



# ***FPT algorithmic techniques***

Dániel Marx

Budapest University of Technology and Economics, Hungary

AGAPE'09 Spring School on Fixed Parameter and Exact Algorithms

May 25-26, 2009, Lozari, Corsica (France)

# ***FPT algorithmic techniques***

- ⑥ Significant advances in the past 20 years or so (especially in recent years).
- ⑥ Powerful toolbox for designing FPT algorithms:

**Bounded Search Tree**

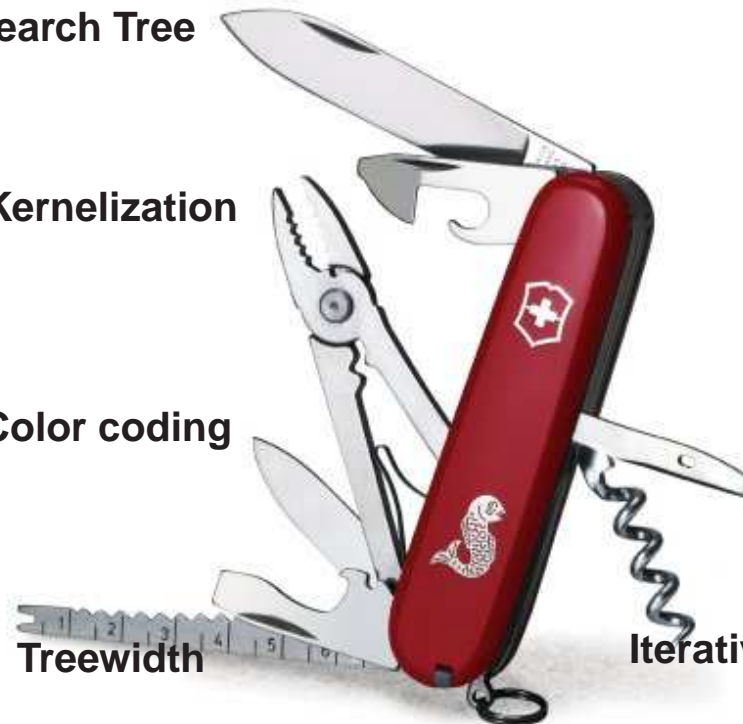
**Kernelization**

**Color coding**

**Treewidth**

**Graph Minors Theorem**

**Iterative compression**



# Goals

- ⑥ Demonstrate techniques that were successfully used in the analysis of parameterized problems.
- ⑥ There are two goals:
  - △ Determine quickly if a problem is FPT.
  - △ Design fast algorithms.
- ⑥ Warning: The results presented for particular problems are not necessarily the best known results or the most useful approaches for these problems.
- ⑥ Conventions:
  - △ Unless noted otherwise,  $k$  is the parameter.
  - △  $O^*$  notation:  $O^*(f(k))$  means  $O(f(k) \cdot n^c)$  for some constant  $c$ .
  - △ Citations are mostly omitted (only for classical results).
  - △ We gloss over the difference between decision and search problems.

# ***Kernelization***



# Kernelization

**Definition:** Kernelization is a polynomial-time transformation that maps an instance  $(I, k)$  to an instance  $(I', k')$  such that

- ⑥  $(I, k)$  is a yes-instance if and only if  $(I', k')$  is a yes-instance,
- ⑥  $k' \leq k$ , and
- ⑥  $|I'| \leq f(k)$  for some function  $f(k)$ .

**Simple fact:** If a problem has a kernelization algorithm, then it is FPT.

**Proof:** Solve the instance  $(I', k')$  by brute force.

**Converse:** Every FPT problem has a kernelization algorithm.

**Proof:** Suppose there is an  $f(k)n^c$  algorithm for the problem.

- ⑥ If  $f(k) \leq n$ , then solve the instance in time  $f(k)n^c \leq n^{c+1}$ , and output a trivial yes- or no-instance.
- ⑥ If  $n < f(k)$ , then we are done: a kernel of size  $f(k)$  is obtained.

# Kernelization for VERTEX COVER

**General strategy:** We devise a list of reduction rules, and show that if none of the rules can be applied and the size of the instance is still larger than  $f(k)$ , then the answer is trivial.

Reduction rules for VERTEX COVER instance  $(G, k)$ :

**Rule 1:** If  $v$  is an isolated vertex  $\Rightarrow (G \setminus v, k)$

**Rule 2:** If  $d(v) > k \Rightarrow (G \setminus v, k - 1)$

If neither Rule 1 nor Rule 2 can be applied:

- ⑥ If  $|V(G)| > k(k + 1) \Rightarrow$  There is no solution (every vertex should be the neighbor of at least one vertex of the cover).
- ⑥ Otherwise,  $|V(G)| \leq k(k + 1)$  and we have a  $k(k + 1)$  vertex kernel.

# Kernelization for VERTEX COVER

Let us add a third rule:

**Rule 1:** If  $v$  is an isolated vertex  $\Rightarrow (G \setminus v, k)$

**Rule 2:** If  $d(v) > k \Rightarrow (G \setminus v, k - 1)$

**Rule 3:** If  $d(v) = 1$ , then we can assume that its neighbor  $u$  is in the solution  $\Rightarrow (G \setminus (u \cup v), k - 1)$ .

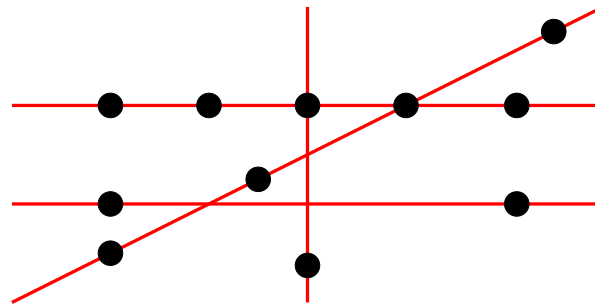
If none of the rules can be applied, then every vertex has degree at least 2.

$\Rightarrow |V(G)| \leq |E(G)|$

- ⑥ If  $|E(G)| > k^2 \Rightarrow$  There is no solution (each vertex of the solution can cover at most  $k$  edges).
- ⑥ Otherwise,  $|V(G)| \leq |E(G)| \leq k^2$  and we have a  $k^2$  vertex kernel.

# COVERING POINTS WITH LINES

**Task:** Given a set  $P$  of  $n$  points in the plane and an integer  $k$ , find  $k$  lines that cover all the points.



**Note:** We can assume that every line of the solution covers at least 2 points, thus there are at most  $n^2$  candidate lines.

**Reduction Rule:**

If a candidate line covers a set  $S$  of more than  $k$  points  $\Rightarrow (P \setminus S, k - 1)$ .

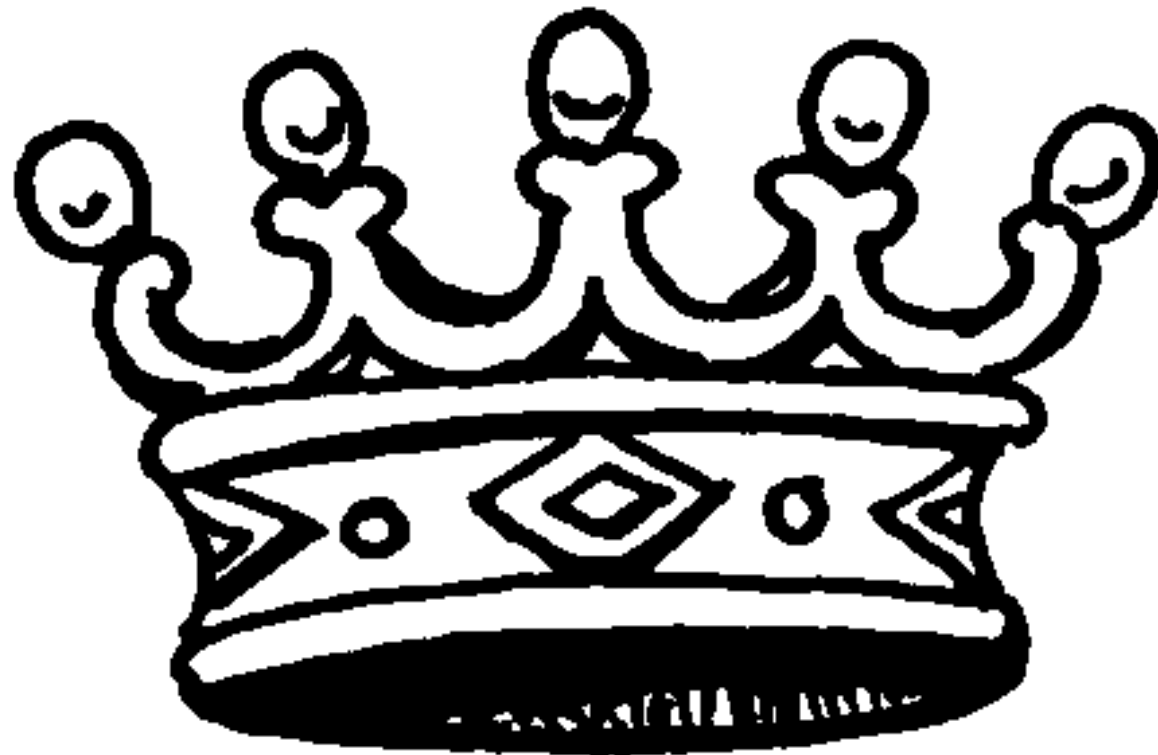
If this rule cannot be applied and there are still more than  $k^2$  points, then there is no solution  $\Rightarrow$  Kernel with at most  $k^2$  points.



# Kernelization

- ⑥ Kernelization can be thought of as a polynomial-time preprocessing before attacking the problem with whatever method we have. “It does no harm” to try kernelization.
- ⑥ Some kernelizations use lots of simple reduction rules and require a complicated analysis to bound the kernel size. . .
- ⑥ . . . while other kernelizations are based on surprising nice tricks (Next: Crown Reduction and the Sunflower Lemma).
- ⑥ Possibility to prove lower bounds (S. Saurabh’s lecture).

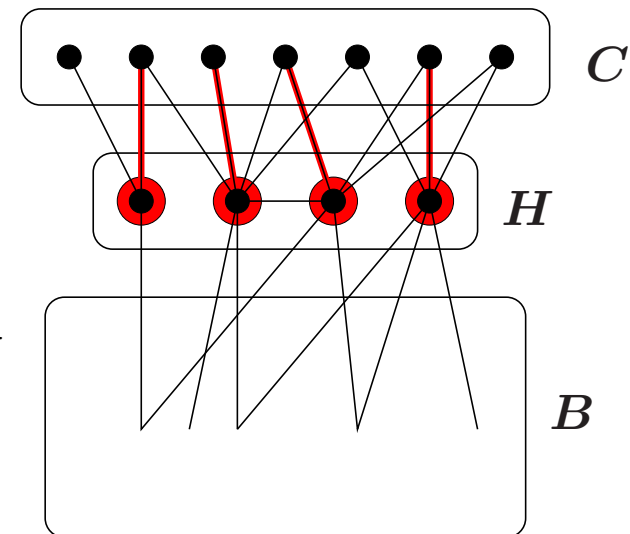
# *Crown Reduction*



# Crown Reduction

**Definition:** A **crown decomposition** is a partition  $C \cup H \cup B$  of the vertices such that

- ⑥  $C$  is an independent set,
- ⑥ there is no edge between  $C$  and  $B$ ,
- ⑥ there is a matching between  $C$  and  $H$  that covers  $H$ .



