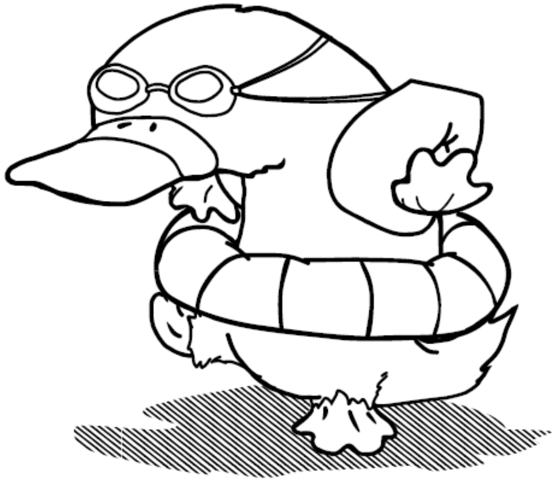


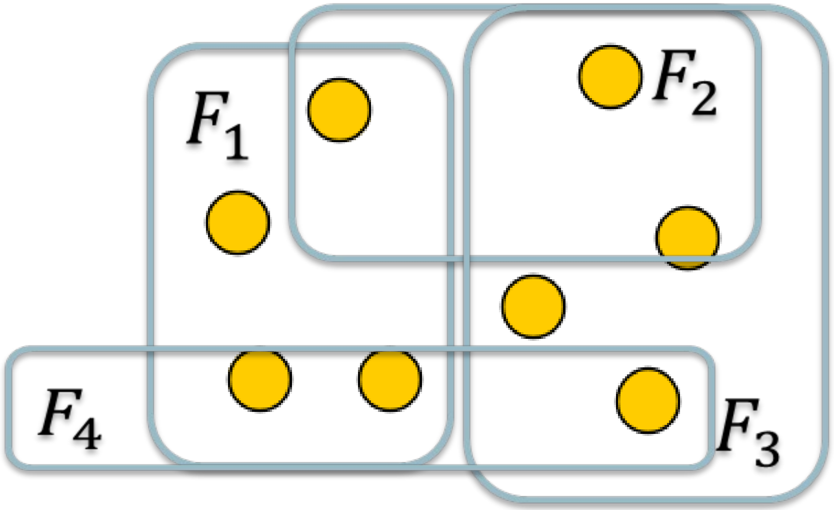
DYNAMIC PROGRAMMING

Slides from Prof. Daniel Marx

The SET COVER problem

- **Input:** A set family $\mathcal{F}$ over a universe U and an integer k
  Parameter: $|U|$
  Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of at most k sets, such that $\bigcup_{F \in \mathcal{F}'} F = U$?

- The subfamily $\mathcal{F}'$ **covers** the universe U
- SET COVER parameterized by the universe size is FPT
  - Algorithm with running time $2^{|U|} \cdot (|U| + |\mathcal{F}|)^c$
  - Based on **dynamic programming**



Dynamic programming for SET COVER

- Let $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$

- We define a DP table for $X \subseteq U$ and $j \in \{0,1,\ldots,m\}$
  $T[X, j]$ = min nr. of sets from $F_1, \ldots, F_j$ needed to cover X
  Or $+\infty$ if impossible

- The value $T[U, m]$ gives the minimum size of a set cover
  - To solve the problem, compute T using base cases and a recurrence

Filling the dynamic programming table

- $T[X, j]$ = min nr. of sets from $F_1, \dots, F_j$ needed to cover X

Base case: $j = 0$

$$T[X, j] = 0 \text{ if } X = \emptyset\text{, otherwise it is } +\infty$$

Recursive step: $j > 0$

$$T[X, j] = \min(T[X, j-1], 1 + T[X \setminus \mathrm{F}_j, j-1])$$

- Skip set F_j, or pay for F_j and afterwards cover $X \setminus \mathrm{F}_j$
- Each entry can be computed in polynomial time
  - $(|\mathcal{F}| + 1) \cdot 2^{|U|}$ entries in total

