decomposition of a graph $G = (V, E)$ is a strong tree decomposition of some super-graph of G.
Thus it is a tree T with nodes I and an assignment $X : I \to 2^V$ of a bag X_i to each node $i \in I$ such that

(i) every vertex appears in at least one bag

(ii) for every $v \in V$, the bags containing v for a subtree of T

(iii) if $vw \in E$ is an edge of G, there is a bag that contains both v, w

The tree decomposition does not tell us the edges of the represented graph. It does however rule out some vertex-pairs that cannot be edges.

Every graph has a tree decomposition–simply put all vertices into one bag.

The tree decomposition is NOT unique.

Definition 10.3.2 (Width)
The width of a tree decomposition is

$$\max_i |X_i| - 1$$

Theorem 10.3.1
The following are equivalent for a graph G with $n \geq k + 1$.

1. G is a partial k-tree
2. G is a subgraph of a chordal graph H with $\omega(G) = k + 1$
3. G has treewidth at most k

Proof
(1) $\implies$ (2) By definition.

(2) $\implies$ (3) Get a strong tree decomposition of H where all bags have size at most $k+1$

(3) $\implies$ (1) Pick a tree decomposition $\mathcal{T}$ of G with width k. Let H be the super-graph for which $\mathcal{T}$ is a strong tree decomposition. $\omega(H) \leq k + 1$ thus H is the subgraph of a k-tree by a previous lemma.

Hence G is a partial k-tree by a previous corollary.

Graphs of bounded treewidth are of interest as they are a generalization of trees, and therefore

many of the dynamic programming techniques on trees can easily be extended to them.

10.3.1 Properties of the Treewidth

Proposition 10.3.2
For any tree decomposition $\mathcal{T}$ of G, and any clique C of G, there is a bag of $\mathcal{T}$ containing C.
In particular

$$\omega(G) \leq \text{tw}(G) + 1$$

Proof
The chordal super graph H for which $\mathcal{T}$ is a strong tree decomposition also contains C as a clique in H.

Obtain a p.e.o. from the $\mathcal{T}$ of H and use the fact that

$$C \subseteq \{v_i\} \cup \text{pred}(v_i)$$

for some $i \in [n]$.

Note that this bound is NOT tight.

Proposition 10.3.3
If $\text{tw}(G) \leq k$ then for any subgraph G' of G we also have

$$\text{tw}(G') \leq k$$

Proof
G' is still a partial k-tree.

Lemma 10.3.4
If G has treewidth at most k, and H is obtained by contracting an edge vw in G, then H has treewidth at most k.

Proof
Fox a tree decomposition of width k for G. Let x be the vertex that replaced v and w.

Replace every occurrence of either v or w by x. It is easy to check that this tree decomposition covers all edges.

To see why the subgraph of bags containing x is connected, notice that the subgraph of bags containing v, w each were connected, and they had a node in common (the bag

containing the edge vw), so the union of these two graphs is also connected.

Thus this yields a tree decomposition for H of width at most k.

Corollary 10.3.4.1
Graphs of treewidth at most k are closed under taking minors.

Corollary 10.3.4.2
Let H be a connected subgraph of G. Then in any tree decomposition $\mathcal{T}$ of G, the set of bags containing a vertex from H is connected.

Proof
Let G/H be the graph obtained from G by contracting all vertices in H into one vertex h. This is a minor of G.

If we modify the tree decomposition $\mathcal{T}$ as in the above proof, then we retain the same bags and vertex h appears in all bags that had a vertex of H in it.

But such bags are connected and we are done.

10.3.2 Graphs with Big Treewidth

How to we show a lower bound on tree-width?

Proposition 10.3.5
Any graph which contains K_{k+1} as a minor has treewidth at least k.

It is easy to see that the $a \times b$-grid is a partial k-tree for $k = \min(a, b)$. On the other hand,

Lemma 10.3.6
The tree width of the $k \times k$ is exactly k for $k \geq 2$.

Corollary 10.3.6.1
If a graph G contains a $k \times k$-grid as a minor then

$$\mathrm{tw}(G) \geq k$$

10.3.3 Series-Parallel Graphs

Recall that 2-terminal SP graphs always have a vertex of degree 2 given $n \geq 3$. Moreoever, doubling or subdividing an edge maintains a 2-termianl SP graph as we can easily update

the SP-tree.

SP-Graphs & Treewidth

Lemma 10.3.7
Every simple 2-terminal SP-graph has treewidth at most 2.

Proof
A tree decomposition of width 2 can be obtained directly from the SP-tree.

Label every node with the terminals of the graph it represents. For an S-node, also add the vertex which resulted from identifying the terminals of the children.

Every vertex and edge is covered via the leaves. The connectivity condition is easily verified by the observation that a vertex appears in two subtrees of a node i only if it was a termianl in both graphs of the children of i.

Combining this with a previous lemma

$$\text{2-trees} \subseteq \text{simple 2-terminal SP-graphs} \subseteq \text{partial 2-trees}$$

Corollary 10.3.7.1
SP-graphs are the same as partial 2-trees.

Taking spanning subgraphs over all the classes, we see that SP-graphs are precisely partial 2-trees.

Corollary 10.3.7.2
SP-graphs are the same as partial 2-trees.

Recognizing SP-Graphs

Lemma 10.3.8
If G is a simple SP-graph, then any simple minor G' of G has a vertex of degree at most 2.

Proof
Since G is a partial 2-tree, so is its simple minor G'. Thus G' is an SP-graph and a (spanning) subgraph of some 2-terminal SP-graph, thus the claim holds.

Given a graph G, we show an algorithm which either detects a minor G' of G with minimum degree 3, or build an SP-tree of a super-graph.

- If G is a single edge, then it has an SP-tree
- If G has a multiple edge, then delete one copy and test the remaining graph G'. If G' has an SP-tree, we can expand this to one of G. Otherwise G cannot be an SP-graph
- If G has a vertex of degree 2, contract an edge to its neighbour and test the remaining graph G'. As above, handle the success and failed cases
- If G has a vertex of degree 1 or 0, then simply delete it and test the remaining graph G'. If this is successful, we will show that we can expand the SP-tree by adding some edges to build an SP-gree of a super-graph of G.

Suppose the deleted vertex v has degree 1. Degree 0 does not happen assuming G is connected. Let w be the unique neighbour of v, w has some other neighbour x.

The SP-tree of G' has a leaf that stores the edge wx. We replace this by a P-node with one child storing wx and the other an S-node which stores the path w, v, x.

Notice we added an edge vx which is not in G but this is permitted as SP-graphs are subgraphs of 2-terminal SP-graphs.

Chapter 11

Branchwidth

11.1 e-Separations & Branch Decompositions

We can view the SP-tree as a partition of edges in the left and right subtrees.

Definition 11.1.1 (e-Separation)
A recursively defined partition of the edges.
If G has more than one edge, partition E into non-empty sets E_1, E_2, then recursively partition E_1, E_2 again.

Definition 11.1.2 (Rooted Branch Decomposition)
A rooted binary tree that describes an e-separation.
It stores edges in its leaves and has an interior node for every recursive partition.

Definition 11.1.3 (Width)
Given a branch decomposition T of G, for any link/arc a of T, the separator defined by a is the set of vertices v that have an incident edge in both subtrees defined by a.
The width of the branch decomposition T is the largest size of the separators of the links of T.

Definition 11.1.4 (Branch-Width)
The branch-width $\text{bw}(G)$ of a graph G is the smallest width of a branch decomposition of G.

Every 2-terminal SP-graph has branchwidth at most 2.

The $k \times k$ grid has branchwidth k.

What are closure properties of graphs of branchwidth k?

Definition 11.1.5 (Branch Decomposition)
A branch decomposition $\mathcal{B}$ consists of a tree T with maximum degree 3 such that the edges of G have been mapped to distinct leaves of T.

Proposition 11.1.1
A graph with a rooted branch decomposition of width w if and only if it has a branch decomposition of width w.

Proof
Any rooted branch decomposition is automatically also a branch decomposition with the same width.

Vice versa, given a branch decomposition T, we can turn it easily into a rooted branch decomposition as follows.

Delete (possibly repeatedly) any leaf of T which has no edge assigned to it.

Subdivide one link/arc if needed to create a node of degree 2. Root T at a node of degree 2.

If any node now has exactly one child, then contract the node into its child.

It is not difficult to see that neither operation affects the width and the result is a rooted branch decomposition.

Lemma 11.1.2
If G has branchwidth at most k, then any minor H of G also has branchwidth at most k.

Proof
Start with a branch decomposition of G that has tree T.

To delete an edge e of G equates simply to removing the leaf which stores that edge.

To contract an edge vw, replace any ocurrence of v, w by the identified vertex x, and remove the leaf that stores vw. Let T' be the resulting tree. Assume x occurs in separator $\sigma(a)$ for some link a. Then both components of $T' - a$ contains an edge incident to x. One of the sides used to contain the leaf that stored vw. Let xy be an edge that is in the other side of $T' - a$. Therefore this side of $T - a$ originally contained an edge vy or wy. Since vw is on the other side, at least one of v or w was in $\sigma(a)$ for tree T, and the width

has not increased.

Corollary 11.1.2.1
Every SP-graph has branchwidth at most 2.

11.2 Branchwidth & Treewidth

Theorem 11.2.1
For any graph G with $\mathrm{bw}(G) \geq 2$, we have

$$\mathrm{tw}(G) \leq \frac{3}{2}\,\mathrm{bw}(G) - 1$$

Proof
Start with a branch decomposition of G of minimal width, and use the same tree T for a tree decomposition.

The leaves of T are associated with an edge vw. Set its bags to be $\{v, w\}$. So leaves have bag-size

$$2 \leq \frac{3}{2}\,\mathrm{bw}(G)$$

For each interior node i, there are 3 separators $\sigma, \sigma', \sigma''$ on the three adjacent links. Set bag

$$X_i := \sigma \cup \sigma' \cup \sigma''$$

The connectivity condition is easily verified. Any vertex v appears on all bags of all paths connecting two leaves with an edge incident to v, and they forn a connected subtree.

It remains to analyze the width of a bag X_i of an interior node. If $v \in X_i$, then v is in one separator of an incident link of i. But then it must be in two of those separators since some path between leaves containing edges incident to v went through i. Since the 3 separators at i together contain at most $2\,\mathrm{bw}(G)$ vertices

$$|X_i| \leq \frac{3}{2}\,\mathrm{bw}(G)$$

This bound is not tight. One possible counterexample are the $k \times k$-grids.

Theorem 11.2.2
For any graph G, we have $\mathrm{bw}(G) \leq \mathrm{tw}(G) + 1$.

Proof
Assume G has a tree decomposition $\mathcal{T}$ of width k. For each edge $e = vw$, find a bag X that contains v and w and add a leaf-node with v, w incident to X. This covers edge e.

Then convert $\mathcal{T}$ into a tree decomposition of the same width for which the tree has maximum degree 3. This can be done by splitting nodes repeatedly and duplicating bags.

Let $a = ij$ be a link of the resulting (unrooted) branch decomposition. If v is a vertex that appears on both subtrees $T - a$, then necessarily v must be in both X_i, X_j.

Therefore, the separator at link a has size at most

$$|X_i \cap X_j| \leq k + 1$$

which proves the result.

Again this bound is not tight by the $k \times k$-grid example.

Thus asymptotically, the branchwidth and treewidth are the same.

11.3 Branch Decomposition of Planar Graphs

In all examples we have seen so far, we were able to indicate the corresponding partitions of edges via a noose (a closed Jordan curve intersecting the graph only at vertices). For planar graphs, we can actually always do this while having minimum branchwidth.

Definition 11.3.1 (Sphere-Cut Decomposition)
Formally, call a branch decomposition of a planar graph G sphere-cut decomposition if for every separator σ_a of a link a, there is a noose of G containing exactly the vertices of σ_a.