## Crown rule for VERTEX COVER:

The matching needs to be covered and we can assume that it is covered by  $H$  (makes no sense to use vertices of  $C$ )

$$\Rightarrow (G \setminus (H \cup C), k - |H|).$$

# Crown Reduction

Key lemma:

**Lemma:** Given a graph  $G$  without isolated vertices and an integer  $k$ , in polynomial time we can either

- ⑥ find a matching of size  $k + 1$ ,  $\Rightarrow$  **No solution!**
- ⑥ find a crown decomposition,  $\Rightarrow$  **Reduce!**
- ⑥ or conclude that the graph has at most  $3k$  vertices.  
 $\Rightarrow$   **$3k$  vertex kernel!**

This gives a  $3k$  vertex kernel for VERTEX COVER.

# Proof

**Lemma:** Given a graph  $G$  without isolated vertices and an integer  $k$ , in polynomial time we can either

- ⑥ find a matching of size  $k + 1$ ,
- ⑥ find a crown decomposition,
- ⑥ or conclude that the graph has at most  $3k$  vertices.

For the proof, we need the classical König's Theorem.

$\tau(G)$  : size of the minimum vertex cover

$\nu(G)$  : size of the maximum matching (independent set of edges)

**Theorem:** [Kőnig, 1931] If  $G$  is **bipartite**, then

$$\tau(G) = \nu(G)$$

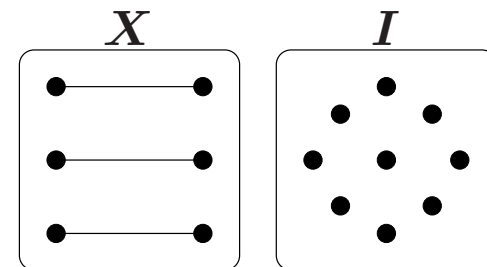
# Proof

**Lemma:** Given a graph  $G$  without isolated vertices and an integer  $k$ , in polynomial time we can either

- ⑥ find a matching of size  $k + 1$ ,
- ⑥ find a crown decomposition,
- ⑥ or conclude that the graph has at most  $3k$  vertices.

**Proof:** Find (greedily) a maximal matching; if its size is at least  $k + 1$ , then we are done. The rest of the graph is an independent set  $I$ .

Find a maximum matching/minimum vertex cover in the bipartite graph between  $X$  and  $I$ .



# Proof

**Lemma:** Given a graph  $G$  without isolated vertices and an integer  $k$ , in polynomial time we can either

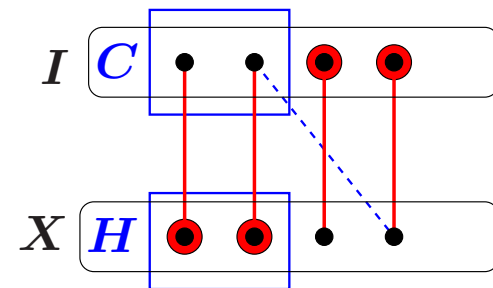
- ⑥ find a matching of size  $k + 1$ ,
- ⑥ find a crown decomposition,
- ⑥ or conclude that the graph has at most  $3k$  vertices.

**Proof:** Case 1: The minimum vertex cover contains at least one vertex of  $X$

$\Rightarrow$  There is a crown decomposition.

Case 2: The minimum vertex cover contains only vertices of  $I \Rightarrow$  It contains every vertex of  $I$

$\Rightarrow$  There are at most  $2k + k$  vertices.



# DUAL OF VERTEX COLORING

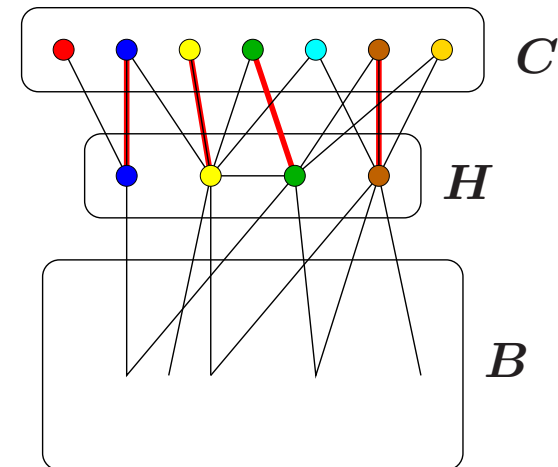
**Parameteric dual** of  $k$ -COLORING. Also known as SAVING  $k$  COLORS.

**Task:** Given a graph  $G$  and an integer  $k$ , find a vertex coloring with  $|V(G)| - k$  colors.

**Crown rule for DUAL OF VERTEX COLORING:**

Suppose there is a crown decomposition for the **complement graph**  $\overline{G}$ .

- ⑥  $C$  is a clique in  $G$ : each vertex needs a distinct color.
- ⑥ Because of the matching,  $H$  can be colored using only these  $|C|$  colors.
- ⑥ These colors cannot be used for  $B$ .
- ⑥  $(G \setminus (H \cup C), k - |H|)$





# *Crown Reduction* for DUAL OF VERTEX COLORING

Use the key lemma for the complement  $\overline{G}$  of  $G$ :

**Lemma:** Given a graph  $G$  without isolated vertices and an integer  $k$ , in polynomial time we can either

- ⑥ find a matching of size  $k + 1$ ,  $\Rightarrow$  YES: we can save  $k$  colors!
- ⑥ find a crown decomposition,  $\Rightarrow$  Reduce!
- ⑥ or conclude that the graph has at most  $3k$  vertices.  
 $\Rightarrow$   $3k$  vertex kernel!

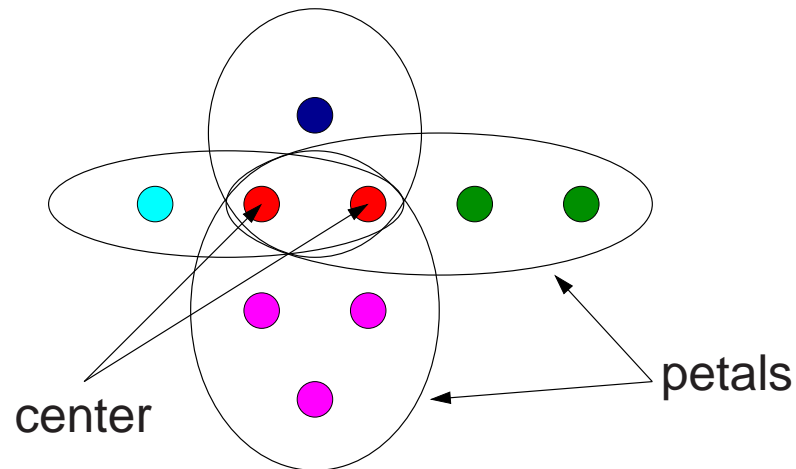
This gives a  $3k$  vertex kernel for DUAL OF VERTEX COLORING.

# ***Sunflower Lemma***



# Sunflower lemma

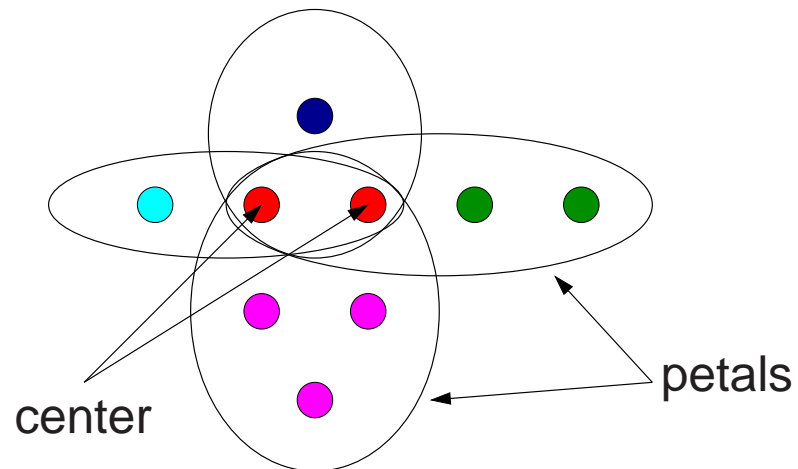
**Definition:** Sets  $S_1, S_2, \dots, S_k$  form a **sunflower** if the sets  $S_i \setminus (S_1 \cap S_2 \cap \dots \cap S_k)$  are disjoint.



**Lemma:** [Erdős and Rado, 1960] If the size of a set system is greater than  $(p - 1)^d \cdot d!$  and it contains only sets of size at most  $d$ , then the system contains a sunflower with  $p$  petals. Furthermore, in this case such a sunflower can be found in polynomial time.

# Sunflowers and $d$ -HITTING SET

$d$ -HITTING SET: Given a collection  $\mathcal{S}$  of sets of size at most  $d$  and an integer  $k$ , find a set  $S$  of  $k$  elements that intersects every set of  $\mathcal{S}$ .

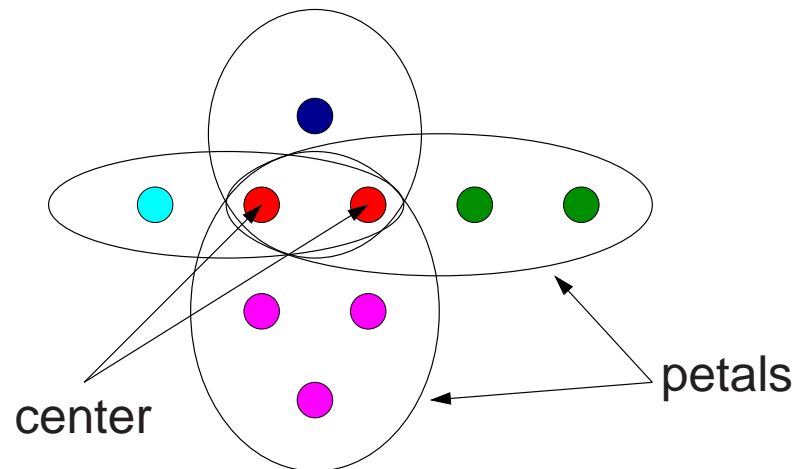


**Reduction Rule:** If  $k + 1$  sets form a sunflower, then remove these sets from  $\mathcal{S}$  and add the center  $C$  to  $\mathcal{S}$  ( $S$  does not hit one of the petals, thus it has to hit the center).

If the rule cannot be applied, then there are at most  $O(k^d)$  sets.

# Sunflowers and $d$ -HITTING SET

$d$ -HITTING SET: Given a collection  $\mathcal{S}$  of sets of size at most  $d$  and an integer  $k$ , find a set  $S$  of  $k$  elements that intersects every set of  $\mathcal{S}$ .



**Reduction Rule (variant):** Suppose more than  $k + 1$  sets form a sunflower.

- ⌚ If the sets are disjoint  $\Rightarrow$  No solution.
- ⌚ Otherwise, keep only  $k + 1$  of the sets.

If the rule cannot be applied, then there are at most  $O(k^d)$  sets.

# *Graph Minors*



Neil Robertson



Paul Seymour

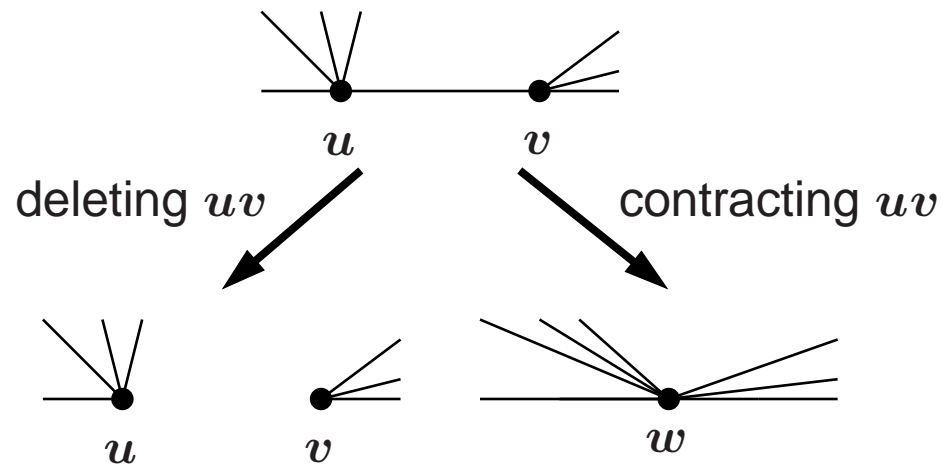
# Graph Minors

- ⑥ Some consequences of the Graph Minors Theorem give a quick way of showing that certain problems are FPT.
- ⑥ However, the function  $f(k)$  in the resulting FPT algorithms can be HUGE, completely impractical.
- ⑥ History: motivation for FPT.
- ⑥ Parts and ingredients of the theory are useful for algorithm design.
- ⑥ 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.