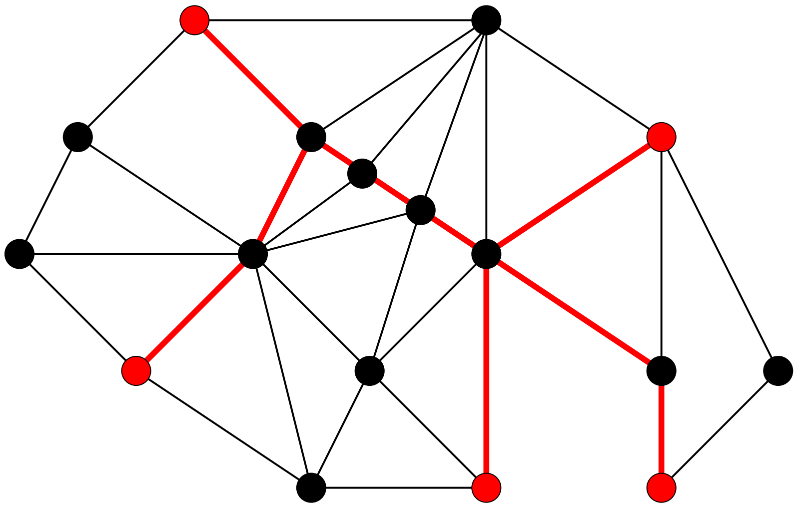
More on dynamic programming

- Dynamic programming is a memory-intensive algorithmic paradigm that yields FPT algorithms in various situations
  - Here: dynamic programming over **subsets** of U
  - Later: dynamic programming over **tree decompositions**
- Research challenge:
  - Determine whether the $2^{|U|}$ factor can be improved to $(2-\epsilon)^{|U|}$ for some $\epsilon > 0$

STEINER TREE

STEINER TREE

Task: Given a graph G with weighted edges and a set S of k vertices, find a tree T of minimum weight that contains S.



Known to be NP-hard. For fixed k, we can solve it in polynomial time: we can guess the Steiner points and the way they are connected.

Theorem: STEINER TREE is FPT parameterized by $k = |S|$.

STEINER TREE

Solution by dynamic programming. For $v \in V(G)$ and $X \subseteq S$,

$c(v, X) :=$ minimum cost of a Steiner tree of X that contains v

$d(u, v) :=$ distance of u and v

Recurrence relation:

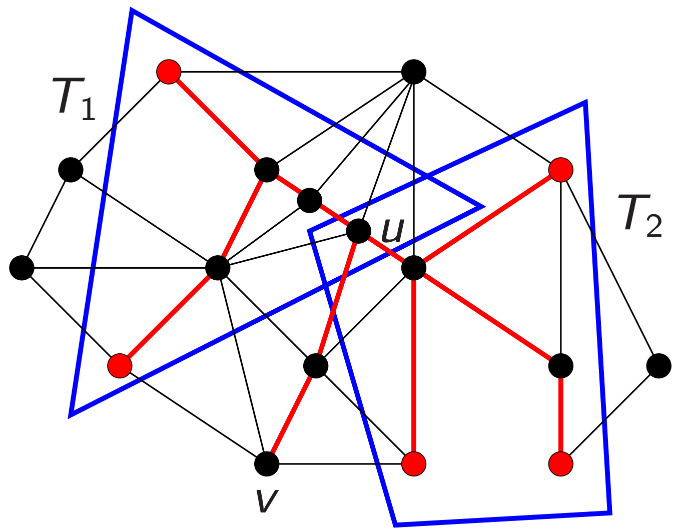
$$c(v, X) = \min_{\substack{u \in V(G) \\ \emptyset \subset X' \subset X}} c(u, X' \setminus u) + c(u, (X \setminus X') \setminus u) + d(u, v)$$

STEINER TREE

Recurrence relation:

$$c(v, X) = \min_{\substack{u \in V(G) \\ \emptyset \subset X' \subset X}} c(u, X' \setminus u) + c(u, (X \setminus X') \setminus u) + d(u, v)$$

- $\le$: A tree T_1 realizing $c(u, X' \setminus u)$, a tree T_2 realizing $c(u, (X \setminus X') \setminus u)$, and the path uv gives a (superset of a) Steiner tree of X containing v.