Theorem 11.3.1
If a planar graph has branchwidth k, then it also has a sphere-cut decomposition of width k.

Theorem 11.3.2
The branchwidth of a planar graph can be computed in quadratic time.

11.3.1 Spanning Trees of Small Height

We later use branch decompositions to obtain upper bounds on the treewidth for some planar graphs. For this, we need to connected branchwidth to the height of spanning trees.

Definition 11.3.2 (Co-Tree)
Given a spanning tree T in a connected planar graph G, let the co-tree T^* be the graph in G^* formed by the edges

$$(E - E(T))^*$$

i.e. take all edges of G that were not in T, and take their duals.

Lemma 11.3.3
Let G be a connected planar graph with spanning tree T. Then the co-tree T^* is a spanning tree of G^*.

Proof
Recall that a cut in G corresponds to a cycle in G^* and vice versa. If T^* were disconnected, then G^* has a cut separating parts of T^*. This cut corresponds to a cycle in T, which is a contradiction.

If T^* had a cycle, then this corresponds to a cut in G. This cut would make T disconnected.

So T^* is connected without a cycle, and hence a tree.

For triangulated graphs, we can use the co-tree to find a branch decomposition.

Lemma 11.3.4
Let G be a triangulated planar graph with a spanning tree T of height R. Then

$$\mathrm{bw}(G) \leq 2R + 1$$

Proof
Let r be the root of T, so any vertex has distance at most R from r, and any two vertices have distance at most $2R$ from each other.

Let T^* be the co-tree of T. This will be the basis for our branch decomposition.

For every edge e of G, create a leaf-node ℓ_e that stores e. Attach it to T^* as follows:

If $e \notin T$, then the dual edge e^* belongs to T^*. Subdivide e^* and attach ℓ_e at the subdivision vertex. Otherwise, if $e \in T$, then make ℓ_e adjacent to one of the nodes of T^* at a face

incident to e.

Since G is triangulated, every node of T^* has at most 3 attached edges (some in T^*, some attached to the create leaf-nodes), thus the resulting tree has maximum degree 3 and is a branch decomposition.

We claim this branch decomposition has width at most $2R+1$. Clearly any link incident to a leaf has a separator of size at most 2, namely, the two endpoints of the edge in the leaf.

Let a be a link not incident to a leaf which means it corresponds to the dual of an edge uv that was not in T. The unique uv-path in T forms a cycle C together with e.

Since all edges of C (except e) belong to T, the two subtrees of $T^* - a$ are separated by C. One subtree, say T_1 is entirely incide C and the other, say C_2 lives entirely outside C.

Because we attached leaves for edges only at incident faces, any edge stored in a leaf of T_1 must be on or inside C, while any edge stored in a leaf of T_2 must be on or outside C.

So if v is a vertex that has incident edges both in T_1, T_2, then v must be on C. Hence the separator at link a has length at most the number of vertices on cycle C, which is at most

$$2R+1$$

Corollary 11.3.4.1
If G is planar with a spanning tree T of height R. Then G has branchwidth at most $2R+1$.

Proof
Add edges until it becomes a planar triangulated graph G^+. T is still a spanning tree of heigh R in G^+. Thus

$$\mathrm{bw}(G) \leq \mathrm{bw}(G^+) \leq 2R+1$$

Corollary 11.3.4.2
Let G be a planar graph with a spanning tree of heigh R. Then G has treewidth at most $3R$.

Proof
We have

$$\begin{aligned}\operatorname{tw}(G) &\leq \left\lfloor \frac{3}{2}\operatorname{bw}(G) - 1 \right\rfloor \\ &\leq \left\lfloor \frac{3}{2}(2R+1) - 1 \right\rfloor \\ &= \left\lfloor 3R + \frac{1}{2} \right\rfloor \\ &= 3R\end{aligned}$$

Chapter 12

Pathwidth

12.1 Path Decomposition

Definition 12.1.1 (Path Decomposition)
A tree decomposition $\mathcal{T}$ for which the tree is a path.

Definition 12.1.2 (Pathwidth)
The pathwidth $\mathrm{pw}(G)$ of a graph G is the smallest k such that G has a path decomposition of width k.

Theorem 12.1.1
A graph G has pathwidth at most k if and only if G is a spanning subgraph of an interval graph G' with

$$\omega(G') \leq k+1$$

For any graph

$$\mathrm{tw}(G) \leq \mathrm{pw}(G)$$

But this is not tight for some graphs. One such example is obtained by subdividing each edge of $K_{1,3}$.

For some graphs the trewidth is precisely the pathwidth. For example the $k \times k$-grid.

12.2 Pathwidth & Trees

Theorem 12.2.1
A tree has pathwidth at most $k > 0$ if and only if there is a path P in T such that all subtrees of

$$T - P$$

have pathwidth at most $k - 1$.

Proof
($\Longrightarrow$) Suppose $\text{pw}(T) \leq k$, and let $X_1, \ldots, X_\ell$ be a path decomposition of width k. Pick some vertices v_ℓ and v_r that are in the leftmost and rightmost bag, respectively, and let the main path P be the path from v_ℓ to v_r in T.

It is entirely possible that $v_\ell = v_r$. Since P is connected, every bag contains at least one vertex of P. But then at most k vertices in each bag belong to $T - P$.

Therefore for any subtree of $T - P$, the induced path decomposition has width at most $k - 1$.

($\Longleftarrow$) Assume that T has such a path P. Enumerate

$$P : v_1, \ldots, v_K$$

and start with a path decomposition

$$\{v_1\}, \{v_1, v_2\}, \{v_2\}, \ldots, \{v_{K-1}\}, \{v_{K-1}, v_K\}, \{v_K\}$$

The single vertex bags will get repeated a number of times.

For each subtree T' of $T - P$, there is a unique vertex v_i of the path that has a neighbour in T'. Recursively compute a path decomposition of $\mathcal{P}'$ of T' with width at most $k - 1$.

Suppose $\mathcal{P}'$ has N bags. Repeat v_i N times so we have $N + 1$ copies. Paste $\mathcal{P}'$ onto the first N copies. This leaves one copy for use with other subtrees.

Repeating this for all subtrees of $T - P$ gives the desired decomposition.

Definition 12.2.1 (k-Caterpillar)
A 0-caterpillar is a single vertex.
A k-caterpillar for $k > 0$ is a tree T that has a path P such that every component of $T - P$ is an ℓ-caterpillar for some $\ell < k$. Path P is also called a spine of T.

Corollary 12.2.1.1
For any tree T, the pathwidth $\text{pw}(T)$ equals the smallest k for which T is a k-caterpillar.

It is then easy to see that $K_{1,3}$ has pathwidth 2, since it is not a caterpillar.

Let T be a rooted tree with height h. Then T is an h-caterpillar. This can be proven by induction on h. So the pathwidth is at most the height for any rooted tree T.

This bound can be tight. Consider the complete ternary tree of height h for example. We can prove by induction on h that $\text{pw}(T) \geq h$.

One can compute the complete ternary tree has height

$$h = \log_3(2n + 1)$$

so the pathwidth can be logarithmically larger than the treewidth.

In general, the height of rooted tree is a vast overestimation of the pathwidth.

Lemma 12.2.2
Any tree with n nodes has

$$\text{pw}(T) \leq \log n \in O(\log n)$$

12.3 Linear Arrangements

A linear arrangement is a vertex ordering.

Let $v_1, \ldots, v_n$ be a lienar arrangement of a graph G.

Definition 12.3.1 (Cut-Edges)
$\text{Cut}[i-1:i]$ is the edges that are in the cut induced by

$$\{v_1, \ldots, v_{i-1}\}$$

Definition 12.3.2 (Separation-Vertices of the Cut)
$\text{Vs}[i-1:i]$ are the vertices in $\{v_1, \ldots, v_{i-1}\}$ that have incident edges in $\text{Cut}[i-1:i]$.

Definition 12.3.3 (Cutwidth)
We say that G has cutwidth at most k

$$\text{cutwidth}(G) \leq k$$

if there is a linear arragement such that

$$|\text{Cut}[i-1:i]| \leq k$$

for all i.

Definition 12.3.4 (Vertex-Separation Number)
We say that G has vertex-separation number at most k

$$\text{vs}(G) \leq k$$

if there is a linear arragement such that

$$|\text{Vs}[i-1:i]| \leq k$$

for all i.

We have

$$\text{cutwidth}(G) \geq \left\lceil \frac{\Delta(G)}{2} \right\rceil$$

where Δ denotes the maximum degree. This is because the cut before and after a vertex v of maximum degree must together include all edges at v, so at least one of them has size at least half the maximum degree.

It is also apparent that

$$\text{vs}(G) \leq \text{cutwidth}(G) \leq \Delta \cdot \text{vs}(G)$$

More precisely, in any linear arragement and any cut $[i-1:i]$, we have

$$|\text{Vs}[i-1:i]| \leq |\text{Cut}[i-1,i]| \leq \Delta \cdot |\text{Vs}[i-1:i]|$$

since for every vertex in $\text{Vs}[i-1,i]$, there is at least one and at most Δ edges in $\text{Cut}[i-1:i]$.

There are graphs where the cutwidth is much bigger than the vertex-separation number. Consider $K_{1,n}$ for example.

One important distinction of cutwidth is that it is NOT closed under taking minors.

12.4 Pathwidth-Equivalence

It turns out that the vertex-separation number is actually just a different way to describe the pathwidth! We already have two other ways to describe pathwidth (aka path decomposition and interval graphs) and can add more equivalences.

Theorem 12.4.1
The following are equivalent for a graph G

(1) $\text{pw}(G) \leq k$

(2) G has a path decomposition where all bags contain at most $k+1$ vertices

(3) G is a subgraph of an interval graph H with $\omega(H) \leq k+1$

(4) We can assign intervals $I(v)$ to vertices such that for every edge vw, the intervals $I(v)$ and $I(w)$ interset, and at most $k+1$ intervals intersect in a point.

(5) G has vertex-separation number at most k.

Proof
The only non-trivial claims are (4) $\iff$ (5).

(4) $\implies$ (5) Sort the vertices by left endpoint of the intervals as $v_1, \ldots, v_n$, breaking ties arbitrarily.

Consider an arbitrary $\text{Cut}[i-1 : i]$ and let ℓ be the left endpoint of $I(v_i)$. For any $v_h \in \text{Vs}[i-1 : i]$ there is some neighbour v_j of v_h with $h < i \leq j$.

Vertex v_j was picked after v_i, which means that its left endpoint is no further left than ℓ, and so $I(v_j)$ does NOT intersect point $\ell - \epsilon$ for any ϵ.

Since $I(v_h)$ intersects $I(v_j)$, therefore $I(v_h)$ intersects ℓ. But then all vertices in $\text{Vs}[i-1 : i]$ intersect ℓ. Also by definition $I(v_i)$ intersects ℓ.

Since $v_i \notin \text{Vs}[i-1 : i]$, there are

$$|\text{Vs}[i-1 : i]| + 1$$

intervals that mutually intersect at point ℓ, hence

$$|\text{Vs}[i-1 : i]| \leq k$$

(5) $\implies$ (4) Let $v_1, \ldots, v_n$ be a linear arrangement with

$$|\text{Vs}[i-1 : i]| \leq k$$

for all i.

For vertex v_i, define $I(v_i)$ to be the interval that begins at i and ends at the index of the rightmost successor of v_i.

Clearly this is an interval representation of G. We know that any maximum clique C is achieved at the left endpoint of the interval for some vertex, so at integer i for some i.

Consider $\text{Cut}[i-1:i]$, and let $v_h \neq v_i$ be a vertex whose interval intersects i. By the definition of $I(v_h)$ this implies $h < i$ and v_h must have a neighbour v_j with $j \geq i$.

