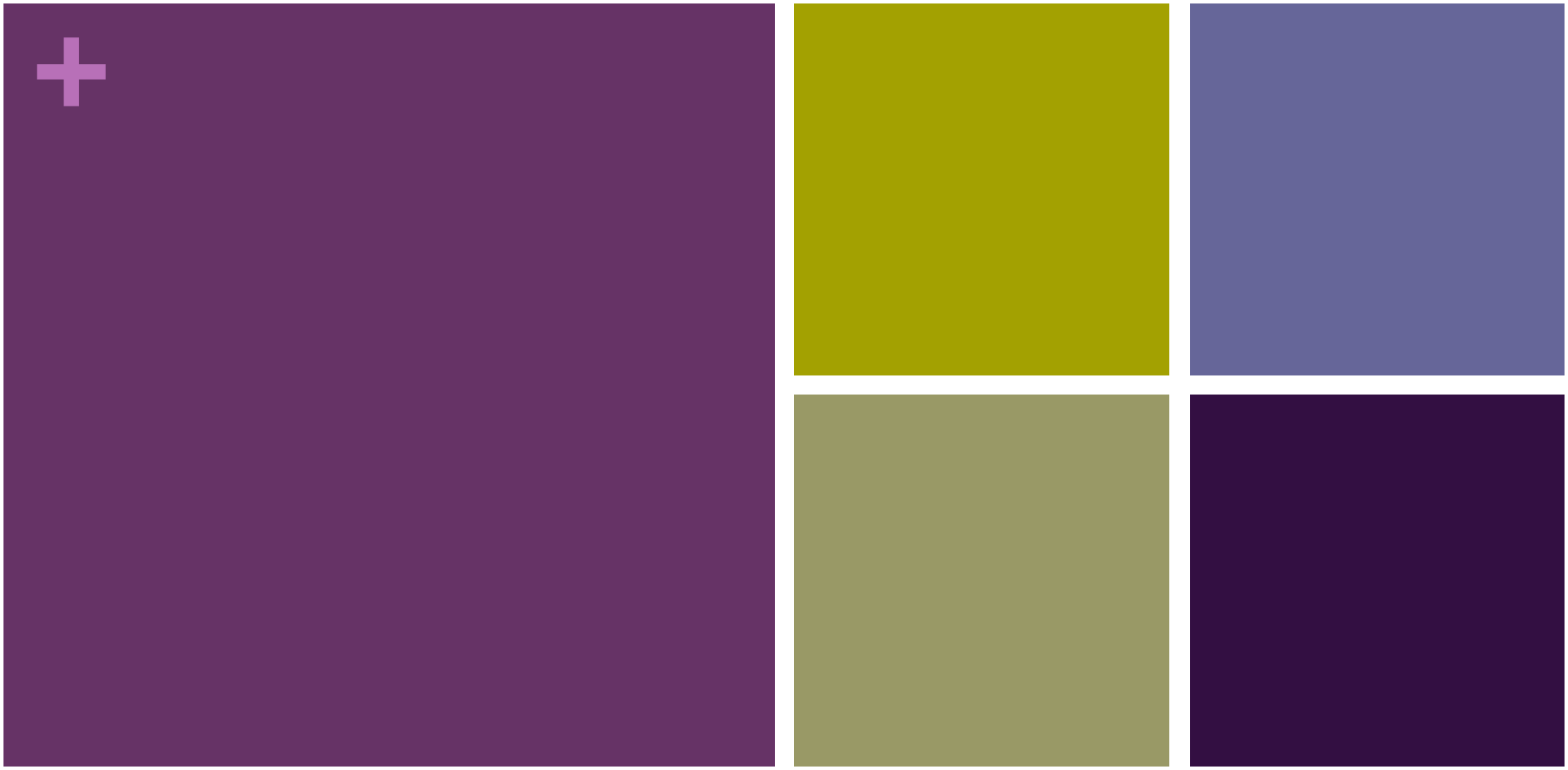


Fixed parameter algorithms

Introduction

Slides based on original material by:
Rolf niedermeier and mike fellows

Map of the next 45 minutes

- By the end of today you will be able to:
  - Define what is an FPT algorithm
  - Explain how to get a k^2 kernel for vertex cover
  - Explain how to get a k^2 kernel for max-satisfiability
  - Understand what is kernelization and what is its connection with FPT

The "classical" P vs NP framework is one-dimensional

n = input size

poly(n) vs $2^{poly(n)}$

"good" P

"bad" NP, etc.

positive toolkit of how to design P-time algorithms

negative toolkit of NP-hardness, etc.