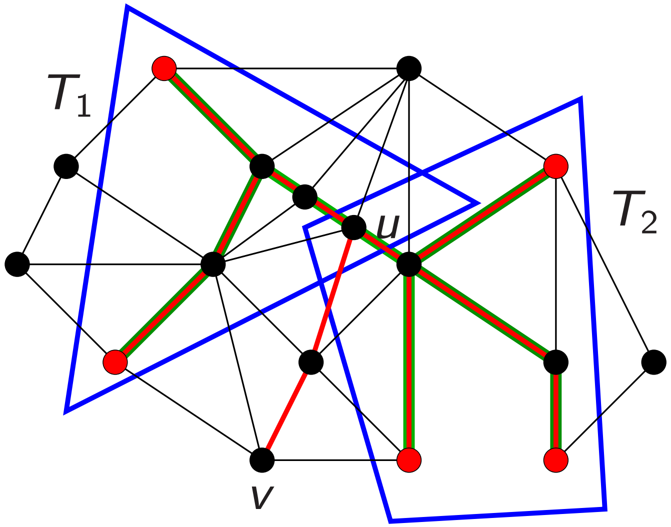


STEINER TREE

Recurrence relation:

$$c(v, X) = \min_{\substack{u \in V(G) \\ \emptyset \subset X' \subset X}} c(u, X' \setminus u) + c(u, (X \setminus X') \setminus u) + d(u, v)$$

- $\geq$: Suppose T realizes $c(v, X)$, let T' be the minimum subtree containing X. Let u be a vertex of T' closest to v. If $|X| > 1$, then there is a component C of $T \setminus u$ that contains a subset $\emptyset \subset X' \subset X$ of terminals. Thus T is the disjoint union of a tree containing $X' \setminus u$ and u, a tree containing $(X \setminus X') \setminus u$ and u, and the path uv.



STEINER TREE

Recurrence relation:

$$c(v, X) = \min_{\substack{u \in V(G) \\ \emptyset \subset X' \subset X}} c(u, X' \setminus u) + c(u, (X \setminus u) \setminus X') + d(u, v)$$

Running time:

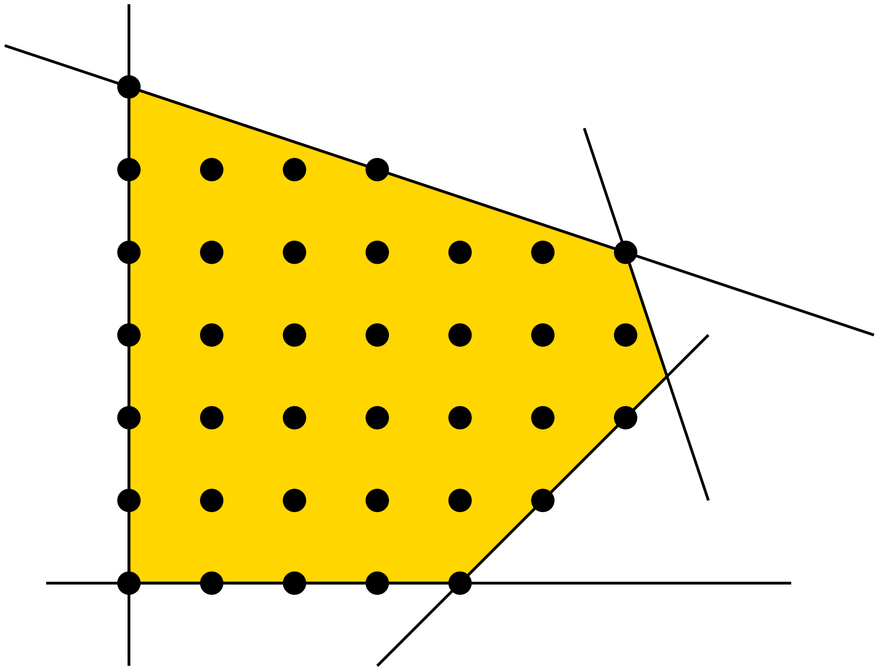
$2^k|V(G)|$ variables $c(v, X)$, determine them in increasing order of $|X|$. Variable $c(v, X)$ can be determined by considering $2^{|X|}$ cases. Total number of cases to consider:

$$\sum_{X \subseteq T} 2^{|X|} = \sum_{i=1}^{k} \binom{k}{i} 2^i \leq (1+2)^k = 3^k.$$

Running time is $O^*(3^k)$.

Note: Running time can be reduced to $O^*(2^k)$ with clever techniques.

Integer Linear Programming



Integer Linear Programming

Linear Programming (LP): important tool in (continuous) combinatorial optimization. Sometimes very useful for discrete problems as well.

$$\max c_1x_1 + c_2x_2 + c_3x_3$$

$$\text{s.t.}$$

$$x_1 + 5x_2 - x_3 \le 8$$

$$2x_1 - x_3 \le 0$$

$$3x_2 + 10x_3 \le 10$$

$$x_1, x_2, x_3 \in \mathbb{R}$$

Fact: It can be decided if there is a solution (feasibility) and an optimum solution can be found in polynomial time.