In consequence $v_h \in \text{Vs}[i-1:i]$. Since $I(v_i)$ also intersects i, we have

$$|C| \leq 1 + \text{Vs}[i-1:i] \leq k+1$$

as desired.

12.5 Cutwidth & Pathwidth

Definition 12.5.1 (Line Graph)
Graph obtained from G that has a vertex v_e for each $e \in E(G)$ and an edge $(v_e, v_{e'})$ if $e, e' \in E(G)$ share an edge.

Lemma 12.5.1
For any graph G, we have

$$\text{pw}(L(G)) \leq \text{cutwidth}(G) + \left\lfloor \frac{\Delta}{2} \right\rfloor - 1$$

Lemma 12.5.2
For any connected graph G with $n \geq 3$ vertices, we hvae

$$\text{cutwidth}(G) \leq \text{pw}(L(G))$$

So now we have the inequality

$$\text{cutwidth}(G) \leq \text{pw}(L(G)) \leq \text{cutwidth}(G) + \left\lfloor \frac{\Delta(G)}{2} \right\rfloor$$

Corollary 12.5.2.1
Let G be a graph with

$$\Delta(G) \leq 3$$

Then

$$\text{cutwidth}(G) = \text{pw}(L(G))$$

12.6 NP-Hardness of Computing Width Parameters

Definition 12.6.1 (Weighted Cutwidth)
Let G be a graph with positive edge-weights $w : E \to \mathbb{R}^+$. We say that G has weighted cutwidth at most k if G has a linear arrangement such that

$$\sum_{e \in \text{Cut}[i-1:i]} w(e) \leq k$$

for all cuts.

Theorem 12.6.1
Computing the weighted cutwidth of a graph is NP-hard even in trees.

Proof
We give a reduction from the PARTITION, which is a special case of the subset sum problem and known to be NP-hard since it can actually be used to decide the subset sum problem.

Given an instance of PARTITION, create an edge-weighted tree as follows: Start with $K_{1,n}$, and let c be the universal vertex and $v_1, \ldots, v_n$ be the other vertices.

Assign weight a_i to the edge cv_i.

Now consider any vertex order of this $K_{1,n}$, and let I be the indices of vertices athat appear before c. Then the cut to the left of c has weight

$$\sum_{i \in I} a_i$$

which is the cut to the right of c has weight

$$\sum_{i \notin I} a_i = 2S - \sum_{i \in I} a_i$$

It follows that $K_{1,n}$ has weighted cutwidth at most S if and only if there is a solution to PARTITION.

Definition 12.6.2 (Strongly NP-Hard)
A weighted decision problem where the weights are polynomial in the input-size.

It can be shown that computing the weighted cutwidth is strongly NP-hard. The current proof does not work since PARTITIONis not strongly NP-hard.

Theorem 12.6.2
Computing the weighted cutwidth of a graph is strongly NP-hard, even in trees.

Corollary 12.6.2.1
Computing the cutwidth is NP-hard even in a simple series-parallel graph of pathwidth 3.

Proof
We do a reduction from weighted cutwidth in a spider G. This is a graph obtained from $K_{1,n}$ by (repeatedly) subdividing edges.

First copy all vertices of G. Then for every edge $e = uv$ in G, define $w(e)$ new vertices in H and make them adjacent to u, v.

It is easy to check verify that H is series-parallel. It also has pathwidth 3 by using a main path/spine argument. Moreoever, the cutwidth of H is the weighted cutwidth of G.

Theorem 12.6.3
Computing the cutwidth is NP-hard even in planar graphs with maximum degree 3.

Corollary 12.6.3.1
Computing the pathwidth is NP-hard even in planar graphs with maximum degree 4.

Proof
We reduce from the proble of computing the cutwidth in a graph G that is planar and has maximum degree 3.

Observe that the line graph $L(G)$ then is also planar and has maximum degree 4. The result follows that

$$\text{cutwidth}(G) = \text{pw}(L(G))$$

by our prior work.

12.7 NP-Hardness of Treewidth

As it turns on, computing the treewidth is also NP-hard. However the proof is quite involved.

Computing the treewidth for planar graphs is an open problem whether it is truly NP-hard.

The related branchwidth IS polynomial in planar graphs. However the proof constructs graphs whose treewidths are very large.

Part III

Treewidth & Algorithms

Chapter 13

Dynamic Programming in Partial k-Trees

13.1 Dynamic Programming in Trees

13.1.1 Maximum Weight Independent Set

Problem 3 (Maximum Weight Independent Set)
Let G be a graph and $w : V \to \mathbb{R}$ a weight-function on the vertices. A maximum-weight independent set of G is a $I \subseteq V$ without edges between and $v, w \in I$ that maximizes the weight

$$\sum_{v \in I} w(v)$$

over all independent sets I.

The idea is that at every subtree T_v rooted at v of a tree T, we can recursively compute the maximum weight independent sets of $T_v - v$, and bring them together.

Specifically, we want to compute the best independent set which MUST contain v and the best independent set which is NOT allowed to contain v.

Let $\alpha(v, 0), \alpha(v, 1), \alpha(v, ?)$ be denote the weight of the maximum weight independent set at T_v which CANNOT, and must include v, and maximum overall.

$$\alpha(v,0) = \sum_{c \text{ child}} \alpha(c,?)$$
$$\alpha(v,1) = w(v) + \sum_{c \text{ child}} \alpha(c,0)$$
$$\alpha(v,?) = \max\{\alpha(v,1), \alpha(v,0)\}$$

We did not specify the base cases since they are trivial.

13.1.2 Maximum Matching

Problem 4
Let G be a graph and $w : E \to \mathbb{R}$ be a weight-function on the edges. A maximum-weight matching of G is a set M of edges with no common endpoint that maximizes

$$w(M) := \sum_{e \in M} w(e)$$

over all such sets

We will again describe the dynamic programming in terms of $\mu(v, Z)$ where

$$Z = \{0, 1, ?\}$$

For a leaf ℓ

$$\mu(\ell, 0) = 0$$
$$\mu(\ell, 1) = -\infty$$

The second case is $-\infty$ since there are no matchings for singleton graphs.

The recursive formula is given below. c_i enumerates all the children of v.

$$\mu(v,0) = \sum_{c_i} \mu(c_i,?)$$
$$\mu(v,1) = \max_{c_j}\{w(vc_j) + \mu(c_j,0) + \sum_{c_i \neq c_j} \mu(c_i,?)\}$$
$$= \sum_{c_i} \mu(c_i,?) + \max_{c_j}\{w(vc_j) - \mu(c_j,?) + \mu(c_j,0)\}$$

Putting the formula for $\mu(v,1)$ in the second format makes the entire algorithm run in linear time.

13.1.3 Solution

The idea is similar so we omit this content

13.1.4 Crucial Idea & Outlook

Dynamic Programming works in trees because the part-solution at subtrees do not influence each other once the situation at the root is specified.

We would hope that if a graph has small separators, similar logic would help us solve the same problems with some brute force enumeration at the separators themselves.

13.2 Dynamic Programming in 2-Terminal SP-Graphs

We now go up in treewidth to study partial 2-trees.

13.2.1 Independent Set

Observe that the SP-tree gives a natural tree-decomposition. Define

$$\alpha(G, Z)$$

to be the maximum weight of an independent set I in G satisfying

$$I \cap \{s, t\} = Z$$

for some $Z \subseteq \{s, t\}$.

The base case is when the graph is just the edge st.

$$\begin{aligned} \alpha(G, \varnothing) &= 0 \\ \alpha(G, \{s\}) &= w(s) \\ \alpha(G, \{t\}) &= w(t) \\ \alpha(G, \{s, t\}) &= -\infty \end{aligned}$$

The recursive formula is more complicated, especially if G was obtained from the series combination of graphs G_1, G_2.

For the parallel combination of G_1, G_2, it is much simpler. We need only avoid double counting.

$$\begin{aligned}
\alpha(G, \varnothing) &= \alpha(G_1, \varnothing) + \alpha(G_2, \varnothing) \\
\alpha(G, \{s\}) &= \alpha(G_1, \{s\}) + \alpha(G_2, \{s\}) - w(s) \\
\alpha(G, \{t\}) &= \alpha(G_1, \{t\}) + \alpha(G_2, \{t\}) - w(t) \\
\alpha(G, \{s,t\}) &= \alpha(G_1, \{s,t\}) + \alpha(G_2, \{s,t\}) - w(s) - w(t)
\end{aligned}$$

The reason which the series combination recursive formula is complicated is due to the fact that the identified vertex can be either in or not in the maximum weight independent set. We must try both possibilities. Suppose G is a series combination of G_1, G_2, whose terminals are s, x and x, t respectively.

$$\begin{aligned}
\alpha(G, \varnothing) &= \max\{\alpha(G_1, \varnothing) + \alpha(G_2, \varnothing), \alpha(G_1, \{x\}) + \alpha(G_2, \{x\}) - w(x)\} \\
\alpha(G, \varnothing) &= \max\{\alpha(G_1, \{s\}) + \alpha(G_2, \varnothing), \alpha(G_1, \{s,x\}) + \alpha(G_2, \{x\}) - w(x)\} \\
\alpha(G, \varnothing) &= \max\{\alpha(G_1, \varnothing) + \alpha(G_2, \{t\}), \alpha(G_1, \{x\}) + \alpha(G_2, \{x,t\}) - w(x)\} \\
\alpha(G, \varnothing) &= \max\{\alpha(G_1, \{s\}) + \alpha(G_2, \{t\}), \alpha(G_1, \{s,x\}) + \alpha(G_2, \{x,t\}) - w(x)\}
\end{aligned}$$

We avoid double counting $w(x)$ by subtracting the value off.

SP-Graphs

Recall that the SP-graph recognition algorithm either returns the SP-tree of one of its super graphs, or that we do not have an SP-graph.

We need only run the algorithm for that supergraph while ignoring edges which are not in in the SP-graph.

13.3 Dynamic Programming in Graphs of Pathwidth 3

Now we move up the treewidth again. To keep things simple, we will start with graphs of pathwidth 3.

13.3.1 Nice Path Decompositions

The first step is to preprocess the path decomposition so that it is easier to describe the dynamic programming formulas.

Let $v_1, \ldots, v_n$ be a linear arrangement of G with vertex-separation number 3. Create a path decomposition as follows:

Start with a one bag X_1 containing $v_1, \ldots, v_4$. We refer to X_1 as the leaf-bag.

There are at most 3 vertices in $\{v_1, \ldots, v_4\}$ that have neighbors in $\{v_5, \ldots, v_n\}$, due to the cut $[4:5]$. Let X_2 consist of these vertices. If there are only one or two such vertices, then arbitrarily add more vertices from X_1 to X_2. Observe that one of the vertices, $\{v_1, \ldots, v_4\}$ exists ONLY in X_1. This is why we will refer to X_2 as a forget-bag, and that sole vertex the forgotten-vertex.

Bag X_3 consists of the vertices of X_2 and the next vertex v_5. We refer to X_3 as an introduce-bag, and v_5 as the introduced-vertex.

In general, bag X_{2i} for $i \geq 1$ consists of all those vertex in the vertex-separation of cut

$$[i+3 : i+4]$$

with more vertices added from X_{2i-1} if needed so that $|X_{2i}| = 3$, and X_{2i} is referred to as a forget-bag.

On the other hand, Bag X_{2i+1} consists of all vertices in X_{2i}, plus the next vertex in the vertex-separation order (ie v_{i+4}). So $|X_{2i+1}| = 4$, and we call X_{2i+1} an introduce-bag.

Definition 13.3.1 (Nice Path Decomposition)
A path decomposition obtained with the description above.

Observe that $X_{2(n-4)+1}$ introduces v_n, and with this the entire graph is represented. So there are $O(n)$ bags in total.