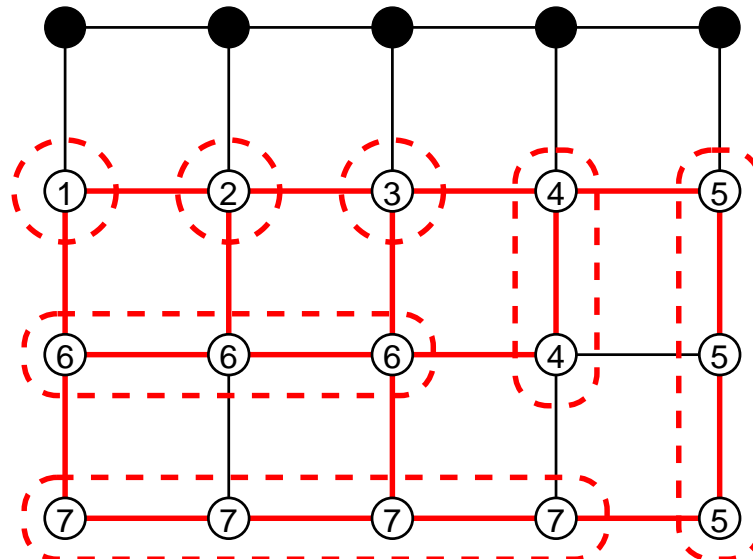
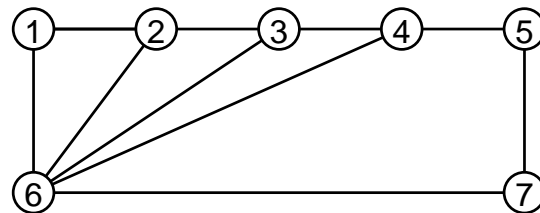
**Example:** A triangle is a minor of a graph  $G$  if and only if  $G$  has a cycle (i.e., it is not a forest).



# Graph minors

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

- ⑥  $\phi(u)$  and  $\phi(v)$  are disjoint if  $u \neq v$ , and
- ⑥ 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:**

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

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”.

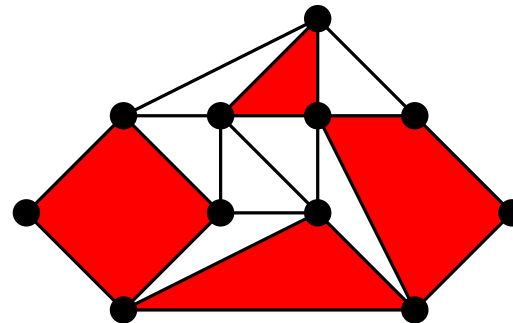
**Note:** The size of the obstruction set can be astronomical even for simple properties.

**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.

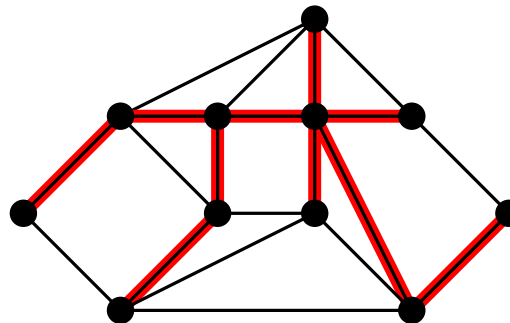


For every fixed  $k$ , there is a  $O(n^3)$  time algorithm for PLANAR FACE COVER.

**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:**

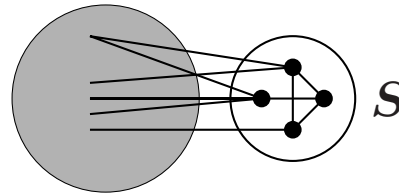
For every fixed  $k$ , the class  $\mathcal{G}_k$  of no-instances is minor closed.



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  $k$ .

$\Rightarrow$  Finding the smallest  $k$  such that a given graph is in  $\mathcal{G} + kv$  is FPT.

- ⑥ 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.
- ⑥ 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.
- ⑥ If  $\mathcal{G} =$  empty graphs  $\Rightarrow \mathcal{G} + kv =$  graphs with vertex cover number at most  $k \Rightarrow$  VERTEX COVER is FPT.

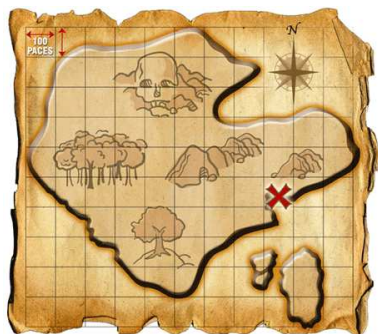
# Two types of problems



**We have to solve some problems.**

Typically **minimization** problems: VERTEX COVER, HITTING SET, DOMINATING SET, covering/stabbing problems, graph modification problems, . . .

**Bounded search trees, iterative compression**



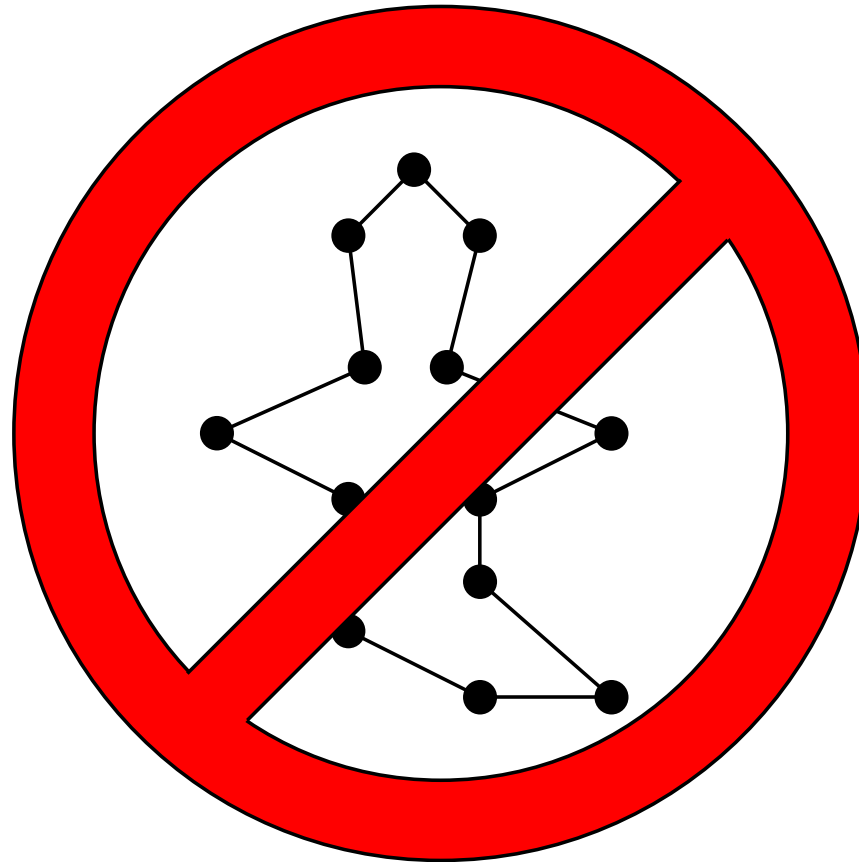
**We have to find something nice hidden somewhere.**

Typically **maximization** problems:  $k$ -PATH, DISJOINT TRIANGLES,  $k$ -LEAF SPANNING TREE, . . .

**Color coding, matroids**



# *Forbidden subgraphs*



# Forbidden subgraphs

**General problem class:** Given a graph  $G$  and an integer  $k$ , transform  $G$  with at most  $k$  modifications (add/remove vertices/edges) into a graph having property  $\mathcal{P}$ .

## Example:

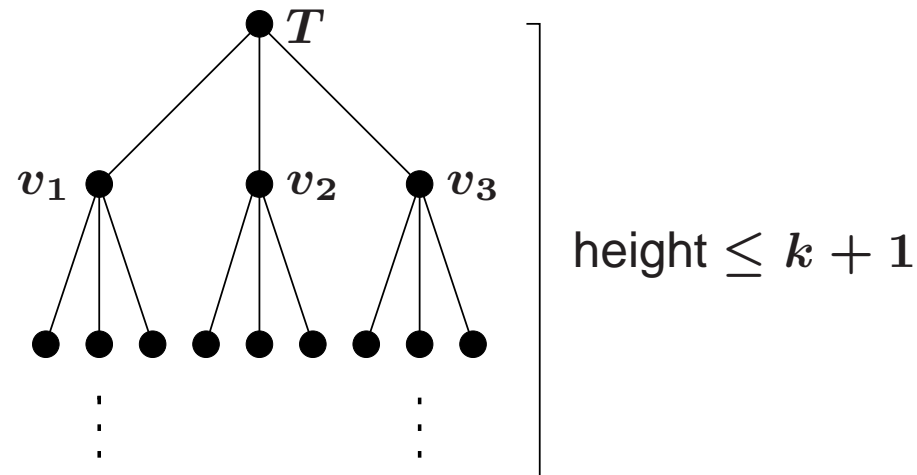
TRIANGLE DELETION: make the graph triangle-free by deleting at most  $k$  vertices.

Branching algorithm:

- ⑥ If the graph is triangle-free, then we are done.
- ⑥ If there is a triangle  $v_1v_2v_3$ , then at least one of  $v_1, v_2, v_3$  has to be deleted  $\Rightarrow$  We branch into 3 directions.

# TRIANGLE DELETION

Search tree:



The search tree has at most  $3^k$  leaves and the work to be done is polynomial at each step  $\Rightarrow O^*(3^k)$  time algorithm.

**Note:** If the answer is “NO”, then the search tree has **exactly**  $3^k$  leaves.

# Hereditary properties

**Definition:** A graph property  $\mathcal{P}$  is **hereditary** if for every  $G \in \mathcal{P}$  and induced subgraph  $G'$  of  $G$ , we have  $G' \in \mathcal{P}$  as well.

**Examples:** triangle-free, bipartite, interval graph, planar

**Observation:** Every hereditary property  $\mathcal{P}$  can be characterized by a (finite or infinite) set  $\mathcal{F}$  of forbidden induced subgraphs:

$$G \in \mathcal{P} \Leftrightarrow \forall H \in \mathcal{F}, H \not\subseteq_{\text{ind}} G$$

**Theorem:** If  $\mathcal{P}$  is hereditary and can be characterized by a **finite** set  $\mathcal{F}$  of forbidden induced subgraphs, then the graph modification problems corresponding to  $\mathcal{P}$  are FPT.