Integer Linear Programming

Integer Linear Programming (ILP): Same as LP, but we require that every x_i is integer.

Very powerful, able to model many NP-hard problems. (Of course, no polynomial-time algorithm is known.)

Theorem: ILP with p variables can be solved in time $p^{O(p)} \cdot n^{O(1)}$.

CLOSEST STRING

Task: Given strings $s_1, \ldots, s_k$ of length L over alphabet Σ, and an integer d, find a string s (of length L) such that $d(s, s_i) \leq d$ for every $1 \leq i \leq k$.

Note: $d(s, s_i)$ is the Hamming distance.

Theorem: CLOSEST STRING parameterized by k is FPT.

Theorem: CLOSEST STRING parameterized by d is FPT.

Theorem: CLOSEST STRING parameterized by L is FPT.

Theorem: CLOSEST STRING is NP-hard for $\Sigma = \{0, 1\}$.

Closest String

Task: Given strings $s_1, \dots, s_k$ of length L over alphabet Σ, and an integer d, find a string s (of length L) such that $d(s, s_i) \leq d$ for every $1 \leq i \leq k$.

Note: $d(s, s_i)$ is the Hamming distance.

Theorem: Closest String parameterized by k is FPT.

Theorem: Closest String parameterized by d is FPT.

Theorem: Closest String parameterized by L is FPT.

Theorem: Closest String is NP-hard for $\Sigma = \{0, 1\}$.

CLOSEST STRING

An instance with $k = 5$ and a solution for $d = 4$:

s_1	CBDCCACBB
s_2	ABDBCABDB
s_3	CDDBACCBD
s_4	DDABACCBD
s_5	ACDBDDCBC
	ADDBCACBD

Each column can be described by a partition $\mathcal{P}$ of $[k]$.

The instance can be described by an integer $c_{\mathcal{P}}$ for each partition $\mathcal{P}$: the number of columns with this type.

CLOSEST STRING

An instance with $k = 5$ and a solution for $d = 4$:

s_1	**CB**D**C**CACB**B**
s_2	A**B**DBCA**BDB**
s_3	**C**DDB**AC**CBD
s_4	**D**D**A**B**AC**CBD
s_5	A**C**DB**DD**CB**C**
	ADDBCACBD

Each column can be described by a partition $\mathcal{P}$ of $[k]$.

The instance can be described by an integer $c_{\mathcal{P}}$ for each partition $\mathcal{P}$: the number of columns with this type.

Closest String

An instance with $k = 5$ and a solution for $d = 4$:

s_1	CBDCCACBB
s_2	ABDBCABDB
s_3	CDDBACCBD
s_4	DDABACCBD
s_5	ACDBDDCBC
	ADDBCACBD

Each column can be described by a partition $\mathcal{P}$ of $[k]$.

The instance can be described by an integer $c_{\mathcal{P}}$ for each partition $\mathcal{P}$: the number of columns with this type.

Closest String

An instance with $k = 5$ and a solution for $d = 4$:

s_1	CBDCCACBB
s_2	ABDBCABDB
s_3	CDDBACCBD
s_4	DDABACCBD
s_5	ACDBDDCBC
	ADDBCACBD

Each column can be described by a partition $\mathcal{P}$ of $[k]$.

The instance can be described by an integer $c_{\mathcal{P}}$ for each partition $\mathcal{P}$: the number of columns with this type.

CLOSEST STRING

An instance with $k = 5$ and a solution for $d = 4$:

s_1	CBDCCACBB
s_2	ABDBCABDB
s_3	CDDBACCBD
s_4	DDABACCBD
s_5	ACDBDDCBC
	ADDBCACBD

Each column can be described by a partition $\mathcal{P}$ of $[k]$.

The instance can be described by an integer $c_{\mathcal{P}}$ for each partition $\mathcal{P}$: the number of columns with this type.

Closest String

An instance with $k = 5$ and a solution for $d = 4$:

s_1	CBDCCACBB
s_2	ABDBCABDB
s_3	CDDBACCBD
s_4	DDABACCBD
s_5	ACDBDDCBC
	ADDBCACBD

Each column can be described by a partition $\mathcal{P}$ of $[k]$.