Lemma 13.3.1
Any graph with pathwidth k has a nice path decomposition of width k.

Proof
The process described above did not rely on $pw(G) = 3$ and thus would create a nice path decomposition

$$X_1, \ldots X_\xi$$

such that $|X_i| = k+1$ for odd i, and $|X_i| = k$ for even i. Moreover X_i, X_{i+1} differs by exactly one vertex for all $i < \xi$.

13.3.2 Subgraphs

For trees and 2-terminal SP-graphs, the subgraphs in which we solve subproblems and combine them naturally derives from their structure.

For graphs of pathwidth 3, we set

$$Y_i := \bigcup_{j \leq i} X_i$$

and process the subgraphs $G[Y_i]$ for increasing i.

13.3.3 Maximum Weight Independent Set

For node i of the host path and some $Z \subseteq X_i$, define

$$\alpha(i, Z) = \text{maximum-weight of an independent set } I \text{ in } G[Y_i] \text{ such that } I \cap X_i = Z$$

The base case occurs at the leaf-node X_1 and is very simple. $Y_1 = X_1$ so we are just brute force checking if Z is an independent set.

$$\alpha(1, Z) = \begin{cases} w(Z), & Z \text{ is an independent set} \\ -\infty, & \text{else} \end{cases}$$

For an introduce bag X_{2i+1} with introduced vertex x.

$$\alpha(2i+1, Z) = \begin{cases} -\infty, & Z \text{ is not an independent set} \\ \alpha(2i, Z), & Z \text{ is an independent set and } x \notin Z \\ \alpha(2i, Z - x) + w(x) & Z \text{ is an independent set and } x \in Z \end{cases}$$

The importance of the introduce-vertex is illustrated here. We know that if there is an independent set I containing $Z - x$ in $G[Y_{2i}]$, then since x only appears in X_{2i+1}, any neighbor of x in Y_i also appears in X_{2i+1}. In particular, we would not be violating the definition of an independent set if we just add x to I as $I \cap N(x) = Z \cap N(x) = \varnothing$ since Z is an independent set.

The case for a forget-bag is similar for an S node in SP-graphs. If x is the forget-vertex, $Z \subseteq X_{2i}$, and I is an independent set in $G[Y_{2i}]$ such that $I \cap Z = Z$, then either $x \in I$ or not.

If so then $\alpha(2i-1, Z \cup \{x\})$ is the best among such I's. Otherwise $\alpha(2i-1, Z)$ is also the best among such I's. We take the maximum of the two.

$$\alpha(2i, Z) = \max\{\alpha(2i-1, Z \cup \{x\}), \alpha(2i-1, Z)\}$$

For arbitrary k, we need to store 2^{k+1} subsets $Z \subseteq X_j$ for $O(n)$ j's. We must also test if Z is an independent set, which can be done in $O(k)$ time since every vertex has at most k predecessors in the vertex-separation order.

Theorem 13.3.2
Let G be a graph with path decomposition of width k.
We can find the maximum-weight independent set in G in

$$O(2^k n)$$

time

Notice that our dynamic programming techniques require that we are given a path decomposition. Computing the minimal decomposition is equivalent to computing the pathwidth and is NP-hard.

13.4 Dynamic Programming in Partial k-Trees

We have nearly all the ingredients. Only minor modifications are required.

13.4.1 Nice Tree Decomposition

Definition 13.4.1 (Nice Tree Decomposition)
A tree decomposition $\mathcal{T}$ is nice if its tree can be rooted such that any node i have one of following types:

leaf node Node i has no children

introduce node Node i has exactly one child j, and $X_i = X_j \cup \{x\}$ for some introduced vertex $x \notin X_j$.

forget node Node i has exactly one child j, and $X_i = X_j - x$ for some forgotten vertex x

join node Node i has exactly two children j_1, j_2 where $X_i = X_{j_1} = X_{j_2}$

Lemma 13.4.1
Any partial k-tree has a nice tree decomposition of width k with $O(n)$ nodes.

Proof
It suffices to show this for when G is a k-tree, since adding edges until G has a p.e.o. $v_1, \ldots, v_n$ such that

$$\text{indeg}(v_i) = k$$

for all $i > k$ and getting a nice tree decomposition of the super graph will do the trick when G is strictly a partial k-tree.

Let $\mathcal{T}_1$ be the strong tree decomposition built from the p.e.o. This creates a bag X_i with

$$\text{pred}(v_i) \cup \{v_i\}$$

for vertex v_i.

We delete the bags $X_1, \ldots, X_k$ since they are subsets of X_{k+1} and root the tree decomposition at X_{k+1}. With this every bag has size $k+1$. Moreover, bag X_i was attached to X_h where v_h is the last predecessor of v_i. Since

$$\text{pred}(v_i) \subset \text{pred}(v_h)$$

any two adjacent bags have k vertices in common.

Let us modify $\mathcal{T}_1$ to get $\mathcal{T}_2$.

If i is a node in $\mathcal{T}_1$ with $d \geq 2$ children

$$j_1, \ldots, j_d$$

then replace i by a binary tree with $d-1$ interior nodes (forming a path) and d leaves. All replacement nodes receive the same bag that i had. This makes $\mathcal{T}_2$ a binary tree.

We need to make any node with 2 children a join node. If node h has children i, j, then subdivide the edges

$$hi, hj.$$

The identified vertices both have X_h. Then the original X_h is now a join node with identified vertices being forget nodes.

This makes $\mathcal{T}_2$ a nice tree decomposition.

13.4.2 Subgraphs

We need to define the subgraphs for which we solve the problem. For node i of the nice tree decomposition, let Y_i be the union of all bags that are descendants of i. We solve the subproblem on $G[Y_i]$ for all i in post order.

13.4.3 Independent Set

For any node i and $Z \subseteq X_i$, define

$$\alpha(i, Z) = \text{maximum-weight of an independent set } i \text{ in } G[Y_i] \text{ that satisfies } I \cap X_i = Z$$

The formulas are exactly the same for leaf, forget, and introduce nodes for graphs with path decomposition of size k since they all have indegree and outdegree equal to 1. The only difference lies in the join nodes.

Say h is a join node with children i, j. Then $X_h = X_i = X_j$ means that

$$\alpha(h, Z) = \alpha(i, Z) + \alpha(j, Z) - w(Z)$$

where we subtract $w(Z)$ to avoid double counting.

Theorem 13.4.2
Let G be a graph with a tree decomposition of width k. Then we can find the maximum weight independent set G in

$$O(k2^k n)$$

time.

13.5 Fixed-Parameter Tractability

The above algorithm for independent set in partial k-trees is one example of a problem that is fixed-parameter tractable in some graphs.

Definition 13.5.1 (Fixed-Parameter Tractable)
Given a problem with n being the input size and an additional parameter k.
The problem is fixed-parameter tractable if there is an algorithm with run time

$$O(f(k) \cdot \text{poly}(k, n))$$

13.6 Monadic Second-Order Logic

It turns out that weighted matching and weighted dominating sets can be solved with dynamic programming in partial k-trees as well.

While an explicit dynamic programming formulation can be developed for many problems, it is quite tedious to write them down.

The idea is to state a property of a graph as a logic formula, then evaluate that logic formula bottom up in the graph.

Definition 13.6.1 (Monadic Second-Order Logic)
A logic formula consisting of

Variables vertices or vertex sets

Membership Tests ie $v \in I$

Boolean Operations ie $\neg, \wedge, \vee, \implies$

Second-Order Predicates roughly means a two-variable boolean function. ie $\text{adj}(v, w)$ is true if and only if $vw \in E$

Monadic Quantifiers ie $\exists, \forall$ for variables but NOT predicates

We will write MSOL for short.

13.6.1 Examples

G has a triangle can be expressed as

$$\exists v_1, v_2, v_3 : \text{adj}(v_1, v_2) \wedge \text{adj}(v_2, v_3) \wedge \text{adj}(v_3, v_1) \wedge v_1 \neq v_2 \wedge v_2 \neq v_3 \wedge v_3 \neq v_1$$

To state that $v \in V$ has degree at least 2, state that it has two adjacent vertices.

Building on this, the statement that G has minimum degree 2 is the statement that $\forall v \in V, \deg(v) \geq 2$.

To describe that G is bipartite. Consider the following.

$$\text{Partition}(A, B, V) := \forall v \in V : (v \in A \vee v \in B) \wedge (v \in A \iff v \notin B)$$

$$\text{Bipartite}(V) := \exists A, B \subseteq V : \text{Partition}(A, B) \wedge \forall v, w \in V : \text{adj}(v, w) \implies (v \in A \wedge w \in B) \vee (v \in B$$

To state G is connected can be done by stating that G is not disconnected.

13.6.2 MSOL2

The property that "G has a perfect matching" cannot be expressed. This is formally provable with formal language theory.

However, if we allow a subset $E' \subseteq E$ of edges as variables, and we assume we have a predicate

$$\text{adj}(u, v, E')$$

which is true if and only if $uv \in E'$.

This variation is denoted MSOL2.

We can now for example describe a perfect matching as a subset of edges such that the graph formed by them has minimum degree and maximum degree 1.

More difficult is the Hamiltonian Cycle property. This can be stated as an edge set with minimum and maximum degree 2, for which the graph formed by it is connected.

$$\exists C \subseteq E : \neg\text{Disconnected}(V, C) \wedge \text{MinDegree2}(C) \wedge \text{MaxDegree2}(C)$$

13.6.3 Courcelle's Theorem

Theorem 13.6.1 (Courcelle)
Let Π be a graph property that can be expressed in MSOL2.
The problem of testing if a graph satisfies Π is fixed-parameter tractable in the treewidth. The run-time is lienar if the treewidth and size of the formula used to describe Π is constant.

13.7 Wrap Up

If we want to solve a problem in partial k-trees, the first thing is to try and express the problem in MSOL2.

If this fails it might be worth it to other options. Finally, if nothing works, a direct dynamic programming approach might be necessary.

Chapter 14

k-Outer-Planar Graphs

In this chapter, we study k-outer-planar graphs and specifically how to use them for approximation algorithms in planar graphs.

14.1 Combinatorial Properties

Proposition 14.1.1
Let G be a plane graph with onion peels

$$L_1, \dots, L_\ell$$

Any edge connects the same or adjacent onion peels.

Proposition 14.1.2
For any $i, k \geq 1$, the graph induced by onion peels

$$L_{i+1}, \dots, L_{i+k}$$

is k-outer-planar.

Lemma 14.1.3
Let G be a planar graph with spanning tree T of height k when rooted at $r \in V$. Then G is $(k+1)$-outer-planar.

Proof
Fix an embedding with r on the outer face.

Removing r removes all vertices of depth 0. Removing the children of r removes then all vertices of depth 1.

We can remove at most all vertices on depth $k+1$ before all vertices are removed. This inductively G is $(k+1)$-outer-planar.

14.2 Treewidth of k-Outer-Planar Graphs

We want to show that all k-outer-planar graphs have treewidth $O(k)$. To do so we create a supergraph H of G with a spanning tree of small height.

Lemma 14.2.1
Let G be a k-outer-planar graph and r a vertex on the outer face.
We can add edges to G while maintaining planarity such that G has a spanning tree T rooted at r with height at most k.

Proof
Let $L_1, \dots, L_k$ be the onion peels of G. We will prove by induction on i that we can add edges such that any vertex in L_i has distance at most i to r.

With this, a breadth-first search tree rooted at r gives the desired spanning tree.

The base case $i = 1$ is easy. Simply add edges through the outer face until r is adjacent to all vertices in L_1.