Unfortunately, almost everything turns out to be NP-hard.

The parameterized framework is two-dimensional

n = input size

k = a relevant secondary measure

$f(k)n^c$ vs $n^{g(k)}$

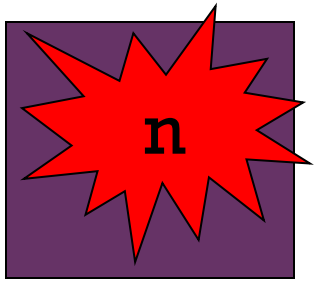
"good" "bad"

FPT Not discussed in this lecture

Framework in pictures

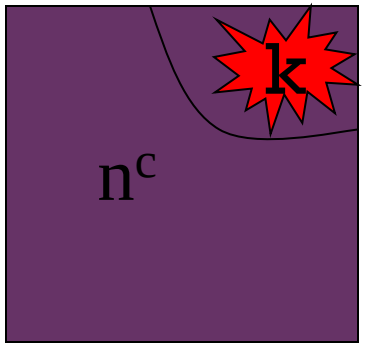
The *classical* P vs NP framework

Intrinsic Combinatorial explosion:
Most problems are NP-hard or worse.



The *parameterized* framework

FPT



Try to confine the explosion to the parameter.

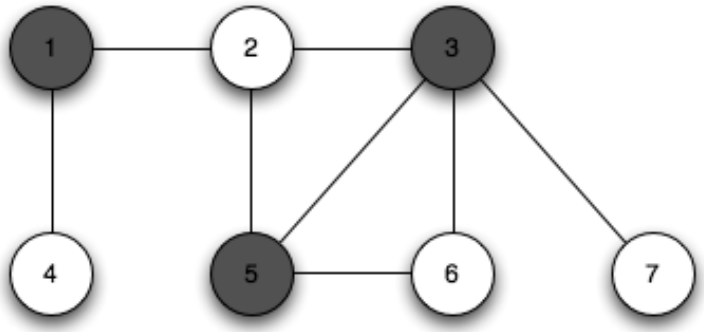
Vertex cover

- An NP-complete problem
- Easily parameterized (by the size of the set)

Vertex-cover(G,k)

Input: An undirected graph G = (V, E) and a nonnegative integer k.

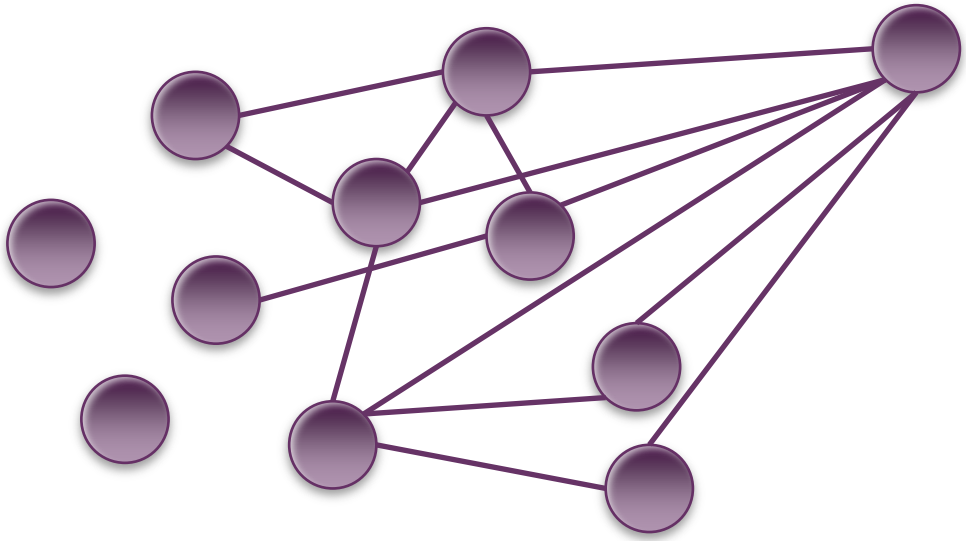
Task: Find a subset of vertices C ⊆ V with k or fewer vertices such that each edge in E has at least one of its endpoints in C.