The instance can be described by an integer $c_{\mathcal{P}}$ for each partition $\mathcal{P}$: the number of columns with this type.

Closest String

Each column can be described by a partition $\mathcal{P}$ of $[k]$.
The instance can be described by an integer $c_{\mathcal{P}}$ for each partition $\mathcal{P}$: the number of columns with this type.

Describing a solution: If C is a class of $\mathcal{P}$, let $x_{\mathcal{P},C}$ be the number of type $\mathcal{P}$ columns where the solution agrees with class C.

There is a solution iff the following ILP has a feasible solution:

$$\sum_{C \in \mathcal{P}} x_{\mathcal{P},C} \leq c_{\mathcal{P}} \qquad \forall \text{partition } \mathcal{P}$$

$$\sum_{i \notin C, C \in \mathcal{P}} x_{\mathcal{P},C} \leq d \qquad \forall 1 \leq i \leq k$$

$$x_{\mathcal{P},C} \geq 0 \qquad \forall \mathcal{P}, C$$

Number of variables is $\leq B(k) \cdot k$, where $B(k)$ is the no. of partitions of $[k]$
$\Rightarrow$ The ILP algorithm solves the problem in time $f(k) \cdot n^{O(1)}$.

Graph Minors

Neil Robertson

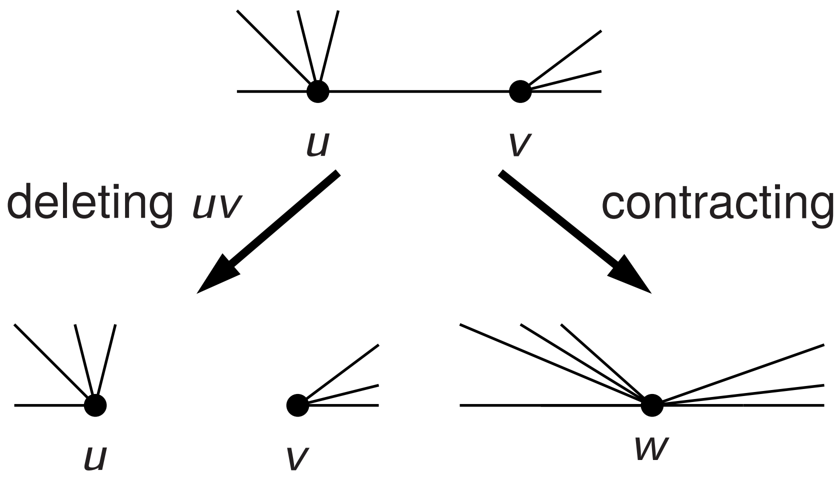
Paul Seymour

Graph Minors

- Some consequences of the Graph Minors Theorem give a quick way of showing that certain problems are FPT.
- However, the function $f(k)$ in the resulting FPT algorithms can be HUGE, completely impractical.
- History: motivation for FPT.
- Parts and ingredients of the theory are useful for algorithm design.
- New algorithmic results are still being developed.

Graph Minors

Definition: Graph H is a **minor** G ($H \leq G$) if H can be obtained from G by deleting edges, deleting vertices, and contracting edges.