Now consider some $v \in L_i, i > 1$. It belongs to L_i and not any L_j for $j < i$. since after removing

$$L_1, \dots, L_{i-2},$$

v was NOT on the outer face. It belongs to L_i and not any L_j where $j > i$ since it was on the outer face after removing

$$L_1, \dots, L_{i-1}$$

This is possible only if at least one face f incident to v contained a vertex w belonging to L_{i-1}. Add the edge vw if it did not exist.

Since $d(w, r) \leq i - 1$ by induction, we are done.

Theorem 14.2.2
Every k-outer-planar graph has treewidth at most $3k$.

Proof
Add edges to G so that it has a spanning tree of height k or less. By our prior work, this spanning tree can be used to obtain a tree decomposition of width $3k$ of the supergraph of G. This is also a tree decomposition of G.

Theorem 14.2.3
Any k-outer-planar graph can be made triangulated by adding edges such that the result is $(k+1)$-outer-planar.

Proof
First add edges to G so that it has a spanning tree T of height k or less.

Then add more edges if needed to triangulate G. T continues being a spanning tree of height at most k.

The result then follows by our earlier lemma.

14.3 Baker's Approximation Scheme

The class of k-outer-planar graphs came to prominence through Baker's result which gave a PTAS. While it was discovered independently to treewidth, it can be re-phrased much more easily via treewidth.

14.3.1 Obtaining Outer-Planar Graphs

Lemma 14.3.1
Any planar G has an induced subgraph H that is outer-planar and has at least

$$\frac{n}{2}$$

vertices.

Proof
Fix a planar embedding of G and compute the onion peels L_i.

Let us consider two sets of vertices. V_1 consists of every other onion peel, and V_2 is the rest. Thus

$$V_1 := L \cup L_3 \cup \ldots, V_2 := L_2 \cup L_3 \cup \ldots$$

Define $G_1 = G \setminus V_1$. These have no edges between them by planarity and each onion peel

is a outer-planar graph. Hence G_1 is outer-planar as well.

The larger of $G \setminus V_1, G \setminus V_2$ must have at least $\frac{n}{2}$ vertices.

14.3.2 A $\frac{1}{2}$-Approximation

Lemma 14.3.2
Fix a planar embedding and compute onion peels L_i.
Set V_1 to be the union of all odd peels and V_2 the even ones.
Each V_i is outer-planar and thus has treewidth at most 3. We can compute the maximum weighted independent set I_i in $G \setminus V_i$ in linear time. Return the one with larger weight.
To see this is indeed within a factor of $\frac{1}{2}$ of any optimal independent set I^*, let

$$D_i := I^* \cap V_i, i = 1, 2$$

Since the V_1, V_2 forms a bipartition of V

$$wD_1 + wD_2 = wI*$$

and so

$$\min_i wD_i \leq \frac{wI^*}{2}$$

But $I^* - D_i$ is an independent set in G_i. Thus

$$\frac{wI^*}{2} \leq wI^* - wD_i \leq wI_i \leq wI^*$$

as required.

14.3.3 Obtaining k-Outer-Planar Graphs

Theorem 14.3.3
Any planar graph can be made into a k-outer-planar graph by removing at most

$$\frac{n}{k+1}$$

vertices.

Proof
For $1 \leq i \leq k+1$ define

$$V_i^{(k+1)} L = L_i \cup L_{i+(k+1)} \cup L_{i+2(k+1)}.$$

In other words $V_i^{(k+1)}$ contains every $(k+1)$st onion peel, and the index i indicates with which onion peel we start.

Define

$$G_i := G \setminus V_i.$$

Notice is is missing every $(k+1)$-st onion peel. Thus any connected component resides within k consecutive onion peels.

Any connected component of G_i is thus k-outer-planar. Therefore so is G_i.

By vertex-disjointness

$$\min_i |V_i^{(k+1)}| \leq \frac{n}{k+1}$$

Say the minimum is achieved at i^*, then

$$G_{i^*}$$

is k-outer-planar and has at least

$$\frac{k}{k+1} n$$

vertices as required.

14.3.4 Baker's PTAS

A PTAS for independent set consists of a class of algorithms which finds and independent set I of weight

$$wI \geq (1-\epsilon) w(I^*)$$

Theorem 14.3.4
There exists a PTAS for independent set in planar graphs.

Proof
Let $\epsilon > 0$ and define k to be the smallest integer such that

$$\frac{1}{k+1} < \epsilon$$

Let $V_i^{(k+1)}$ and G_i from before. Then $\text{tw}(G_i) \leq 3k$ by k-outer-planarity.

With dynamic programming, we can find the maximum independent set I_i in G_i. Let I be the best of these independent sets $I_1, \ldots, I_{k+1}$, so

$$wI = \max_i wI_i$$

To see I attains the approximation guarantee, let I^* be an optimal solution and

$$D_i := I^* \cap V_i$$

By vertex-disjointness

$$\min_i wD_i \leq \frac{wI^*}{k+1}$$

But as before

$$wI^* - wD_i \leq wI_i$$

Putting it all together

$$\begin{aligned} wI &= \max_i wI_i \\ &\geq \max_i wI^* - wD_i \\ &\geq wI^* - \frac{1}{k+1} wI^* \\ &\geq (1-\epsilon) wI^* \end{aligned}$$

by the choice of k.

14.3.5 Final Comments

The run time for Baker's PTAS is

$$O(k2^{3k}n)$$

which is snail-like for all but the smallest k. Anything beyond an approximation of $\frac{1}{10}$ would be hopelessly slow in practice.

To appreciate the power of PTAS, observe that it is NP-hard to approximate independent set within any factor better than n^ϵ. This holds even for graphs of bounded maximum degree.

Baker's scheme extends naturally to other problems which are polynomial on graphs of bounded treewidth.

Unfortunately the scheme does not work for finding the longest path. More sophisticated methods are required for these "global" problems.

The running time of the provided scheme runs in time

$$O\left(f\left(\frac{1}{\epsilon}\right)n\right)$$

The PTAS provided is fixed-parameter tractable in $\frac{1}{\epsilon}$. This is sometimes called a efficient PTAS (EPTAS). However the presented algorithm is NOT a full-PTAS (FPTAS) where the function $f\left(\frac{1}{\epsilon}\right)$ would have to be a polynomial.

There is an earlier PTAS for independent set in planar graphs. However the other one relies on planar separators and is slower. Moreover it is not a EPTAS.

Finally, observe that we would have instead defined onion peels by picking an arbitrary root and running a breadth-first search to use the layers of the BFS tree as onion peels.

Assume the graph class has locally bounded treewidth in the sense that a spanning tree of height k guarantees having treewidth $O(f(k))$ for some f, then all other steps of the PTAS can work using these layers instead of planar onion peels.

Thus Baker's approximation scheme can be generalized to any graph with locally bounded treewidth. This includes graphs of bounded genus and 1-planar graphs.

The concept can be further generalized using so-called nowhere dense graphs.

Chapter 15

Separators

Our previous work with dynamic programming in partial k-trees relied heavily on having separators of constant size. In the following chapter, we will assume graphs are vertex-weighted with non-negative weights.

15.1 Separators

Definition 15.1.1 (α-Separator)
For $\alpha \in (0, 1)$, an α-separator is a separator $S \subseteq V$ such that every flap C of S satisfies

$$wC \leq \alpha wV$$

$S = V$ is always a separator. But this is next to useless. Thus we are interested in separators of small cardinality.

Any α-separator is also an α'-separator fo $\alpha' > \alpha$. For the same reason as above, we would like to focus on finding separators where α is small.

For many applications, we are specifically interested in graphs of which have vertex weight 1 for all vertices.

Definition 15.1.2 (Uniform Weights)
$w(v) = 1$ for all $v \in V$.

We will always mean vertex-separators in this chapter.

15.2 Some Examples

Lemma 15.2.1
The $a \times b$-grid has a $\frac{1}{2}$-separator of size $\min(a, b)$.

Proof
This is not completely trivial when weights are not uniform.

Without loss of generality suppose that $a \leq b$. Let $c_1, \ldots, c_b$ be the columns of G. There is some i such that

$$\sum_{j=1}^{i-1} wc_j < \frac{1}{2}wV, \sum_{j=1}^{i} wc_j \geq \frac{1}{2}wV$$

Taking c_i suffices.

Lemma 15.2.2
The $k \times k$-grid does NOT have a $\frac{1}{2}$-separator of size $\frac{1}{2}k$, even for uniform weights.

Proof
Suppose for a contradiction that such a separator S exists. Then at least half the k rows do not contain any vertex of S. Moreoever, at least half the k columns do nto contain any vertex of k.

The union of those rows and columns form a connected component of $G - S$. The $\lceil \frac{k}{2} \rceil$ rows alone contain

$$k \cdot \left\lceil \frac{k}{2} \right\rceil \geq \frac{k^2}{2}$$

vertices.

Thus

$$wC > \frac{k^2}{2} = \frac{1}{2}wV$$

which is a contradiction.

Create a 3-tree as follows. Start with K_4 with each vertex set to weight 0. Insert into each face of K 3 additional vertices of weight 1 incident to each of the 3 vertices of the face.

Lemma 15.2.3
The 3-tree described does not have an α-separator of size 3 or less for the given vertex-weights and $\alpha < \frac{3}{4}$.

Proof
Suppose such a separator S exists.

If S contains at most 2 vertices of K_4, then $G - S$ is connected and

$$w(V - S) \geq 12 - 3 = \frac{3}{4}wV$$

On the other hand, if S contains exclusively vertices of K_4, the remaining vertex in $K_4 - S$ is adjacent to 9 other vertices in $G - S$. Thus

$$w(V - S) \geq 9 = \frac{3}{4}wV$$

Either way we arrive at a contradiction.

15.3 Trees

Root any T arbitrarily at r. For $v \in V$, define T_v to be the subtree rooted at v.

```
v = r
x = max_weighted_child(v)
while w(T_x) > w(V)/2:
  v = x
  x = max_weighted_child(v)

return v
```

Proposition 15.3.1
The returned vertex v^* is a $\frac{1}{2}$-separator of T.

Proof
We know that

$$wT_{v^*} > \frac{1}{2}wV$$

but every child x of v^* has $wT_x \leq \frac{1}{2}wV$.

But then the only other component of $T - v$ is $T - T_v$. This has weight at most

$$wV - \frac{1}{2}wV = \frac{1}{2}wV$$

but the choice of v^*.

15.3.1 Pathwidth of Trees

We claimed that

$$\mathrm{pw}(T) \leq \log_2 n$$

We now provide an easy proof.

Lemma 15.3.2
Every tree has pathwidth at most $\log_2 n$.

Proof
The argue by induction on n. For $n = 0$, this clearly holds.

Consider now $n > 1$. Find a $\frac{1}{2}$-separator of T with respect to uniform weights. Thus all connected components of $T \setminus v$ contain at most $\frac{n}{2}$ vertices.

By induction, each connected component of $T - v$ has pathwidth at most

$$\log\left(\frac{n}{2}\right) = \log n - 1$$

The result follows from the fact that T is a $\log_2 n$-caterpillar with v being the spine.

15.4 Partial k-Trees

Theorem 15.4.1
Every partial k-tree G has a weighted $\frac{1}{2}$-separator of size at most $k + 1$.

Proof
Take a tree decomposition of width k and let T be its tree.

For every $v \in V(G)$, there is a bag X which contains v. Add a leaf ℓ_v to X which contains only v. Give ℓ_v weight equal to $w(v)$. Set the weight of all original bags to 0.

Compute a $\frac{1}{2}$-separator i^* in the resulting tree. Then take

$$S := X_{i^*}$$

It looks like $k+1$ is "1 more" than necessary. It turns out not every k-tree has a $\frac{1}{2}$-separator of size k. This can be generalized to show that k-tree do not have a α-separator of size k for $\alpha < \frac{k}{k+1}$. On the other hand, every partial k-tree DOES have a $\frac{k}{k+1}$-separator of size k.