Any other parameters?

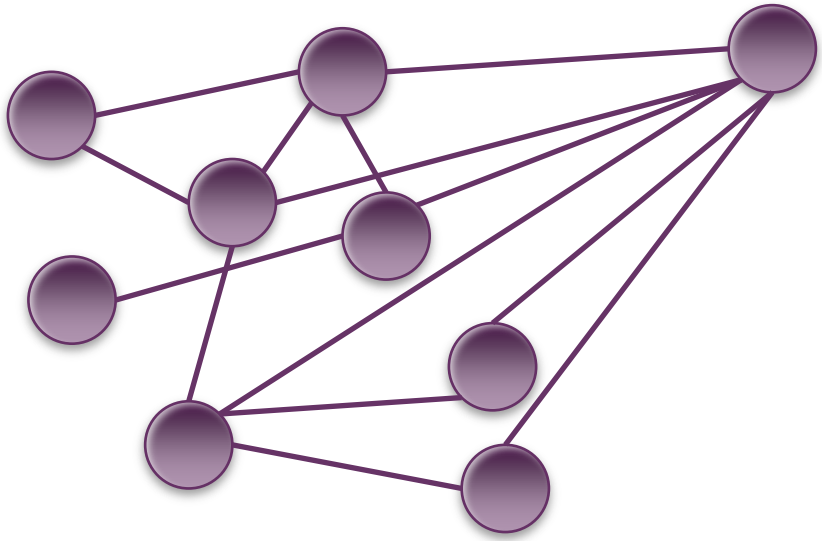
Vertex cover: Observations

- First, all vertices without edges connected are discarded.



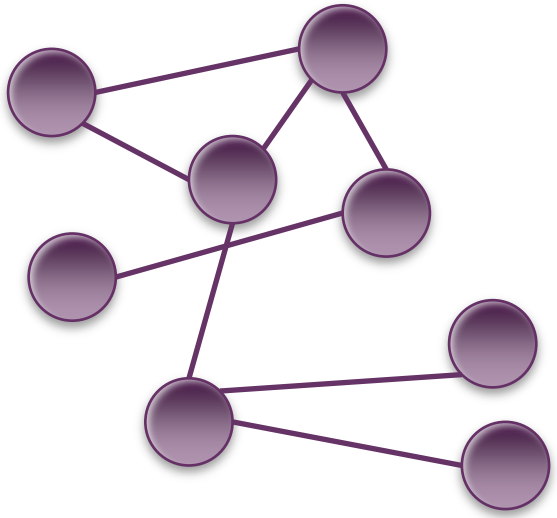
Vertex cover: Observation 1

- If v has degree at least $k+1$, then v belongs to each vertex cover in G of size at most k.
  - If v is not in the vertex cover, then all its neighbors are in the vertex cover.



Vertex cover: Observation 2

- (After applying observation 1) We are left with vertices with degree at most k
- Every edge needs to be covered, but every vertex has at most k neighbors.
  - Thus, if we have more than k^2 edges, no Vertex cover is possible



We are left with

- At most k^2 edges and at most $2k^2$ vertices are left (otherwise we can safely reject the instance)

Iterating on the vertices in the kernel we can achieve a running time of :

$$O^*(2^{2k^2})$$

- Exponential explosion ? Yes, but only w.r.t k and not n.