# Hereditary properties

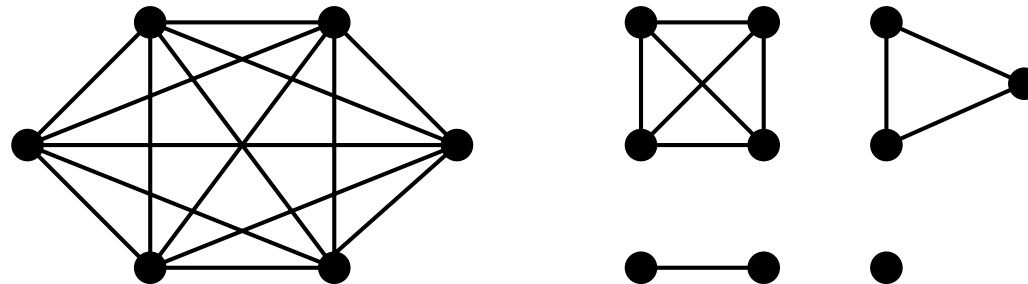
**Theorem:** If  $\mathcal{P}$  is hereditary and can be characterized by a **finite** set  $\mathcal{F}$  of forbidden induced subgraphs, then the graph modification problems corresponding to  $\mathcal{P}$  are FPT.

## Proof:

- ⑥ Suppose that every graph in  $\mathcal{F}$  has at most  $r$  vertices. Using brute force, we can find in time  $O(n^r)$  a forbidden subgraph (if exists).
- ⑥ If a forbidden subgraph exists, then we have to delete one of the at most  $r$  vertices or add/delete one of the at most  $\binom{r}{2}$  edges  $\Rightarrow$  Branching factor is a constant  $c$  depending on  $\mathcal{F}$ .
- ⑥ The search tree has at most  $c^k$  leaves and the work to be done at each node is  $O(n^r)$ .

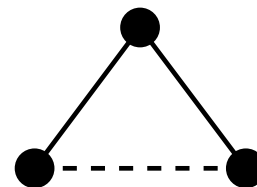
# CLUSTER EDITING

**Task:** Given a graph  $G$  and an integer  $k$ , add/remove at most  $k$  edges such that every component is a clique in the resulting graph.



Property  $\mathcal{P}$ : every component is a clique.

Forbidden induced subgraph:



$O^*(3^k)$  time algorithm.

# CHORDAL COMPLETION

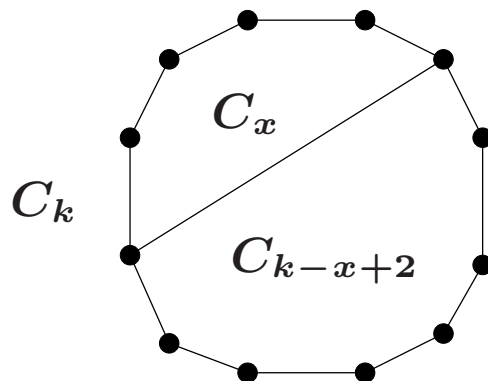
**Definition:** A graph is **chordal** if it does not contain an induced cycle of length greater than 3.

CHORDAL COMPLETION: Given a graph  $G$  and an integer  $k$ , add at most  $k$  edges to  $G$  to make it a chordal graph.

The forbidden induced subgraphs are the cycles of length greater 3  
 $\Rightarrow$  Not a finite set!

**Lemma:** At least  $k - 3$  edges are needed to make a  $k$ -cycle chordal.

**Proof:** By induction.  $k = 3$  is trivial.



$C_x$ :  $x - 3$  edges

$C_{k-x+2}$ :  $k - x - 1$  edges

$C_k$ :  $(x - 3) + (k - x - 1) + 1 =$   
 $k - 3$  edges

# CHORDAL COMPLETION

Algorithm:

- ⑥ Find an induced cycle  $C$  of length at least 4 (can be done in polynomial time).
- ⑥ If no such cycle exists  $\Rightarrow$  Done!
- ⑥ If  $C$  has more than  $k + 3$  vertices  $\Rightarrow$  No solution!
- ⑥ Otherwise, one of the

$$\binom{|C|}{2} - |C| \leq (k + 3)(k + 2)/2 - k = O(k^2)$$

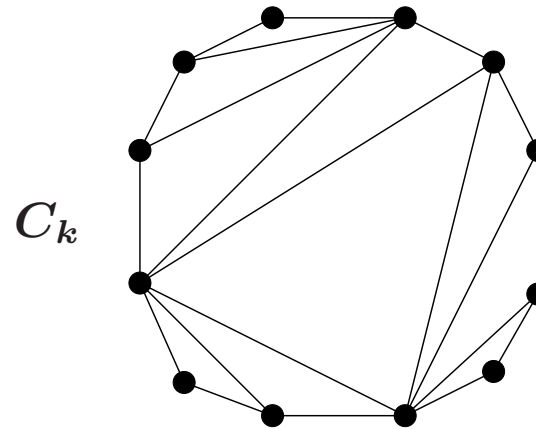
missing edges has to be added  $\Rightarrow$  Branch!

Size of the search tree is  $k^{O(k)}$ .



# CHORDAL COMPLETION – *more efficiently*

**Definition:** Triangulation of a cycle.



**Lemma:** Every chordal supergraph of a cycle  $C$  contains a triangulation of the cycle  $C$ .

**Lemma:** The number of ways a cycle of length  $k$  can be triangulated is exactly the  $(k - 2)$ th Catalan number

$$C_{k-2} = \frac{1}{k-1} \binom{2(k-2)}{k-2} \leq 4^{k-3}.$$

# CHORDAL COMPLETION – *more efficiently*

Algorithm:

- ⑥ Find an induced cycle  $C$  of length at least 4 (can be done in polynomial time).
- ⑥ If no such cycle exists  $\Rightarrow$  **Done!**
- ⑥ If  $C$  has more than  $k + 3$  vertices  $\Rightarrow$  **No solution!**
- ⑥ Otherwise, one of the  $\leq 4^{|C|-3}$  triangulations has to be in the solution  $\Rightarrow$  **Branch!**

**Claim:** Search tree has at most  $T_k = 4^k$  leaves.

**Proof:** By induction. Number of leaves is at most

$$T_k \leq 4^{|C|-3} \cdot T_{k-(|C|-3)} \leq 4^{|C|-3} \cdot 4^{k-(|C|-3)} = 4^k.$$

# *Iterative compression*



# *Iterative compression*

- ⑥ A surprising small, but very powerful trick.
- ⑥ Most useful for deletion problems: delete  $k$  things to achieve some property.
- ⑥ Demonstration: ODD CYCLE TRANSVERSAL aka BIPARTITE DELETION aka GRAPH BIPARTIZATION: Given a graph  $G$  and an integer  $k$ , delete  $k$  vertices to make the graph bipartite.
- ⑥ Forbidden induced subgraphs: odd cycles. There is no bound on the size of odd cycles.

# BIPARTITE DELETION

Solution based on iterative compression:

## ⑥ Step 1:

Solve the **annotated problem** for bipartite graphs:

Given a **bipartite** graph  $G$ , two sets  $B, W \subseteq V(G)$ , and an integer  $k$ , find a set  $S$  of at most  $k$  vertices such that  $G \setminus S$  has a 2-coloring where  $B \setminus S$  is black and  $W \setminus S$  is white.

## ⑥ Step 2:

Solve the **compression problem** for general graphs:

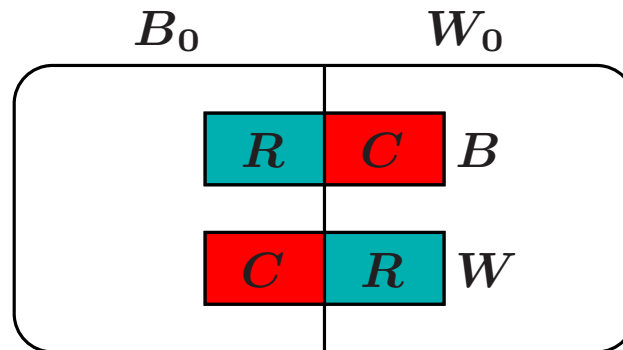
Given a graph  $G$ , an integer  $k$ , and **a set  $S'$  of  $k + 1$  vertices such that  $G \setminus S'$  is bipartite**, find a set  $S$  of  $k$  vertices such that  $G \setminus S$  is bipartite.

## ⑥ Step 3:

Apply the magic of iterative compression. . .

# Step 1: The annotated problem

Given a **bipartite** graph  $G$ , two sets  $B, W \subseteq V(G)$ , and an integer  $k$ , find a set  $S$  of at most  $k$  vertices such that  $G \setminus S$  has a 2-coloring where  $B \setminus S$  is black and  $W \setminus S$  is white.



Find an arbitrary 2-coloring  $(B_0, W_0)$  of  $G$ .

$C := (B_0 \cap W) \cup (W_0 \cap B)$  should change color, while

$R := (B_0 \cap B) \cup (W_0 \cap W)$  should remain the same color.

**Lemma:**  $G \setminus S$  has the required 2-coloring if and only if  $S$  separates  $C$  and  $R$ , i.e., no component of  $G \setminus S$  contains vertices from both  $C \setminus S$  and  $R \setminus S$ .

# Step 1: The annotated problem

**Lemma:**  $G \setminus S$  has the required 2-coloring if and only if  $S$  separates  $C$  and  $R$ , i.e., no component of  $G \setminus S$  contains vertices from both  $C \setminus S$  and  $R \setminus S$ .

**Proof:**

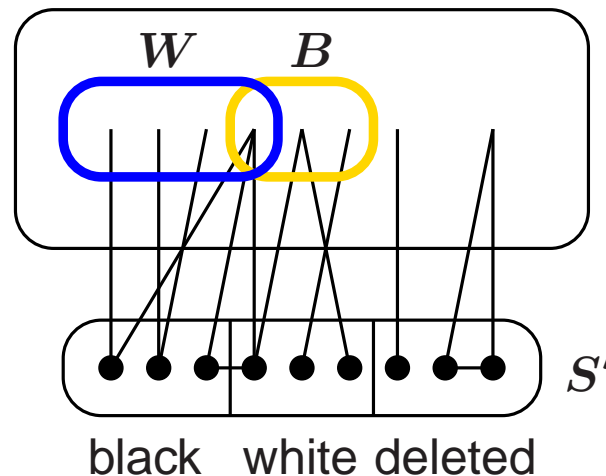
$\Rightarrow$  In a 2-coloring of  $G \setminus S$ , each vertex either remained the same color or changed color. Adjacent vertices do the same, thus every component either changed or remained.

$\Leftarrow$  Flip the coloring of those components of  $G \setminus S$  that contain vertices from  $C \setminus S$ . No vertex of  $R$  is flipped.

**Algorithm:** Using max-flow min-cut techniques, we can check if there is a set  $S$  that separates  $C$  and  $R$ . It can be done in time  $O(k|E(G)|)$  using  $k$  iterations of the Ford-Fulkerson algorithm.

## Step 2: The compression problem

Given a graph  $G$ , an integer  $k$ , and a set  $S'$  of  $k + 1$  vertices such that  $G \setminus S'$  is bipartite, find a set  $S$  of  $k$  vertices such that  $G \setminus S$  is bipartite.



Branch into  $3^{k+1}$  cases: each vertex of  $S'$  is either black, white, or deleted.

Trivial check: no edge between two black or two white vertices.

Neighbors of the black vertices in  $S'$  should be white and the neighbors of the white vertices in  $S'$  should be black.



## Step 3: Iterative compression

How do we get a solution of size  $k + 1$ ?

**We get it for free!**

Let  $V(G) = \{v_1, \dots, v_n\}$  and let  $G_i$  be the graph induced by  $\{v_1, \dots, v_i\}$ .

For every  $i$ , we find a set  $S_i$  of size  $k$  such that  $G_i \setminus S_i$  is bipartite.

- ⑥ For  $G_k$ , the set  $S_k = \{v_1, \dots, v_k\}$  is a trivial solution.
- ⑥ If  $S_{i-1}$  is known, then  $S_{i-1} \cup \{v_i\}$  is a set of size  $k + 1$  whose deletion makes  $G_i$  bipartite  $\Rightarrow$  We can use the compression algorithm to find a suitable  $S_i$  in time  $O(3^k \cdot k|E(G_i)|)$ .