Lemma 15.4.2
The $k \times k$-grid has treewidth at least $\frac{k}{2}$.

Proof
If it had treewidth $\frac{k}{2} - 1$ or less, then it would have a $\frac{1}{2}$-separator of size $\frac{k}{2}$.

This contradicts our previous result.

15.5 Planar Graphs

Theorem 15.5.1 (Planar Separator)
Every planar graph G has a weighted $\frac{1}{2}$-separator of size

$$O(\sqrt{n})$$

Proof
make the graph k-outer-planar by removing a set S_1 of at most

$$\frac{n}{k+1}$$

vertices.

We know that $G - S_1$ has treewidth at most $3k$. Thus $G - S_1$ has a weighted $\frac{1}{2}$-separator S_2 of size at most $3k + 1$.

Combining them gives a weighted $\frac{1}{2}$-separator

$$S := S_1 \cup S_2$$

of size

$$\frac{n}{k+1} + 3k + 1$$

Using

$$k := \sqrt{*}\frac{n}{3} - 1$$

we get a separator of size at most

$$2\sqrt{3}\sqrt{n} - 2$$

It is worthwhile to mention that the planar separator can be found in $O(n)$ time. This is non-trivial but essentially follows from following the proof.

15.6 Divide & Conquer

Separators natually lend themselves to divide and conquer algorithms.

15.6.1 Shortest Cycles

Problem 5
Given a directed G with non-negative edge-weights, find a directed cycle C in G minimizing wC.

We can find a $\frac{1}{2}$-separator S ,recursively compute the shortest cycle in each flap, and keep the shortest of those.

Then we try to improve the solution by allowing it to include a vertex s of the separator.

This reduces to finding the shortest ss-dipath for each $s \in s$. We can to this by splitting a vertex into s_i, s_f where incoming edges to s_i are diverted to s_i, outgoing edges from s are moved to s_f, and the edge $s_i s_f$ is added. Djikstra's algorithm solves this in

$$O(m + n \log n) \subseteq O(n \log n)$$

time for planar graphs.

The recursion for running time $T(n)$ is then

$$T(n) = \sum_{\text{flap } C_i} T(n_i) + O(\sqrt{n} n \log n) \in O\left(n^{\frac{3}{2}} \log n\right)$$

15.6.2 Global Minimum Cut

Recall that global minimum cuts in planar graphs correspond to shortest cycles in the dual graph and vice versa.

We have already solved this for directed graphs. There is a way to solve it for undirected graphs as well.

15.6.3 Matching

Fix a $\frac{1}{2}$-separator S and recursively find the best matching in each flap. Set M' to be the union of all those matchings.

For each $s \in S$, add it back to G one-by-one. Any augmenting paths introduced necessarily begins at s. Find the best augmenting path can be done in

$$O(m \log n) = O(n \log n)$$

for planar graphs.

To find the best overall matching in G, the recursion satisfies

$$T(n) = \sum_{\text{flap } C_i} T(n_i) + O(\sqrt{n} n \log n) \in O\left(n^{\frac{3}{2}} \log n\right)$$

15.7 Sparator Hierarchies

The idea of finding a separator and then a separator for each flap is an important idea.

Definition 15.7.1 (α-Hierarchy)
An α-separator of a weighted G is a rooted T such that each node i is labelled with $S(i) \neq \varnothing$. where

(i) each $v \in V$ is stored at exactly one node

(ii) for every internal node i, $S(i)$ is an α-separator of $G(i)$, the graph formed by nodes stored at i or its descendents

(iii) for each internal node i, the graph of the children of i correspond to flaps of $G(i) - S(i)$

Let us write

$$n(i)$$

to denote the number of vertices in $G(i)$.

Proposition 15.7.1
Fix an α-separator hierarchy with respect to uniform weights.
Then any node i at depth d has

$$n(i) \leq n\alpha^d$$

and the heigh of the hierarchy is at most

$$\log_{\frac{1}{\alpha}} n$$

15.7.1 Path Decompositions

Given a separator hierarchy, create a path decomposition as follows.

For each leaf ℓ, set X_ℓ to be the union $S(a)$ of all nodes a that are ancestors of ℓ.

Arrange these bags according to the order of the leaves from left to right in the separator hierarchy.

Proposition 15.7.2
The resulting

$$X_1, \ldots, X_\xi$$

is a path decomposition of G.

Proof
We argue by induction on the height of node i that the bags at the leaves below i are a path decomposition for $G(i)$.

Clearly this holds at a leaf ℓ, since all vertices of $G(\ell)$ are added to X_ℓ.

If i is not a leaf, say it has children

$$j_1, \ldots, j_d$$

then the path decomposition can be viewed as follows.

First list all bags for $G(j_1), \ldots, G(j_d)$, and add $S(a)$ to those bags. Any edge of $G(a)$ is within one of the flaps, within $S(a)$, or connects a vertex in $S(a)$ with a neighbour in some flap.

Since no edge connect vertices in different flaps of $S(a)$, this covers all cases and so we have a path decomposition.

We can use this to obtain path decompositions of any graph class $\mathcal{G}$ for which all graphs have an α-separator and which is closed under taking induced subgraphs (induced-closed).

Theorem 15.7.3
Any induced-closed graph class $\mathcal{G}$ with α-separators of size $\sigma(n)$ has pathwidth at most

$$\sigma(n) \log_{\frac{1}{\alpha}}(n)$$

Proof
Build a α-hierarchy and get the path decomposition described. Any bag X satisfies

$$|X| \leq 1 + \sum_{d=0}^{h-1} \sigma(n) \leq 1 + \sigma(n) \log_{\frac{1}{\alpha}} n$$

Corollary 15.7.3.1
Every partial k-tree has pathwidth at most

$$(k+1)\log n$$

For planar graphs, applying this theorem would show that the pathwidth is at most

$$O(\sqrt{n}\log n)$$

But since the sizes of the separators decreases with each level, they follow a geometric series and we can get a better bound.

Lemma 15.7.4
If an induced-closed graph class $\mathcal{G}$ has α-separators of size $O(\sqrt{n})$, then all graphs in $\mathcal{G}$ have pathwidth

$$O(\sqrt{n})$$

Proof
Build an α-separator hierarchy, continuing to split until all leaves store only one vertex. We can do this such that

$$|S(i)| \leq c\sqrt{n(i)}$$

for all nodes i and some constant c.

Each bag X then has size

$$|X| \leq c\sqrt{n} + c\sqrt{\alpha n} + \cdots \leq c\sqrt{n}\sum_{i=0}^{\infty}\sqrt{a}^{i} \in O(\sqrt{n})$$

Corollary 15.7.4.1
Any planar graph has

$$\mathrm{tw}(G) \leq \mathrm{pw}(G) \in O(\sqrt{n})$$

Notice that this bound is asymptotically tight. A $k \times k$-grid has k^2 vertices and $tw = k$.

15.8 Generalized Separator Theorem

Theorem 15.8.1 (Generalized Separator)
If an induced-closed graph class $\mathcal{G}$ has α-separators of size $O(\sqrt{n})$, then for any $\epsilon > 0$, it also has ϵ-separators of size

$$O\left(\sqrt{\frac{n}{\epsilon}}\right)$$

15.8.1 Approximation Schemes

Recall Baker's PTAS for independent set in planar graphs. We give an older PTAS which is not quite as good in many aspects but contains useful ideas.

The Approximation

1) Use uniform weights. Set $\epsilon = \frac{\log\log n}{n}$ and compute a separator hierarchy
2) Each leaf ℓ has weight at most $\epsilon w V \leq \log\log n$, so its graph $G(\ell)$ has at most $\log\log n$ vertices. Find the optimal independent set with brute force in $O\left(n(\ell)2^{\log\log n}\right) = O\left(n(\ell)\log n\right)$ time
3) Set I to be the union over all leaves. This is the optimal independent set in $G - S$.
4) We know $S \in O\left(\sqrt{\frac{n}{\epsilon}}\right) = O\left(\frac{n}{\sqrt{\log\log n}}\right)$, say $|S| \leq \frac{cn}{\sqrt{\log\log n}}$ for constant c. The sublinearity will be crucial below
5) We know any optimal independent set I^* of G contains at least $\frac{n}{4}$ vertices by the 4-color theorem. In particular, the size of the optimal set is linear.
6) Since $I^* - S$ is an independent set in $G - S$, $\frac{|I|}{I^*} \geq 1 - \frac{|S|}{|I^*|} \geq 1 - \frac{4c}{\sqrt{\log\log n}}$

Analysis

As $n \to \infty$, the algorithm finds an independent set arbitrarily close to the maximum independent set in size.

How does this compare with Baker's Scheme? Baker's scheme is more versatile in that vertices are not necessarily uniformly weighted and we can choose the desired approximation ratio.

On the other hand, the current approach works without change to ANY graph class with α-separators and which is k-colourable for some constant k. This holds for toroidal graphs,

where Baker's scheme would require significantly more work to adapt.

Chapter 16

Approximation Treewidth

16.1 Approximating Treewidth

The idea is to test for the existence of a $\frac{1}{2}$-separator of size $k+1$. Not having this separator implies that G is not a partial k-tree. Otherwise, use this separator to build a tree decomposition.

Definition 16.1.1 ((α, W)-Separator)
Let G be a graph and $W \subseteq V$.
An (α, W)-separator is an α-separator with respect tot the weight function

$$\chi_W$$

Notice that a (α, W)-separator may have flaps of VERY uneven size. We only force there to be at most $\alpha|W|$ vertices of W in each flap.

Clearly any α-separator is an (α, W)-separator.

16.1.1 Separations

Even testing the existence of a (α, W)-separator is not easy. We will pivot to a slight variant.

Definition 16.1.2 (2-Way Separation)
A (2-way) separation of G is a partition

$$V = A_1 \cup S \cup A_2$$

such that there are no edges between A_1, A_2.

Definition 16.1.3 (Balance)
We say a 2-way separation has balance α if

$$wA_1, wA_2 \leq \alpha wV$$

Definition 16.1.4 (Separation Size)
$|S|$.

Clearly any separation of balance α gives a α-separator. The converse is not necessarily true.

Lemma 16.1.1
If G has an α-separator S, then G has a separation (A_1, S, A_2) with balance

$$\max\left\{\alpha, \frac{2}{3}\right\}$$

Proof
Let $C_1, \ldots, C_d$ be the flaps of $G - S$ where C_1 has maximum weight among them. Let i be the minimum index such that

$$\sum_{j=1}^{i} wC_j > \frac{1}{3}wV$$

To see that $wA_1 \leq \frac{2}{3}wV$, we have two cases.

The first case is where $wC_1 > \frac{1}{3}wV$, for which we set $A_1 := C_1$ and get

$$wA_1 = wC_1 \leq \alpha wV$$

Then

$$wA_2 \leq \frac{2}{3}wV$$

Otherwise $wC_1 > \frac{1}{2}wV$ and $i \geq 2$. By the choice of C_1, we have

$$wC_1 \leq \frac{1}{3}wV$$

But then by the choice of i

$$\sum_{j=1}^{i} wC_j \leq \frac{1}{3}wV$$

Thus

$$wA_i = \sum_{j=1}^{i-1} wC_j + wC_i \leq \frac{2}{3}wV$$

16.1.2 W-Separations

We say a separation with respect to χ_W having balance α to be a W-separation with balance α.

Lemma 16.1.2
Let G be a graph and $W \subseteq V$.
For any $\alpha \geq \frac{2}{3}$, there is an algorithm which finds the smallest W-separation of balance α in time

$$O\left(2^{|W|}|W|(n+m)\right)$$

Proof
If $|W| = 1$, the only possible W-separation uses $S = W$. Thus we may assume $|W| \geq 2$.