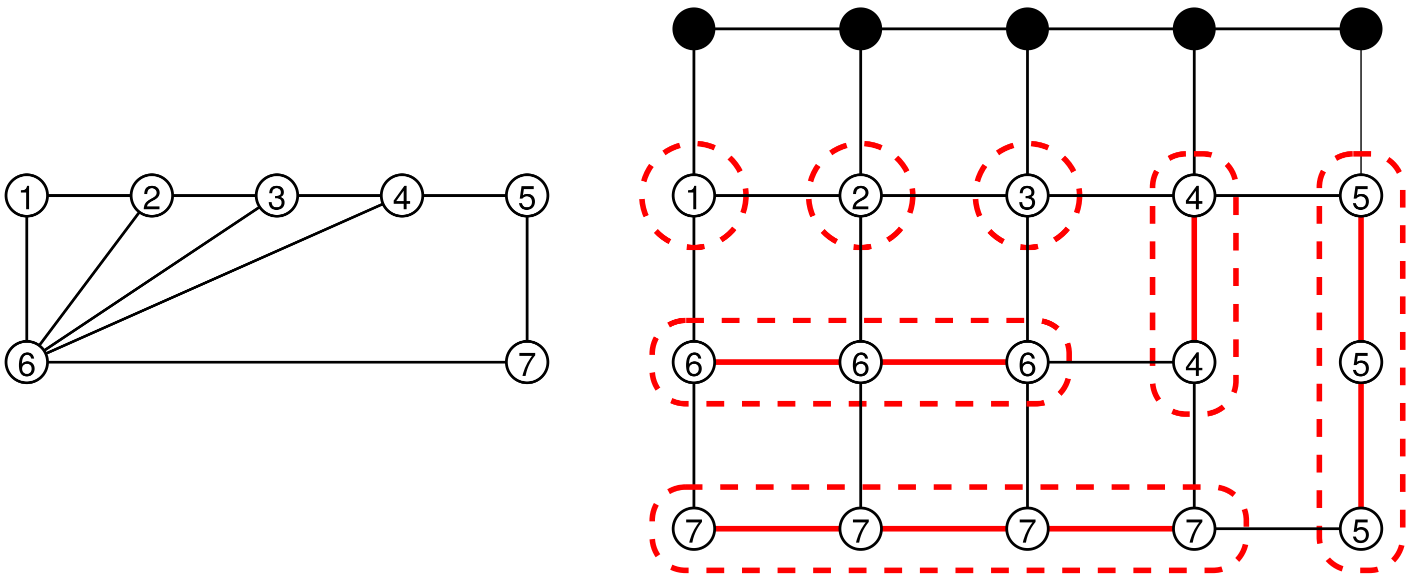


Example: A triangle is a minor of a graph G if and only if G has a cycle (i.e., it is not a forest).

Graph minors

Equivalent definition: Graph H is a **minor** of G if there is a mapping ϕ that maps each vertex of H to a connected subset of G such that

- $\phi(u)$ and $\phi(v)$ are disjoint if $u \neq v$, and
- if $uv \in E(G)$, then there is an edge between $\phi(u)$ and $\phi(v)$.



Minor closed properties

Definition: A set $\mathcal{G}$ of graphs is **minor closed** if whenever $G \in \mathcal{G}$ and $H \leq G$, then $H \in \mathcal{G}$ as well.

Examples of minor closed properties:

planar graphs

acyclic graphs (forests)

graphs having no cycle longer than k

empty graphs

Examples of not minor closed properties:

complete graphs

regular graphs

bipartite graphs

Forbidden minors

Let $\mathcal{G}$ be a minor closed set and let $\mathcal{F}$ be the set of "minimal bad graphs": $H \in \mathcal{F}$ if $H \notin \mathcal{G}$, but every proper minor of H is in $\mathcal{G}$.

Characterization by forbidden minors:

$$G \in \mathcal{G} \iff \forall H \in \mathcal{F}, H \not\leq G$$

The set $\mathcal{F}$ is the **obstruction set** of property $\mathcal{G}$.

Forbidden minors

Let $\mathcal{G}$ be a minor closed set and let $\mathcal{F}$ be the set of "minimal bad graphs": $H \in \mathcal{F}$ if $H \notin \mathcal{G}$, but every proper minor of H is in $\mathcal{G}$.

Characterization by forbidden minors:

$$G \in \mathcal{G} \iff \forall H \in \mathcal{F}, H \not\leq G$$

The set $\mathcal{F}$ is the **obstruction set** of property $\mathcal{G}$.

Theorem: [Wagner] A graph is planar if and only if it does not have a K_5 or $K_{3,3}$ minor.

In other words: the obstruction set of planarity is $\mathcal{F} = \{K_5, K_{3,3}\}$.

Does every minor closed property have such a finite characterization?

Graph Minors Theorem

Theorem: [Robertson and Seymour] Every minor closed property $\mathcal{G}$ has a finite obstruction set.

Note: The proof is contained in the paper series “Graph Minors I–XX”.

Note: The size of the obstruction set can be astronomical even for simple properties.

Graph Minors Theorem

Theorem: [Robertson and Seymour] Every minor closed property $\mathcal{G}$ has a finite obstruction set.

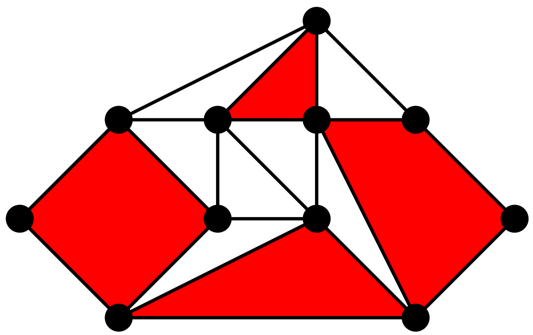
Note: The proof is contained in the paper series “Graph Minors I–XX”.
Note: The size of the obstruction set can be astronomical even for simple properties.