## Step 3: Iterative Compression

Bipartite-Deletion( $G, k$ )

1.  $S_k = \{v_1, \dots, v_k\}$
2. for  $i := k + 1$  to  $n$
3. Invariant:  $G_{i-1} \setminus S_{i-1}$  is bipartite.
4. Call Compression( $G_i, S_{i-1} \cup \{v_i\}$ )
5. If the answer is “NO”  $\Rightarrow$  return “NO”
6. If the answer is a set  $X \Rightarrow S_i := X$
7. Return the set  $S_n$

**Running time:** the compression algorithm is called  $n$  times and everything else can be done in linear time

$\Rightarrow O(3^k \cdot k|V(G)| \cdot |E(G)|)$  time algorithm.

# ***Color coding***



# Color coding

- ⑥ Works best when we need to ensure that a small number of “things” are disjoint.
- ⑥ We demonstrate it on two problems:
  - △ Find an  $s$ - $t$  path of length exactly  $k$ .
  - △ Find  $k$  vertex-disjoint triangles in a graph.
- ⑥ Randomized algorithm, but can be derandomized using a standard technique.
- ⑥ Very robust technique, we can use it as an “opening step” when investigating a new problem.

# $k$ -PATH

**Task:** Given a graph  $G$ , an integer  $k$ , two vertices  $s, t$ , find a **simple**  $s$ - $t$  path with exactly  $k$  internal vertices.

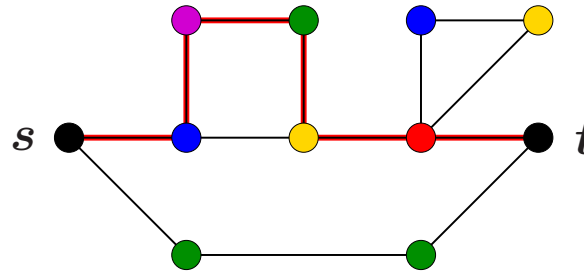
**Note:** Finding such a **walk** can be done easily in polynomial time.

**Note:** The problem is clearly NP-hard, as it contains the  $s$ - $t$  HAMILTONIAN PATH problem.

The  $k$ -PATH algorithm can be used to check if there is a cycle of length exactly  $k$  in the graph.

# $k$ -PATH

- ⑥ Assign colors from  $[k]$  to vertices  $V(G) \setminus \{s, t\}$  uniformly and independently at random.



- ⑥ Check if there is a **colorful**  $s$ - $t$  path: a path where each color appears exactly once on the internal vertices; output “YES” or “NO”.
  - △ If there is no  $s$ - $t$   $k$ -path: no such colorful path exists  $\Rightarrow$  “NO”.
  - △ If there is an  $s$ - $t$   $k$ -path: the probability that such a path is colorful is

$$\frac{k!}{k^k} > \frac{\left(\frac{k}{e}\right)^k}{k^k} = e^{-k},$$

thus the algorithm outputs “YES” with at least that probability.

# Error probability

- ⑥ If there is a  $k$ -path, the probability that the algorithm **does not** say “YES” after  $e^k$  repetitions is at most

$$(1 - e^{-k})^{e^k} < \left(e^{-e^{-k}}\right)^{e^k} = 1/e \approx 0.38$$

- ⑥ Repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant.
- ⑥ For example, by trying  $100 \cdot e^k$  random colorings, the probability of a wrong answer is at most  $1/e^{100}$ .

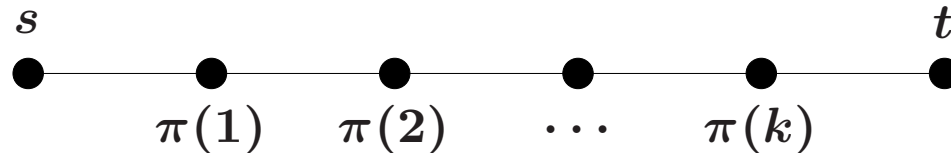
It remains to see how a colorful  $s$ - $t$  path can be found.

**Method 1:** Trying all permutations.

**Method 2:** Dynamic programming.

# Method 1: Trying all permutations

The colors encountered on a colorful  $s$ - $t$  path form a permutation  $\pi$  of  $\{1, 2, \dots, k\}$ :

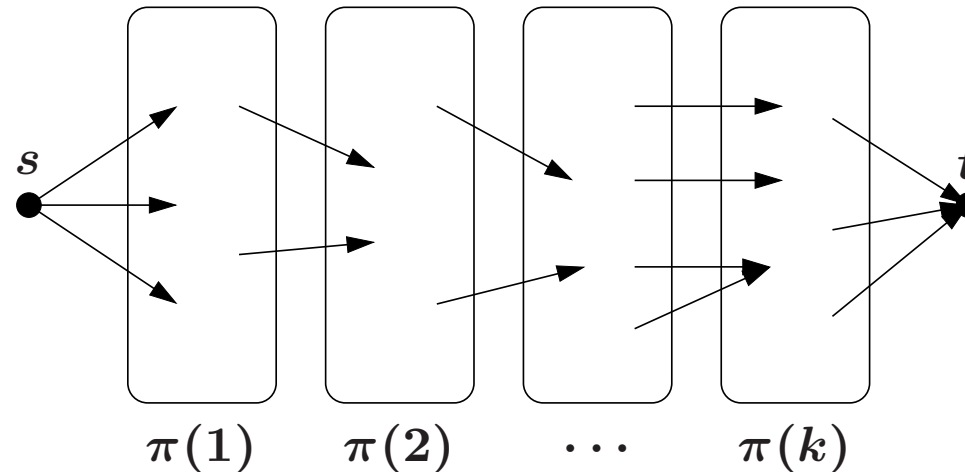


We try all possible  $k!$  permutations. For a fixed  $\pi$ , it is easy to check if there is a path with this order of colors.



# Method 1: Trying all permutations

We try all possible  $k!$  permutations. For a fixed  $\pi$ , it is easy to check if there is a path with this order of colors.



- ⑥ Edges connecting nonadjacent color classes are removed.
- ⑥ The remaining edges are directed.
- ⑥ All we need to check if there is a directed  $s-t$  path.
- ⑥ Running time is  $O(k! \cdot |E(G)|)$ .

# Method 2: Dynamic Programming

We introduce  $2^k \cdot |V(G)|$  Boolean variables:

$x(v, C) = \text{TRUE}$  for some  $v \in V(G)$  and  $C \subseteq [k]$



There is an  $s$ - $v$  path where each color in  $C$  appears exactly once and no other color appears.

Clearly,  $x(s, \emptyset) = \text{TRUE}$ . Recurrence for vertex  $v$  with color  $r$ :

$$x(v, C) = \bigvee_{uv \in E(G)} x(u, C \setminus \{r\})$$

If we know every  $x(v, C)$  with  $|C| = i$ , then we can determine every  $x(v, C)$  with  $|C| = i + 1 \Rightarrow$  All the values can be determined in time  $O(2^k \cdot |E(G)|)$ .

There is a colorful  $s$ - $t$  path  $\Leftrightarrow x(v, [k]) = \text{TRUE}$  for some neighbor of  $t$ .

# Derandomization

Using Method 2, we obtain a  $O^*((2e)^k)$  time algorithm with constant error probability. How to make it deterministic?

**Definition:** A family  $\mathcal{H}$  of functions  $[n] \rightarrow [k]$  is a  **$k$ -perfect** family of hash functions if for every  $S \subseteq [n]$  with  $|S| = k$ , there is a  $h \in \mathcal{H}$  such that  $h(x) \neq h(y)$  for any  $x, y \in S, x \neq y$ .

Instead of trying  $O(e^k)$  random colorings, we go through a  $k$ -perfect family  $\mathcal{H}$  of functions  $V(G) \rightarrow [k]$ . If there is a solution  $\Rightarrow$  The internal vertices  $S$  are colorful for at least one  $h \in \mathcal{H} \Rightarrow$  Algorithm outputs “YES”.

**Theorem:** There is a  $k$ -perfect family of functions  $[n] \rightarrow [k]$  having size  $2^{O(k)} \log n$ .

$\Rightarrow$  There is a **deterministic**  $2^{O(k)} \cdot n^{O(1)}$  time algorithm for the  $k$ -PATH problem.

# $k$ -DISJOINT TRIANGLES

**Task:** Given a graph  $G$  and an integer  $k$ , find  $k$  vertex disjoint triangles.

**Step 1:** Choose a random coloring  $V(G) \rightarrow [3k]$ .

**Step 2:** Check if there is a colorful solution, where the  $3k$  vertices of the  $k$  triangles use distinct colors.

- ⑥ **Method 1:** Try every permutation  $\pi$  of  $[3k]$  and check if there are triangles with colors  $(\pi(1), \pi(2), \pi(3)), (\pi(4), \pi(5), \pi(6)), \dots$
- ⑥ **Method 2:** Dynamic programming. For  $C \subseteq [3k]$  and  $|C| = 3i$ , let  $x(C) = \text{TRUE}$  if and only if there are  $|C|/3$  disjoint triangles using exactly the colors in  $C$ .

$$x(C) = \bigvee_{\{c_1, c_2, c_3\} \subseteq C} (x(C \setminus \{c_1, c_2, c_3\}) \wedge \exists \Delta \text{ with colors } c_1, c_2, c_3)$$

# $k$ -DISJOINT TRIANGLES

**Step 3:** Colorful solution exists with probability at least  $e^{-3k}$ , which is a lower bound on the probability of a correct answer.

**Running time:** constant error probability after  $e^{3k}$  repetitions  $\Rightarrow$  running time is  $O^*((2e)^{3k})$  (using Method 2).

**Derandomization:**  $3k$ -perfect family of functions instead of random coloring. Running time is  $2^{O(k)} \cdot n^{O(1)}$ .

# Color coding

We have seen that color coding can be used to find paths, cycles of length  $k$ , or a set of  $k$  disjoint triangles.

What other structures can be found efficiently with this technique?

The key is treewidth:

**Theorem:** Given two graphs  $H, G$ , it can be decided if  $H$  is a subgraph of  $G$  in time  $2^{O(|V(H)|)} \cdot |V(G)|^{O(w)}$ , where  $w$  is the treewidth of  $G$ .

Thus if  $H$  belongs to a class of graphs with bounded treewidth, then the subgraph problem is FPT.

# *Matroid Theory*



# ***Matroid Theory***

- ⑥ Matroids: a classical subject of combinatorial optimization.
- ⑥ Matroids lurk behind matching, flow, spanning tree, and some linear algebra problems.
- ⑥ A general FPT result that can be used to show that some concrete problems are FPT.



# Matroids

**Definition:** A set system  $\mathcal{M}$  over  $E$  is a **matroid** if

- (1)  $\emptyset \in \mathcal{M}$ .
- (2) If  $X \in \mathcal{M}$  and  $Y \subseteq X$ , then  $Y \in \mathcal{M}$ .
- (3) If  $X, Y \in \mathcal{M}$  and  $|X| > |Y|$ , then  $\exists e \in X \setminus Y$  such that  $Y \cup \{e\} \in \mathcal{M}$ .

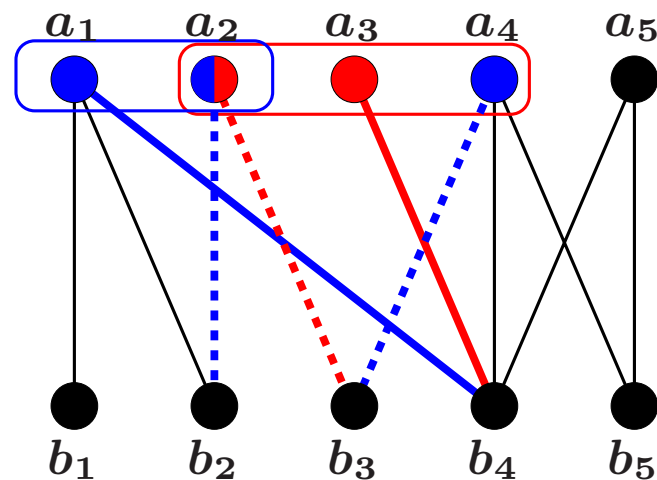
**Example:**  $\mathcal{M} = \{\emptyset, 1, 2, 3, 12, 13\}$  is a matroid.