Let us first characterize such a W-separation (A_1, S, A_2). We have

$$|W \cap A_i| \leq \alpha|W|, i = 1, 2$$

Set $W_i := W \cap A_i$ and then add the vertices of $W \cap S$ to W_1, W_2 such that $|W_i| \leq \alpha|W|$ for $i = 1, 2$. This is possible since $\alpha \geq \frac{2}{3}$ (if $|W|_i = \frac{2}{3}|W|$, fill the other one) and $|W| \geq 2$.

We then have a (W_1, W_2)-separation where $W = W_1 \cup W_2$ is a bipartition with $|W_i| \leq \alpha|W|$ such that G has a separation (A_1, S, A_2) with

$$W_i \subseteq A_i \cup S$$

It suffices to search for the smallest (W_1, W_2)-separation over all possible partitions. There are at most $2^{|W|}$ such partitions. to find the smallest (W_1, W_2)-separation, we add a vertex s_i adjacent to each W_i and compute the number of vertex-disjoint s_1s_2-paths.

This can be solved in $O(k(n+m))$ time.

Clearly no (W_1, W_2)-separation can have fewer than k vertices. Since no $k+1$ vertex-disjoint paths exist, there must be a set S of size k which separates s_1, s_2.

Define A_i to be the set of all vertices reachable from S_i in $G - S$. This gives the desired partition (A_1, S, A).

Repeating this for all possible partitions of W means we can find the smallest W-separation of balance α.

16.1.3 W-Separations to Tree Decompositions

We now show that if a graph has small α-separators, it also has small treewidth.

Definition 16.1.5 (W-Tree-Decomposition)
Let G be a graph and $W \subseteq V$ a vertex set.
A W-tree-decomposition of G is a tree decomposition of G where all vertices of W appear in one bag.

Lemma 16.1.3
Let $\mathcal{G}$ be an induced-closed graph class where any $G \in \mathcal{G}$ has a $\left(\frac{D-1}{D}, W\right)$-separator of size at most w, for any vertex set $|W| \leq Dw + 1$ and $D = 2, 3, 4$.
Then G has a W-tree-decomposition $\mathcal{T}$ of width at most

$$(D+1)w$$

Theorem 16.1.4
Let $\mathcal{G}$ be an induced-closed graph class with $\frac{D-1}{D}$-separators of size at most w for some $D = 2, 3, 4$.
Any graph in $\mathcal{G}$ has treewidth at most $D + 1w$.

Proof
If G has at most Dw vertices, all vertices can be put into one bag at we are done. Assume then $|W| \leq Dw + 1$.

Find a $\left(\frac{D-1}{D}, W\right)$-separator S of size at most w.

Define Subgraphs & W-Sets Let $C_1, \ldots, C_d$ be the flaps of S. For $i \in [d]$, let U_i be the vertices of W appearing in C_i.

Since we had a $\frac{D-1}{D}$-separator, we get

$$|U_i| \leq \frac{D-1}{D}|W| \leq (D-1)w + \frac{1}{D}$$

which by integrality means

$$|U_i| \leq (D-1)w$$

Let G_i be the graph induced by C_i, S. Set

$$W_i := U_i \cup S$$

and notice that

$$|W_i| \leq Dw < |W|$$

Specifically, G_i is strictly smaller than G.

Get W_i-Tree-Decompositions for the Subgraphs By induction, there is a W_i-Tree-Decomposition $\mathcal{T}_i$ of G_i with width at most $(D+1)w$.

Combine the Tree Decompositions Combine

$$\mathcal{T}_1, \ldots, \mathcal{T}_d$$

by creating a bag X storing $W \cup S$.

This bag has size

$$|W| + |S| \leq (D+1)w + 1$$

and is the bottleneck for the width of this decomposition.

Connect S to the bags in $\mathcal{T}_i$ that contains W_i. This gives the desired tree decomposition.

Observe that this gives a converse of the fact where partial k-trees have $\frac{1}{2}$-separators of size $k+1$. This is seen with $D = 2, w = k+1$.

Corollary 16.1.4.1
Let $\mathcal{G}$ be an induced-closed graph class with $\frac{1}{2}$-separators of size $k+1$. Then the graphs in $\mathcal{G}$ have treewidth at most $3k+3$.

16.1.4 Approximation of Treewidth

Lemma 16.1.5
There is an algorithm which either correctly reports that $\mathrm{tw}(G) > k$, or finds a W-Tree-Decomposition of width at most $4k + 4$.
The run-time of this algorithm is

$$O(8^k k^2 n^2)$$

Proof
Assume that $m \geq kn$, or else it cannot be a partial k-tree.

With $D = 3, w = k + 1$, search for a W-separation of G with balance $\frac{2}{3}$ and size $k + 1$.

If $\mathrm{tw}(G) \leq k$, then this succeeds. G has a $\frac{1}{2}$-separator S of size $k + 1$ for any weight function (ie for χ_W). This can be turned into a separation of balance $\frac{2}{3}$ and size $k + 1$.

So if we cannot find such a separation, then $\mathrm{tw}(G) > k$. If we do, define subgraphs as in the previous lemma and recurse in the subgraphs. If the algorithm fails for either of those two, then we know $\mathrm{tw}(G) > k$.

If both recursions succeed, we can build a W-tree-decomposition for G and return it.

The next approximation is an "asymptotic" 4-approximation, which means that we ignore lower-order terms in the approximation factor.

Theorem 16.1.6
There is an asymptotic 4-approximation of treewidth with run time

$$O(8^{\mathrm{tw}(G)}\,\mathrm{tw}(G)n^2)$$

Proof
Run the algorithm for $k = 1, 2, \ldots, n$.

16.2 Grid Minors

For planar graphs, there are simpler methods to approximation treewidth.

For example, recall that the branchwidth of a planar graph G can be computed in polynomial time. Moreover,

$$\mathrm{bw}(G) \leq \mathrm{tw}(G) \leq \frac{3}{2}\,\mathrm{bw}(G) - 1.$$

So just computing the branchwidth gives a $\frac{3}{2}$-approximation of treewidth.

However this approach takes $O(n^2)$ time to get the value of the branchwidth and $O(n^4)$ to get the actual branch (tree) decomposition.

For planar graphs, observe that finding a $k \times k$-grid as a minor of a graph G, then surely the treewidth of G is bigger than k.

Lemma 16.2.1
Let G be a plane graph and W a set of at most $4k-4$ vertices on the outer face. Then either G has a $k \times k$-grid as a minor, or we can find a W-tree-decomposition of width at most $5k-5$.

Proof
Let us assume that $n \geq 5k-4$, or else put all vertices into one bag and we are done.

Furthermore, assume that G has at least $4k-4$ vertices on the outer face. Otherwise, set W to be the outer face vertices, delete an edge e on the outer face and recurse on $G' := G - e$. If we have a $k \times k$-grid minor, then it is also a minor of G. Otherwise a W-tree-decomposition of G' also covers e, thus it is also a W-tree-decomposition of G.

The main case is then when G has at least $4k-4$ vertices on the outer face, meaning we can pad W so that $|W| = 4k-4$ exactly.

Add 4 new vertices v_N, v_E, v_S, v_W on the outer face of G. Connect v_N to the first k vertices of W in clockwise order. Connect v_E to the last of the k vertices and also the next $k-1$ vertices. Similarly, connect v_S to the last of the previous vertices and the next $k-1$ vertices. Analogously for v_W. Clearly the new graph is still planar.

Find the maximum number of vertex-disjoint $v_N v_S$-paths in $G \cup \{v_N, v_S\}$ as well as the maximum number of vertex-disjoint $v_E v_W$-paths in $G \cup \{v_E, v_W\}$.

Case I: Suppose there are at most $k-1$ $v_N v_S$-paths. By Menger's theorem. there exists a cutting set S with $|S| \leq k-1$.

Let C_i be any connected component of $G - S$. Then C_i contains at most $3k-4$ vertices of W since it cannot contain both a neighbor of v_N and a neighbor of v_S by the definition of S. It follows that S is a $(3k-4 \leq \frac{3}{4}|W|, W)$-separator.

Similar to the previous lemma, Let U_i be the vertices of W appearing in C_i. Define $G_i := G[C_i \cup S]$ and $W_i := U_i \cup S$. Observe that W_i must be on the outer face of G_i.

Either G_i contains a $k \times k$-grid minor, or we can combine all tree decompositions to get a W-tree-decomposition of width

$$(D+1)w = 5k-5.$$

Case II: Suppose there are at most $k-1$ $v_W v_E$ paths. Symmetrically, we can recurse in subgraphs and either find a $k \times k$-grid minor or get a W-tree-decomposition of G with width at most $5k-5$.

Case III: There are at least k vertex-disjoint $v_N v_S$-paths

$$P_1, \ldots, P_k$$

and at least k vertex-disjoint $v_E v_W$-paths

$$P'_1, \ldots, P'_k.$$

Among all such path systems, pick the one minimizing the total number of edges. Let $V_{i,j}$ be the set of vertices in common to P_i, P'_j.

It can be shown that V_{ij} is necessarily a consecutive set of vertices along P_i. If two paths meet, separate, and meet again, then by re-routing paths appropriately, we can find a path system with fewer edges.

After contracting each $V_{i,j}$ into one vertex along the edges of P_i, we hence get a $k \times k$-grid minor of G.

Corollary 16.2.1.1
For any planar graph G and any fixed k, there is a $O(n^2)$ algorithm that either returns a $k \times k$-grid that is a minor of G or returns a tree decomposition of width at most

$$5k-5.$$

Proof
Follow the steps of the previous proof. The bottleneck is to test for k vertex-disjoint paths.

This is the Menger problem and can be solved in linear time with right-first search. Thus each recursion is linear time. Clearly then the run-time is

$$O(n^2)$$

Although the algorithm has the same run-time as the branchwidth approximation with a much worse approximation guarantee, it does return a grid-minor certificate, which is very useful.

Theorem 16.2.2
Let G be a planar graph.
The either G contains a $k \times k$-grid as a minor so that

$$\text{tw}(G) \geq k$$

or

$$\text{tw}(G) \leq 5k - 5.$$

16.2.1 Planar Minors

We are interested in finding minors.

Theorem 16.2.3
Testing whether H is a minor of G can be done in

$$O(f(h, \text{tw}(G)) \cdot n)$$

time for some function f.

Proof
We show how to phrase H is a minor of G in MSOL, where the size of the formula depends only on the size of H. Then Courcelle's theorem applies to give the result.

Enumerate the vertices of H as

$$v_1, \ldots, v_h.$$

Assuming that H was a minor of G, there is a sequence of edge contractions from a subgraph of G at the end of which is H.

Let V_i be the vertices of G contracted to create v_i of H. since we only contract edges, V_i is connected. Also, V_i is disjoint from V_j for $i \neq j$.

Finally for each edge v_iv_j of H, there must have existed at least one edge from some vertex in V_i to some vertex in V_j so that this edge is preserved during contraction and represents v_iv_j.

It is clear that the existence of such vertex-sets implies that H is a minor of G. We have given the formulas in MSOL to test for these properties. This concludes the proof.

If both H, G are planar, then this insight in combination with the previous corollary gives an algorithm to test whether H is a minor even if G does not have small treewidth.

Theorem 16.2.4
Let H, G be two planar graphs on h, n vertices.
We can test in

$$O(f(h)n + n^2)$$

time whether H is a minor of G.

Proof
Set $k := 3h$ and run the previous algorithm with this k. This takes quadratic time and either returns $\text{tw}(G) \leq 5k = 15h$ (dependent only on h), or G contains a $k \times k$-grid minor.

In the first case, We can test for the existence of a H-minor in $O(f(h)n)$-time. For the second case, any planar graph on h vertices is the minor of a $3h \times 3h$-grid, so no further work is needed.