Definitions

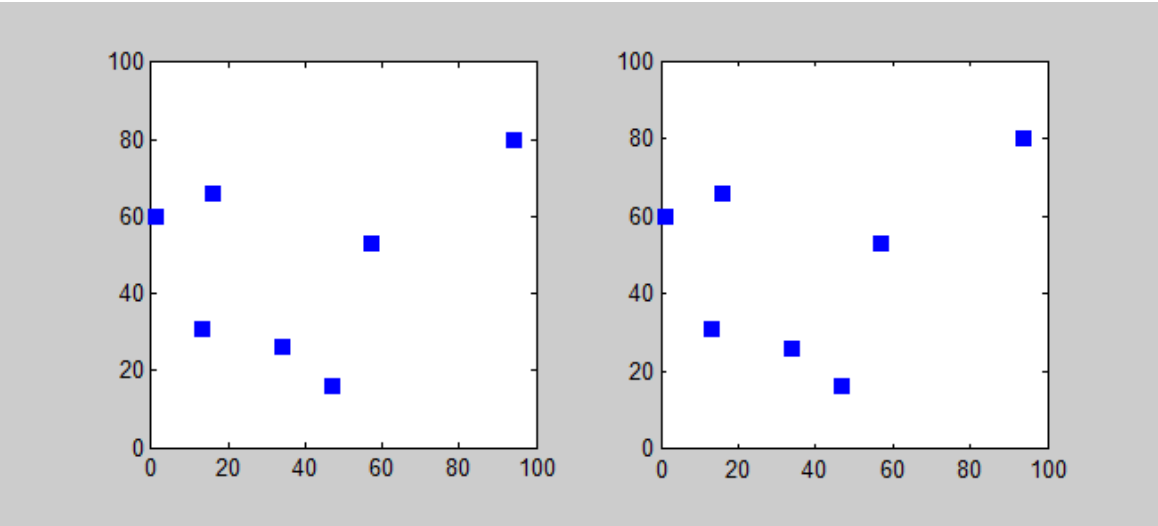
- A *parameterized* problem is a language

$L \subseteq \Sigma^* \times \Sigma^*$, where Σ is a finite alphabet. The second component is called the *parameter* of the problem.

- A parameterized problem is *fixed-parameter tractable* if it can be determined in $f(k) \cdot |x|^{O(1)}$ time whether $(x, k) \in L$, where f is a computable function only depending on k. The corresponding complexity class is called *FPT*.

TSP is also FPT …

- for k = number of cities by "try all permutations" implies a - k! n algorithm
- A parameterized problem can be "trivially FPT"
- It is still interesting to look for "better FPT" such as $2^k n$



Kernelization

- Preprocessing rules reduce starting instance to one of size f(k)
  - Should work in polynomial time
- Then use any algorithm to solve problem on kernel
- Time will be:

  p(n) + g(f(k))

Formal definition of kernelization

- Let L be a parameterized problem, that is, L consists of (I, k), where I is the problem instance and k is the parameter. *Reduction to a problem kernel* then means to replace instance (I, k) by a "reduced" instance (I', k') (called *problem kernel*) such that:

  1. $k' \leq k$, $|I'| \leq g(k)$ for **some** function g only depending on k
  2. $(I,k) \in L$ iff $(I',k') \in L$
  3. The reduction from (I, k) to (I', k') has to be **computable in polynomial time**.

Kernelization for Vertex Cover

H = G; (S = ø ;)

While there is a vertex v in H of degree at least

k+1 do:

Remove v and its incident edges from H

k = k – 1; (S = S + v ;)

If k < 0 then return false

If H has at least k^2+1 edges, then return false

Remove vertices of degree 0

Solve vertex cover on (H,k) with some algorithm

CNF, a reminder

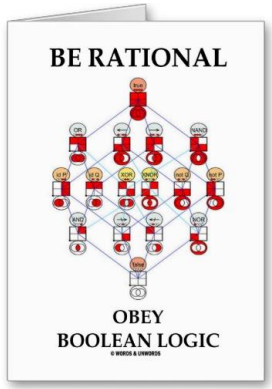
- A formula is in conjunctive normal form (CNF) if it is a conjunction of **clauses**, where a clause is a disjunction of **literals**

- Example :

$(x1 \vee x2 \vee \neg x3) \wedge (x1 \vee x2 \vee x3) \wedge (\neg x1 \vee x2 \vee x3) \wedge (x2)$

The size of the instance is the number of literals in total

In this example : 10



MAXIMUM SATISFIABILITY

- Input: A boolean formula F in conjunctive normal form consisting of **m clauses** and a nonnegative integer k.
- Task: Find a truth assignment satisfying at least k clauses.