**Example:**  $\mathcal{M} = \{\emptyset, 1, 2, 12, 3\}$  is not a matroid.

If  $X \in \mathcal{M}$ , then we say that  $X$  is **independent** in matroid  $\mathcal{M}$ .

# Transversal matroid

**Fact:** Let  $G(A, B; E)$  be a bipartite graph. Those subsets of  $A$  that can be covered by a matching form a matroid.



- (1) The empty set can be clearly covered.
- (2) If  $X$  can be covered, then every subset  $Y \subseteq X$  can be covered.
- (3) Suppose  $|X| > |Y|$  and they are covered by matchings  $M_X$  and  $M_Y$ , respectively. There is a component of  $M_X \cup M_Y$  containing more red edges than blue edges. We can augment  $M_Y$  along this path.

# Linear matroids

**Fact:** Let  $A$  be matrix and let  $E$  be the set of column vectors in  $A$ . The subsets  $E' \subseteq E$  that are linearly independent form a matroid.

**Proof:**

(1) and (2) are clear.

(3) If  $|X| > |Y|$  and both of them are linearly independent, then  $X$  spans a subspace with larger dimension than  $Y$ . Thus  $X$  contains a vector  $v$  not spanned by  $Y \Rightarrow Y \cup \{v\}$  is linearly independent.

**Example:**

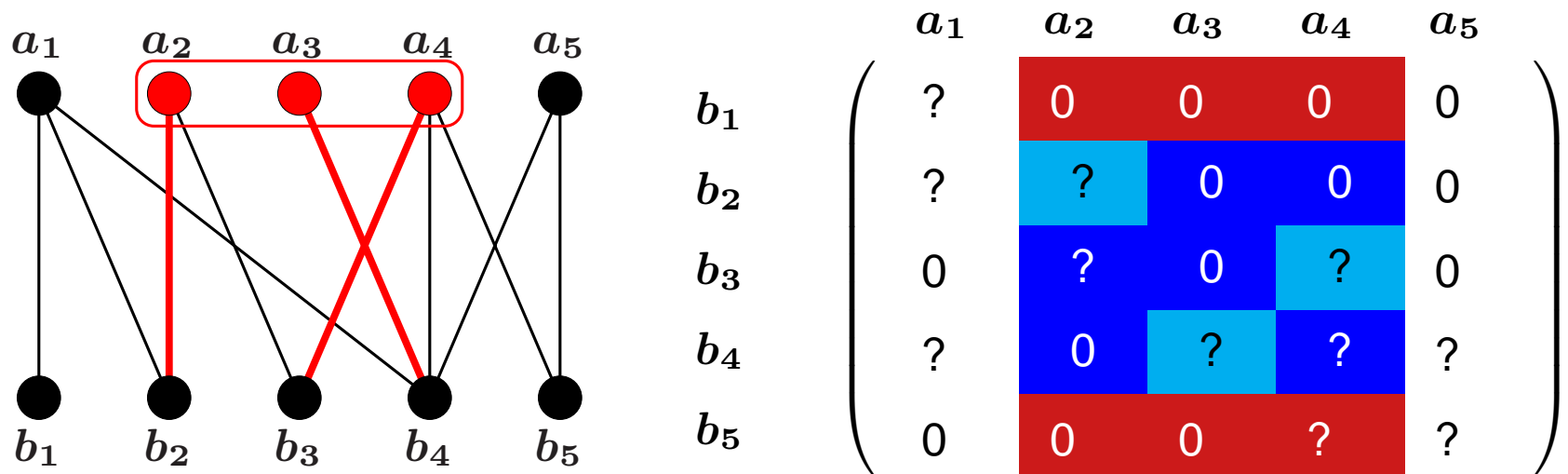
$$\begin{matrix} a & b & c & d \\ \begin{pmatrix} 1 & 0 & 2 & 3 \\ 0 & 1 & 4 & 6 \end{pmatrix} & \Rightarrow & \mathcal{M} = & \{\emptyset, a, b, c, d, ab, ac, ad, bc, bd\} \end{matrix}$$

# Representation

- ⑥ If  $\mathcal{M}$  is the matroid of the columns of a matrix  $A$ , then  $A$  is a **representation** of  $\mathcal{M}$ .
- ⑥ If  $A$  is a matrix over a field  $\mathbb{F}$ , then  $\mathcal{M}$  is **representable** over  $\mathbb{F}$ .
- ⑥ If  $\mathcal{M}$  is representable over some field  $\mathbb{F}$ , then  $\mathcal{M}$  is **linear**.
- ⑥ There are non-linear matroids (i.e., they cannot be represented over any field).

# Transversal matroids are linear

**Fact:** Let  $G(A, B; E)$  be a bipartite graph. Those subsets of  $A$  that can be covered by a matching form a linear matroid.



Construct the bipartite adjacency matrix: if  $a_i$  and  $b_j$  are neighbors, then the  $i$ -th element of row  $j$  is a random integer between 1 and  $N$ .

Elements can be matched  $\Rightarrow$  The determinant is nonzero with high probability (Schwartz-Zippel)

# FPT result

**Main result:** Let  $\mathcal{M}$  be a linear matroid over  $E$ , given by a representation  $A$ . Let  $\mathcal{S}$  be a collection of subsets of  $E$ , each of size at most  $\ell$ . It can be decided in randomized time  $f(k, \ell) \cdot n^{O(1)}$  whether  $\mathcal{M}$  has an independent set that is the union of  $k$  disjoint sets from  $\mathcal{S}$ .

**Immediate application:**  $k$ -DISJOINT TRIANGLES is (randomized) FPT (let  $\mathcal{S}$  be the set of all triangles in the graph).

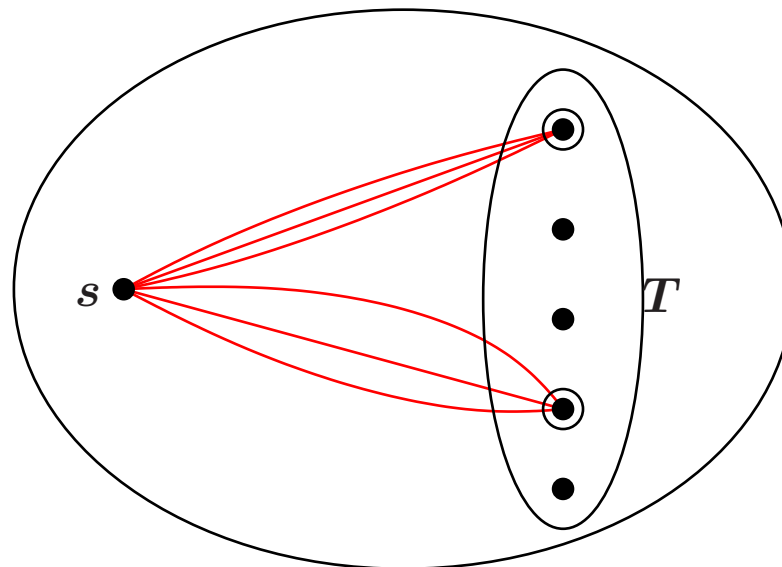
Two not so obvious applications:

- ⑥ RELIABLE TERMINALS
- ⑥ ASSIGNMENT WITH COUPLES

# RELIABLE TERMINALS

Let  $D$  be a directed graph with a source vertex  $s$  and a subset  $T$  of vertices.

**Task:** Select  $k$  terminals  $t_1, \dots, t_k \in T$ , and  $\ell$  paths from  $s$  to each  $t_i$  such that these  $k \cdot \ell$  paths are pairwise internally vertex disjoint.

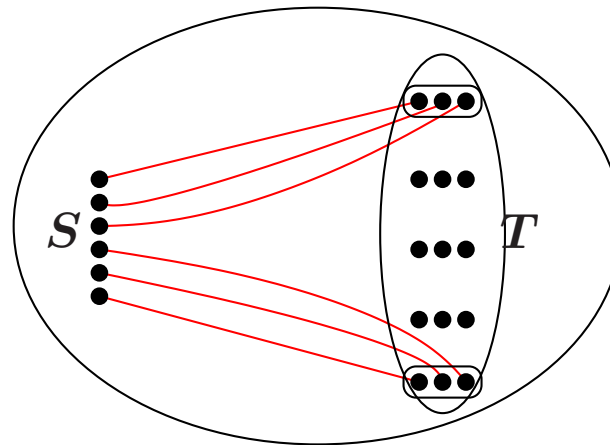


$$k = 2, \ell = 3$$

**Theorem:** The problem can be solved in randomized time  $f(k, \ell) \cdot n^{O(1)}$ .

# RELIABLE TERMINALS

A technical trick: replace each  $t \in T$  with  $\ell$  copies, and replace  $s$  with a set  $S$  of  $k \cdot \ell$  copies.



$$k = 2, \ell = 3$$

Now if a terminal  $t$  is selected, then we should connect the  $\ell$  copies of  $t$  with  $\ell$  different vertices of  $S$ .

**Fact:** [Perfect] Let  $D$  be a directed graph and  $S$  a subset of vertices. Those subsets  $X$  that can be reached from  $S$  by disjoint paths form a matroid.



# RELIABLE TERMINALS

**Fact:** [Perfect] Let  $D$  be a directed graph and  $S$  a subset of vertices. Those subsets  $X$  that can be reached from  $S$  by disjoint paths form a matroid.

The problem is equivalent to finding  $k$  blocks whose union is independent in this matroid  $\Rightarrow$  We can solve it in randomized time  $f(k, \ell) \cdot n^{O(1)}$ .

The matroid is actually a transversal matroid of an appropriately defined bipartite graph, hence it is linear and we can construct a representation for it.

# ASSIGNMENT WITH COUPLES

**Task:** Assign people to jobs (bipartite matching).

However, the set of people includes couples and the members of a couple cannot be assigned independently (say, they want to be in the same town).

**Task:** Given

- ⑥ a set of singles and a list of suitable jobs for each single,
  - ⑥ a set of couples and a list of suitable pairs of jobs for each couple,
- assign a job to each single and a pair of jobs to each couple.

**Theorem:** ASSIGNMENT WITH COUPLES is randomized FPT parameterized by the number  $k$  of couples.

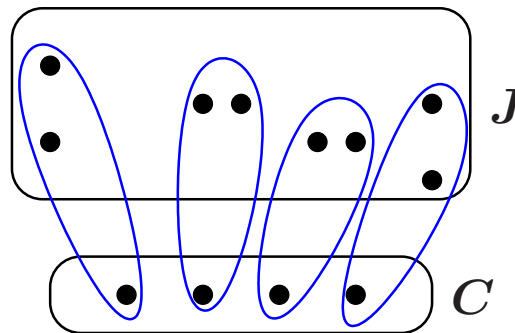
# ASSIGNMENT WITH COUPLES

$J$ : jobs,  $S$ : singles,  $C$ : couples

Let  $X \subseteq J$  be in  $\mathcal{M}$  if and only if  $S$  has a matching with  $J \setminus X$ .

**Lemma:**  $\mathcal{M}$  is matroid.

Let  $\mathcal{M}'$  be the matroid over  $J \cup C$  such that  $X \in \mathcal{M}' \Leftrightarrow X \cap J \in \mathcal{M}$ .



For each couple  $c \in C$  and suitable pair  $\{j_1, j_2\}$ , add triple  $\{c, j_1, j_2\}$  to  $\mathcal{S}$ .