16.2.2 Non-Planar Graphs

Theorem 16.2.5
There is a function $f(k)$ such that any graph of treewidth at least $f(k)$ contains a $k \times k$-grid minor.

Corollary 16.2.5.1
Let H be a planar graph with h vertices and G an arbitrary graph.
We can test in $O(f(h)\,\text{poly}(n))$ time whether H is a minor of G.

So testing for planar minors is polynomial time.

16.3 Exploiting Dichotomies

16.3.1 Longest Path and DFS

Problem 6 (Longest Path)
Given G and $\ell \geq 0$, decide if G has a path of length at least ℓ.

Clearly this is NP-hard since setting $\ell = n$ is the Hamiltonian Path problem.

Theorem 16.3.1
There is an algorithm to test if G has a path of length ℓ with run-time

$$O(f(\ell)(n+m)).$$

Proof
Run DFS on G. If the resulting DFS tree T has height ℓ or more, clearly the answer is YES.

Assume then that T has height at most $\ell - 1$, we claim that $\text{pw}(G) \leq \ell - 1$. This is seen by the DFS-tree property that no subtrees have edges between them. Thus we can easily construct a separator hierarchy from the DFS-order with each separator having size 1. Then the path decomposition from in which bags are ancestors has maximum width $\ell - 1$.

We can easily phrase the existence of a path of length at least ℓ in MSOL2 of size $O(\ell)$. Thus if the DFS-tree T has height at most $\ell - 1$, then we can solve the longest-path problem on G in time

$$O(f(\ell) \cdot n)$$

by Courcelle's theorem.

Bi-Dimensionality

We know show that for planar graphs, the dependency can be changed to $f(\sqrt{\ell})$.

Theorem 16.3.2
Given a planar graph G and an integer ℓ, there is an

$$O(f(\sqrt{\ell})n + n^2)$$

algorithm that tests whether G has a path of length ℓ.

Proof
Define $k := \lceil \sqrt{\ell - 1} \rceil$ and retrieve in quadratic time either a $k \times k$-minor or a tree decomposition at most $5k \in O(\sqrt{\ell})$.

In the first case, walk along the rows of the grid left-to-right down then right-to-left etc to find a path of length $k^2 - 1 \geq \ell$.

Otherwise, using the given tree decomposition, we can solve the longest path problem in

$$O(f'(k)n)$$

time using dynamic programming.

Observe that $f'(k) = \hat{f}(\sqrt{\ell})$ for some $\hat{f}$. This yields the result.

This type of result works for any problem $\mathcal{P}$ that contraction (minor) bi-dimensional.

Definition 16.3.1 (Contraction Bi-Dimensional)
If the problem $\mathcal{P}$ has solution size ℓ in a graph G, then it has a solution size at most ℓ in any minor of G.
In a $k \times k$-grid, problem $\mathcal{P}$ has solution size at least ck^2 for some constant c.
Finally, $\mathcal{P}$ is fixed-parameter tractable in the treewidth of the graph.

Chapter 17

The Graph Minor Theorem

17.1 $\mathcal{H}$-Free Graphs

Recall that the $\mathcal{H}$-minor-free graphs are the graphs which do not contain a graph from $\mathcal{H}$ as a minor.

The induced-$\mathcal{H}$-free graphs are the graphs which do not contain a graph from $\mathcal{H}$ as an induced subgraph.

For example, the chordal graphs are the induced-$\{C_4, C_5, C_6, \ldots\}$-free graphs.

The third of "free" graphs are the $\mathcal{H}$-subgraph-free graphs, which are the graphs which do not contain a graph of $\mathcal{H}$ as a subgraph.

An easy example are the bipartite graphs which are the $\{C_3, C_5, C_7, \ldots\}$-subgraph-free graphs.

To avoid repetition, we write $\mathcal{H}$-α-free, α-subgraph, and α-closed for

$$\alpha \in \{\text{subgraph, induced, minor}\}.$$

Lemma 17.1.1
If a graph class $\mathcal{G}$ is α-closed, then $\mathcal{G}$ is the same as the $\mathcal{H}$-α-free graphs for some $\mathcal{H}$.

Proof
Just take $\mathcal{H} := \overline{\mathcal{G}}$.

Corollary 17.1.1.1
There exists a set $\mathcal{H}$ of graphs such that the interval graphs are exactly the induced-$\mathcal{H}$-free graphs.

Proposition 17.1.2
There exist graph classes $\mathcal{G}$ that are α-closed. but any description as $\mathcal{H}$-α-free graphs require an infinite list $\mathcal{H}$.

Proof
Consider the chordal graphs, which are closed under taking induced subgraphs.

Any finite list however, cannot capture all $C_k, k \geq 4$.

However, specifically for minor-closed graphs classes we have seen so far (planar graphs), the list was always finite. In fact Robertson & Seymour proved this always holds in a series of 20 papers and over 500 pages.

The proof is clearly beyond the scope of this course, but we will see a glimpse of how it was accomplished.

17.2 The Minor-Poset

Any collection of finite graphs $\mathcal{G}$ admits a partial order by

$$H \preceq G \iff H \text{ is a minor of } G.$$

Recall that a chain in a partial order is a totally ordered subset. In the context of $\mathcal{G}$, we call this a minor chain.

17.2.1 Infinite Anti-Chains

Definition 17.2.1 (Anti-Chain)
The anti-chain of a partial order is a set of mutually incompatible elements.

Lemma 17.2.1
Suppose for any graph class $\mathcal{F}$, the minor poset defined by $\mathcal{F}$ has no infinite anti-chain. Then for any minor-enclosed graph class $\mathcal{G}$, there is a finite set $\mathcal{H}$ of graphs such that $\mathcal{G}$ is precisely the $\mathcal{H}$-minor-free graphs.

Proof
Set $\mathcal{H}_0 := \overline{\mathcal{G}}$. We know that G is precisely the $\mathcal{H}_0$-minor-free graphs.

For every $H \in \mathcal{H}_0$, let

$$\mathcal{H}_H := \begin{cases} \mathcal{H}_0, & H \text{ has no minor in } \mathcal{H}_0 \\ \mathcal{H}_0 \setminus \{H\}, & \exists H' \preceq H \in \mathcal{H}_0 \end{cases}$$

Observe that if any G contains H as a minor, it would contain H' as a minor. Thus G is also the $\mathcal{H}_H$-minor-free graphs for each H.

Let

$$\mathcal{H}_\omega := \bigcap_{H \in H_0} H_H.$$

By construction. $\mathcal{H}_\omega$ is an anti-chain of $\mathcal{H}_0$. By assumption this is finite.

So $\mathcal{H}_\omega$ is the finite list of forbidden minors characterizing $\mathcal{G}$.

Well-Quasi-Ordered

Definition 17.2.2 (Sink)
For a sequence of graphs

$$G_1, G_2, \ldots$$

A sink is a graph G_i such that there is no $j > i$ with

$$G_i \preceq G_j.$$

Definition 17.2.3 (Sink Sequence)
A sequence of graphs in $\mathcal{G}$ where all graphs are sinks.

In other words, we are weakening the condition of anti-chain to go only in one direction. We allow

$$G_i \preceq G_j \implies i > j.$$

Lemma 17.2.2
If for any graph class $\mathcal{F}$, the partial order defined by $\mathcal{F}$ has no infinite sink-sequence. Then for any graph class $\mathcal{G}$, the minor poset defined by $\mathcal{G}$ has no infinite anti chain.

Proof
Suppose that $\mathcal{G}$ has an infinite anti-chain. Enumerate a countable subset.

Clearly this is an infinite sink-sequence.

Definition 17.2.4 (Well-Quasi-Ordered)
A partial order is well-quasi-ordered (WQO) if it has no infinite sink sequence. Correspondingly, for any infinite sequences of graphs in $\mathcal{G}$, there are indices $i < j$ such that

$$G_i \preceq G_j.$$

Theorem 17.2.3 (Robertson-Seymour)
For any graph class $\mathcal{G}$, the minor poset is WQO.

17.3 Well-Quasi-Ordered Graph Classes

Lemma 17.3.1
The stars are WQO with respect to the minor relation.

Proof
Consider any infinite sequence of starts $S_1, S_2, \ldots$ and let s be the size of S_1.

If some $S_i, i > 1$ has size at least s, then $S_1 \preceq S_i$.

Otherwise, all starts $S_2, S_3, \ldots$ have size at most $s - 1$. Since there are infinitely many stars, some size must repeat say $S_i = S_j$. Then

$$S_i \preceq S_j.$$

17.3.1 Chain Subsequences

Lemma 17.3.2
Let $\mathcal{G}$ be a graph class that is WQO. Any sequence $G_1, G_2, \ldots$ contains an infinite minor chain as a subsequence.

Proof
Only a finite number of graphs in $G_1, G_2, \ldots$ can be sinks, else we are not WQO.

Let j_0 be the largest index of a sink and set

$$j_1 := j_0 + 1.$$

Since G_{j_1} is not a sink, there is some $j_2 > j_1$ such that

$$G_{j_1} \preceq G_{j_2}.$$

Since G_{j_2} is not a sink, we can repeat to get the desired minor sequence.

17.3.2 Star Pairs & Connected Components

Write

$$\mathcal{G} - \mathcal{G}'$$

to be all graphs with two connected components; one from $\mathcal{G}$ and one from $\mathcal{G}'$.

Lemma 17.3.3
If $\mathcal{G}, \mathcal{G}'$ are WQO, then so is $\mathcal{G} - \mathcal{G}'$.

Proof
Say that G_i is an infinite sequence in $\mathcal{G} - \mathcal{G}'$ where

$$G_i = C_i \cup C_i'.$$

Take an infinite minor subsequence C_{i_j}. Then apply the definition of WQO to C'_{i_j} to get $C'_{i_j} \preceq C'_{i_k}$.

Then

$$G_{i_j} \preceq G_{i_k}$$

by construction.

For a graph class $\mathcal{G}$, let $\mathcal{G}^*$ be the graph class where connected components belong to $\mathcal{G}$.

Lemma 17.3.4
If $\mathcal{G}$ is WQO, then so is G^*.

Theorem 17.3.5 (Kruskal)
The set $\mathcal{F}$ of forests is WQO.

17.3.3 More WQO Classes

Lemma 17.3.6
The class of partial k-trees is well-quasi-ordered.

Theorem 17.3.7
The class of planar graphs is WQO.

Proof
Let G_i be a sequence of planar graphs. Let n be the number of vertices of G_1.

Case I: Suppose all graphs in the sequence have treewidth at most $15n$, then this is a sequence of partial $15n$-trees, thus there are two graphs G_i, G_j with

$$G_i \preceq G_j.$$

Case II: If there is some graph G_j where $\text{tw}(G_j) > 15n$, then it has a $3n \times 3n$-grid minor. But then G_1 is a minor of a $3n \times 3n$ grid. Thus $G_1 \preceq G_i$ and we are done.

The ideas here can be generalized.

Let $G_1, G_2, \ldots$ be a sequence of graphs such that G_1 is planar. There is a function $f(k)$ such that any graph of treewidth greater than $f(k)$ contains a $k \times k$-grid. Apply this with $f(3n)$ and argue as in the previous theorem.

Now suppose G_1 has genus $g > 0$. We can apply induction on the genus to find $i < j$ with

$$G_i \preceq G_j.$$

This requires a "cutting" operation in which we reduce a graph of large genus to two subgraphs that are "smaller" (think smaller genus). With an alternative definition of "minor", the proof works.

This proves the result for any sequence of graphs which starts with a graph of finite genus. By letting the genus go to infinity (in a way which makes sense), we can finally attain the result.

17.4 Implications

An immediate corollary is that for every minor enclosed graph class, there exists polynomial time algorithm to test for membership of arbitrary G. Simply test if any of the finite list of

forbidden minors is a minor of G. This was shown if the forbidden minors are planar, but it is still generally true.