Theorem: [Robertson and Seymour] For every fixed graph H, there is an $O(n^3)$ time algorithm for testing whether H is a minor of the given graph G.

Corollary: For every minor closed property $\mathcal{G}$, there is an $O(n^3)$ time algorithm for testing whether a given graph G is in $\mathcal{G}$.

Applications

PLANAR FACE COVER: Given a graph G and an integer k, find an embedding of planar graph G such that there are k faces that cover all the vertices.



One line argument:

For every fixed k, the class $\mathcal{G}_k$ of graphs of yes-instances is minor closed.

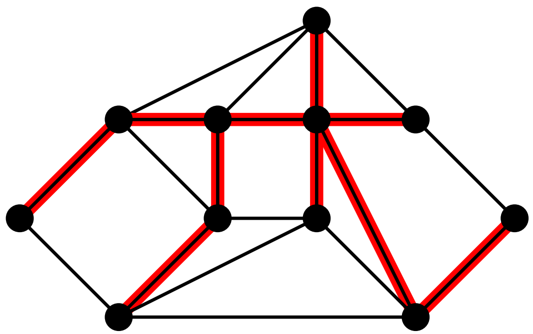
$$\Downarrow$$

For every fixed k, there is a $O(n^3)$ time algorithm for PLANAR FACE COVER.

Note: non-uniform FPT.

Applications

k-LEAF SPANNING TREE: Given a graph G and an integer k, find a spanning tree with **at least** k leaves.



Technical modification: Is there such a spanning tree for at least one component of G?

One line argument:

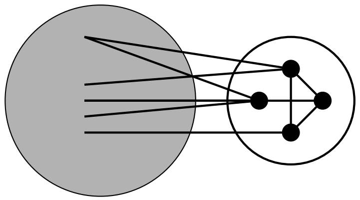
For every fixed k, the class $\mathcal{G}_k$ of no-instances is minor closed.

$$\Downarrow$$

For every fixed k, k-LEAF SPANNING TREE can be solved in time $O(n^3)$.

$\mathcal{G} + k$ vertices

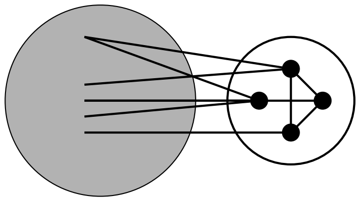
Let $\mathcal{G}$ be a graph property, and let $\mathcal{G} + kv$ contain graph G if there is a set $S \subseteq V(G)$ of k vertices such that $G \setminus S \in \mathcal{G}$.



Lemma: If $\mathcal{G}$ is minor closed, then $\mathcal{G} + kv$ is minor closed for every fixed k.
$\Rightarrow$ It is (nonuniform) FPT to decide if G can be transformed into a member of $\mathcal{G}$ by deleting k vertices.

$\mathcal{G} + k$ ***vertices***

Let $\mathcal{G}$ be a graph property, and let $\mathcal{G} + kv$ contain graph G if there is a set $S \subseteq V(G)$ of k vertices such that $G \setminus S \in \mathcal{G}$.



Lemma: If $\mathcal{G}$ is minor closed, then $\mathcal{G} + kv$ is minor closed for every fixed k.

$\Rightarrow$ It is (nonuniform) FPT to decide if G can be transformed into a member of $\mathcal{G}$ by deleting k vertices.

- If $\mathcal{G}$ = forests $\Rightarrow$ $\mathcal{G} + kv$ = graphs that can be made acyclic by the deletion of k vertices $\Rightarrow$ FEEDBACK VERTEX SET is FPT.
- If $\mathcal{G}$ = planar graphs $\Rightarrow$ $\mathcal{G} + kv$ = graphs that can be made planar by the deletion of k vertices (k-apex graphs) $\Rightarrow$ k-APEX GRAPH is FPT.
- If $\mathcal{G}$ = empty graphs $\Rightarrow$ $\mathcal{G} + kv$ = graphs with vertex cover number at most k $\Rightarrow$ VERTEX COVER is FPT.