The  $k$  couples and all the singles can be assigned a job



There are  $k$  disjoint triples in  $\mathcal{S}$  whose union is independent in  $\mathcal{M}'$

# *Cut problems*

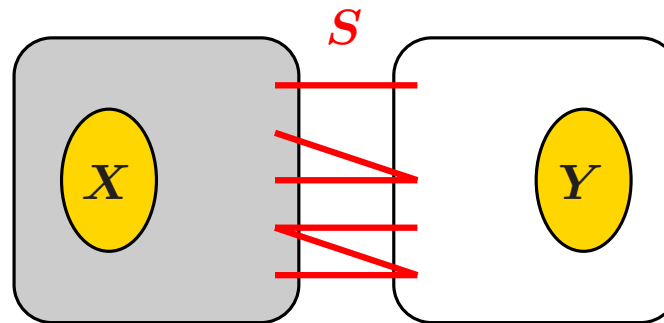


# MULTIWAY CUT

**Task:** Given a graph  $G$ , a set  $T$  of vertices, and an integer  $k$ , find a set  $S$  of at most  $k$  edges that separates  $T$  (each component of  $G \setminus S$  contains at most one vertex of  $T$ ).

Polynomial for  $|T| = 2$ , but NP-hard for  $|T| = 3$ .

**Theorem:** MULTIWAY CUT is FPT parameterized by  $k$ .



$\delta(R)$ : set of edges leaving  $R$

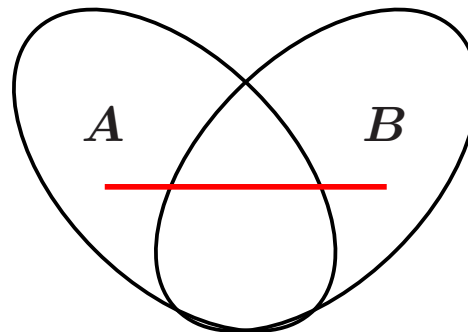
$\lambda(X, Y)$ : minimum number of edges in an  $(X, Y)$ -separator

# Submodularity

**Fact:** The function  $\delta$  is **submodular**: for arbitrary sets  $A, B$ ,

$$|\delta(A)| + |\delta(B)| \geq |\delta(A \cap B)| + |\delta(A \cup B)|$$

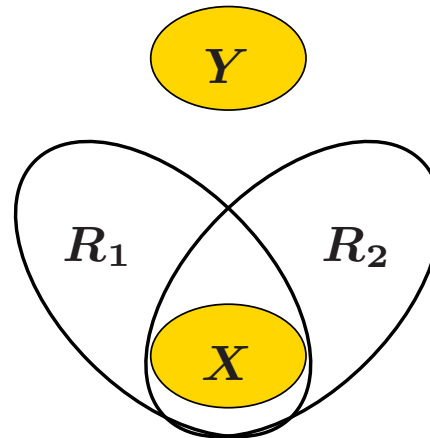
**Proof:** Determine separately the contribution of the different types of edges.



# Submodularity

**Consequence:** There is a unique maximal  $R_{\max} \supseteq X$  such that  $\delta(R_{\max})$  is an  $(X, Y)$ -separator of size  $\lambda(X, Y)$ .

**Proof:** Let  $R_1, R_2 \supseteq X$  be two sets such that  $\delta(R_1), \delta(R_2)$  are  $(X, Y)$ -separators of size  $\lambda := \lambda(X, Y)$ .

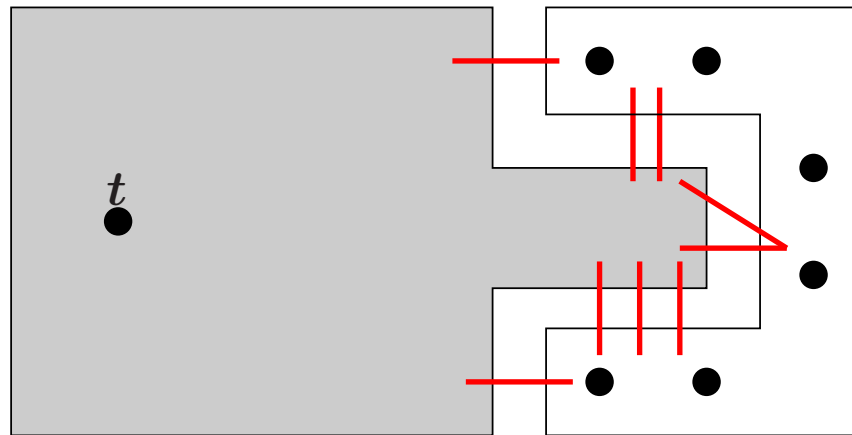


$$\begin{aligned} |\delta(R_1)| + |\delta(R_2)| &\geq |\delta(R_1 \cap R_2)| + |\delta(R_1 \cup R_2)| \\ \lambda + \lambda &\geq \lambda + |\delta(R_1 \cup R_2)| \\ \Rightarrow |\delta(R_1 \cup R_2)| &\leq \lambda \end{aligned}$$

**Note:** Analogous result holds for a unique minimal  $R_{\min}$ .

# MULTIWAY CUT

**Intuition:** Consider a  $t \in T$ . A subset of the solution separates  $t$  and  $T \setminus \{t\}$ .



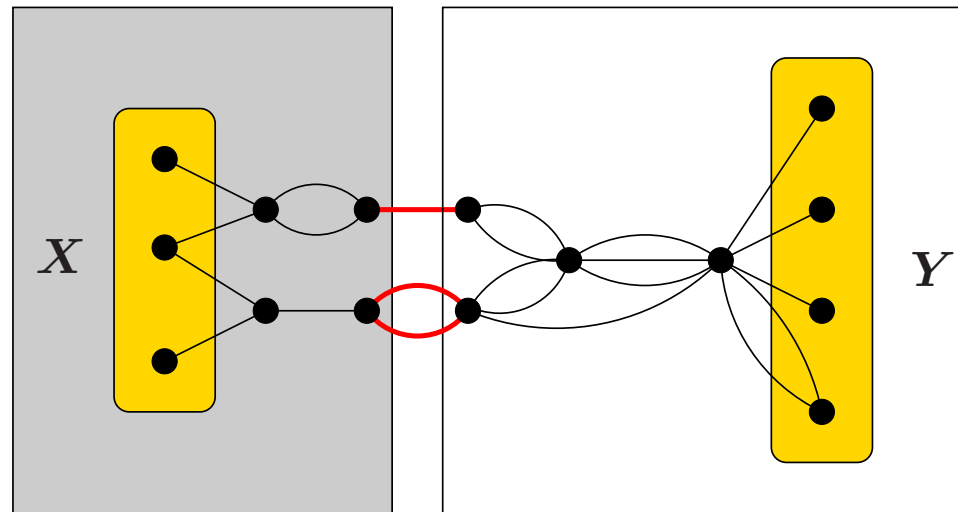
There are many such separators.

But a separator farther from  $t$  and closer to  $T \setminus \{t\}$  seems to be more useful.



# Important separators

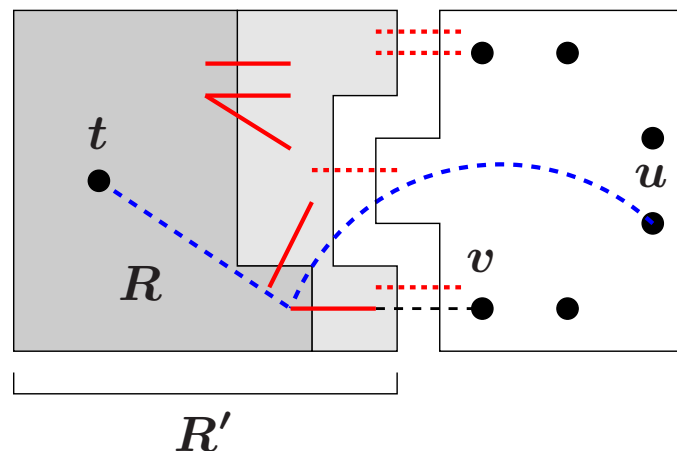
**Definition:** An  $(X, Y)$ -separator  $\delta(R)$  ( $R \supseteq X$ ) is **important** if there is no  $(X, Y)$ -separator  $\delta(R')$  with  $R \subset R'$  and  $|\delta(R')| \leq |\delta(R)|$ .



# Important separators

**Lemma:** Let  $t \in T$ . The MULTIWAY CUT problem has a solution  $S$  such that  $S$  contains an important  $(t, T \setminus \{t\})$ -separator.

**Proof:** Let  $R$  be the vertices reachable from  $t$  in  $G \setminus S$ .



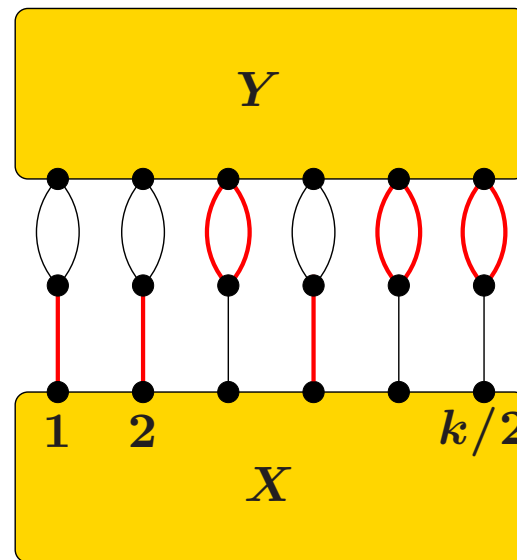
If  $\delta(R)$  is not important, then there is an important separator  $\delta(R')$  that dominates it. Replace  $S$  with  $S' := (S \setminus \delta(R)) \cup \delta(R')$  ( $|S'| \leq |S|$ ).

A  $u$ - $v$  path in  $G \setminus S'$  implies a  $u$ - $t$  path, a contradiction.

# Important separators

**Lemma:** There are at most  $4^k$  important  $(X, Y)$ -separators of size at most  $k$ .

**Example:**



There are exactly  $2^{k/2}$  important  $(X, Y)$ -separators of size at most  $k$  in this graph.

# Important separators

**Lemma:** There are at most  $4^k$  important  $(X, Y)$ -separators of size at most  $k$ .

**Proof:** First we show that  $R_{\max} \subseteq R$  for every important separator  $\delta(R)$ .

$$|\delta(R_{\max})| + |\delta(R)| \geq |\delta(R_{\max} \cap R)| + |\delta(R_{\max} \cup R)|$$

$$\lambda \geq \lambda$$

↓

$$|\delta(R_{\max} \cup R)| \leq |\delta(R)|$$

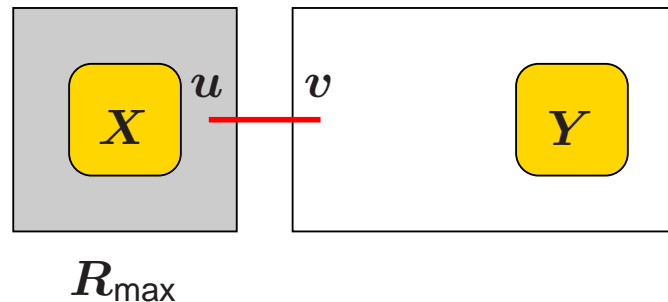
↓

If  $R \neq R_{\max} \cup R$ , then  $\delta(R)$  is not important.

Thus the important  $(X, Y)$ - and  $(R_{\max}, Y)$ -separators are the same.

# Important separators

**Lemma:** There are at most  $4^k$  important  $(X, Y)$ -separators of size at most  $k$ .



The edge  $uv$  leaving  $R_{\max}$  is either in the separator or not.

**Branch 1:** Edge  $uv$  is in the separator. Delete  $uv$  and set  $k := k - 1$ .

$\Rightarrow k$  decreases by one,  $\lambda$  decreases by at most 1.

**Branch 2:** Edge  $uv$  is not in the separator. Set  $X := R_{\max} \cup \{v\}$ .

$\Rightarrow k$  remains the same,  $\lambda$  increases by 1.

The measure  $2k - \lambda$  decreases in each step.

$\Rightarrow$  Height of the search tree  $\leq 2k \Rightarrow \leq 2^{2k}$  important separators.

# Algorithm for MULTIWAY CUT

1. If every vertex of  $T$  is in a different component, then we are done.
2. Let  $t \in T$  be a vertex with that is not separated from every  $T \setminus \{t\}$ .
3. Branch on a choice of an important  $(\{t\}, T \setminus \{t\})$  separator  $S$  of size at most  $k$ .
4. Set  $G := G \setminus S$  and  $k := k - |S|$ .
5. Go to step 1.

Size of the search tree:

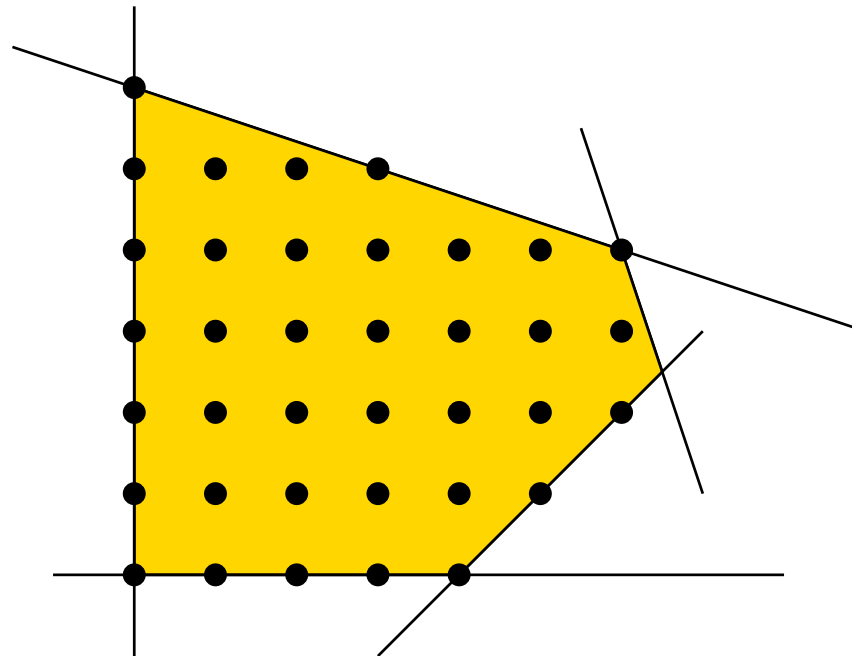
- ⑥ When searching for the important separator,  $2k - \lambda$  decreases at each branching.
- ⑥ When choosing the next  $t$ ,  $\lambda$  changes from 0 to positive, thus  $2k - \lambda$  does not increase.

Size of the search tree is at most  $2^{2k}$ .

# Other separation problems

- ⑥ Some other variants:
  - △  $|T|$  as a parameter
  - △ MULTITERMINAL CUT: pairs  $(s_1, t_1), \dots, (s_\ell, t_\ell)$  have to be separated.
  - △ Directed graphs
  - △ Planar graphs
- ⑥ Useful for deletion-type problems such as DIRECTED FEEDBACK VERTEX SET (via iterative compression).
- ⑥ Important separators: is it relevant for a given problem?

# *Integer Linear Programming*





# Integer Linear Programming

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

$$\max c_1x_1 + c_2x_2 + c_3x_3$$

s.t.

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

$$2x_1 - x_3 \leq 0$$

$$3x_2 + 10x_3 \leq 10$$

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

**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  $s_1, \dots, s_k$  of length  $L$  over alphabet  $\Sigma$ , 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
$s_3$	CDDBACCB
$s_4$	DDABACCB
$s_5$	ACDBDDCBC
<hr/>	
	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.

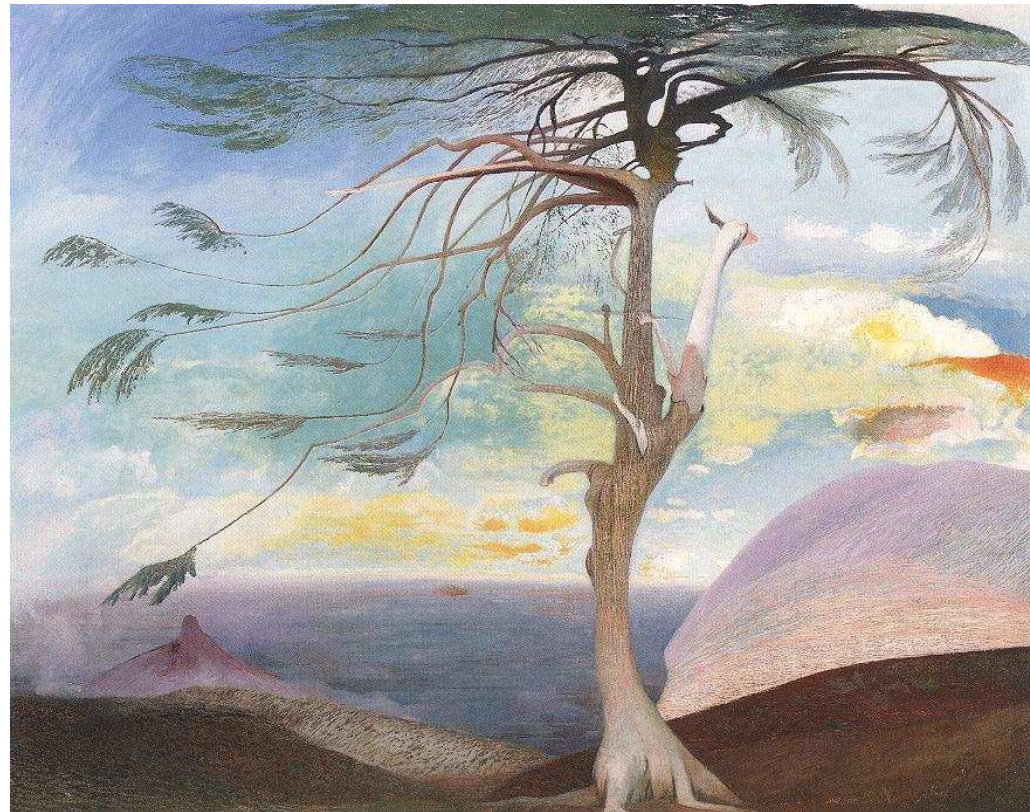
**Describing a solution:** If  $C$  is a class of  $\mathcal{P}$ , let  $x_{\mathcal{P},C}$  be the number of type  $\mathcal{P}$  columns where the solution agrees with class  $C$ .

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

$$\begin{aligned} \sum_{C \in \mathcal{P}} x_{\mathcal{P},C} &\leq c_{\mathcal{P}} && \forall \text{partition } \mathcal{P} \\ \sum_{i \notin C, C \in \mathcal{P}} x_{\mathcal{P},C} &\leq d && \forall 1 \leq i \leq k \\ x_{\mathcal{P},C} &\geq 0 && \forall \mathcal{P}, C \end{aligned}$$

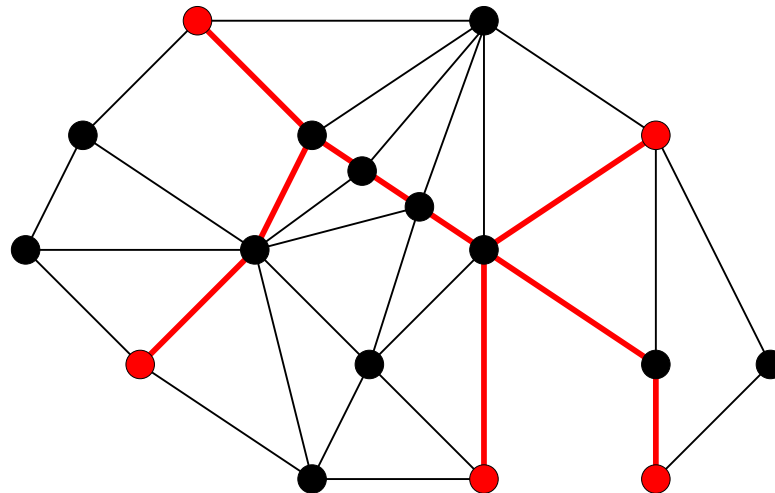
Number of variables is  $\leq B(k) \cdot k$ , where  $B(k)$  is the no. of partitions of  $[k]$   
 $\Rightarrow$  The ILP algorithm solves the problem in time  $f(k) \cdot n^{O(1)}$ .

# 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:**

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

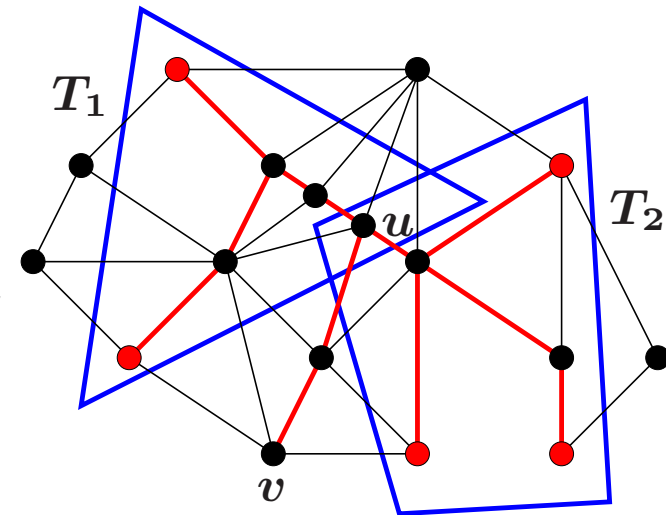


# STEINER TREE

Recurrence relation:

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

- ⑥  $\leq$ : 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  $v$ .

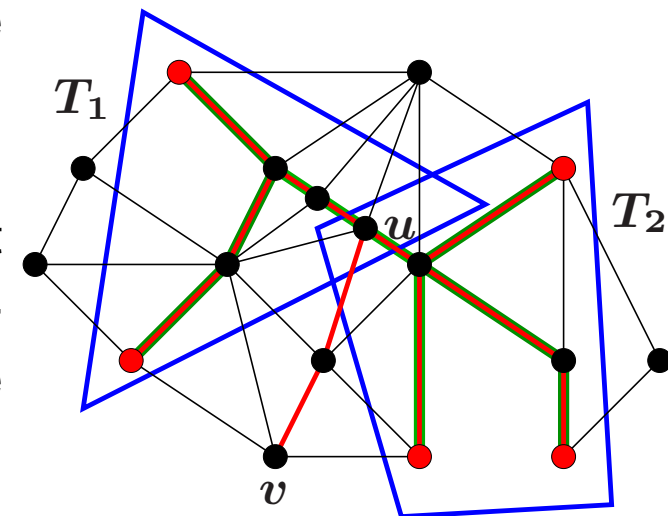


# STEINER TREE

Recurrence relation:

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

- ⑥  $\geq$ : 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.

# Conclusions

- ⑥ Many nice techniques invented so far — and probably many more to come.
- ⑥ A single technique might provide the key for several problems.
- ⑥ How to find new techniques? By attacking the open problems!
- ⑥ Needed: flexible, highly expressive problems. Solve other problems by reduction to these problems.
  - △ Courcelle's Theorem
  - △ The matroid result
  - △ 2SAT DELETION: given a 2SAT formula and an integer  $k$ , delete  $k$  clauses to make it satisfiable
  - △ Constraint Satisfaction Problems