$(x1 \vee \neg x2 \vee x3) \wedge (x1 \vee x2 \vee x3) \wedge (x1 \vee x2 \vee x3) \wedge (x2)$

- We will prove that the problem has a kernel of size k^2

Observation 1:

- If $k \leq \lceil m/2 \rceil$, then the desired truth assignment trivially exists:
  - Take a random truth assignment.
  - If it satisfies at least k clauses then we are done.
  - Otherwise, flip the assignment

$(x1 \vee \neg x2 \vee x3) \wedge (\neg x1 \vee \neg x2 \vee \neg x3) \wedge (x1 \vee x2 \vee x3) \wedge (x1) \wedge$

$(x1 \vee \neg x2)$

Class ex.:

1. Find an assignment that satisfies at most 2 clauses
2. Show that by flipping the assignment, 3 or more clauses are satisfied

- From now on we can assume that $k > \lceil m/2 \rceil$, which implies $m < 2k$

Flipping : example

(x1 ∨¬ x2 ∨x3)∧(¬x1 ∨¬x2 ∨¬x3)∧(x1 ∨x2 ∨x3)∧(x1) ∧
(x1∨ ¬x2)

x1=F x2=T x3=F

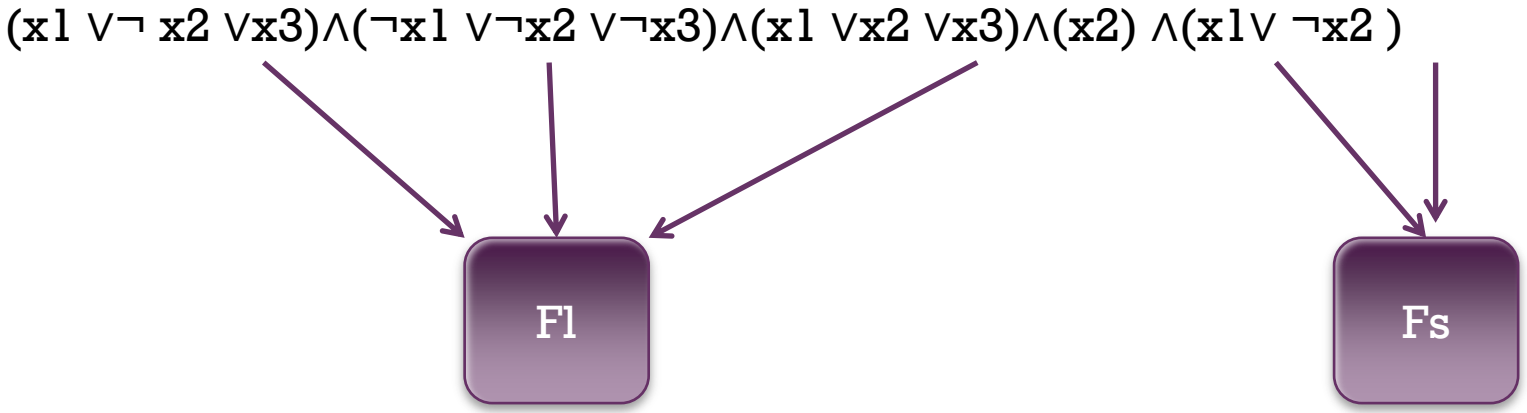
1,4,5 are false 2 and 3 are true. Let's fliip it!

X1=T x2=F x3=T

Without even looking at 2 or 3, we know that 1,4,5 are now satisfied

Observation 2:

- Partition the clauses of F into Fl and Fs:
  Fl: long clauses containing at least k literals;
  Fs: short clauses containing less than k literals.



Observation 2 (cont.)

- Let L := number of long clauses (|Fl|) .
- If L ≥ k, then at least k clauses can be satisfied.
  - for each (w.c) pick one literal and set it to satisfy the formula

(x1 ∨¬ x2 ∨x3 ∨ … ∨ x_k ∨ … ∨ x_{ci})

Fl

Fs

Now we are left with the case where L < k

Example

- (x1 ∨¬ x2 ∨x3)∧(¬x1 ∨¬x2 ∨¬x3)∧(x1 ∨x2 ∨x3)∧(x2) ∧(x1∨ ¬x2)
- The 3 members of Fl are the first 3.
- We can set x1 as the first clause demands (T), x2 for the second (F) and x3 for the third (T)
- We are now secure that at least k (3) clauses are satisfied

Observation 3:

- By now we already know that the instance has $L < k$ large clauses and the total number of clauses m is strictly smaller than $2k$
- (F, k) is a yes-instance **if and only if** $(Fs, k - L)$ is a yes-instance.

Observation 3:

- (F, k) is a yes-instance **if and only if** (Fs, k – L) is a yes-instance.

  ->if (F,K) has an assignment, there are between 0 and L large clauses satisfied but certainly no less than k-L small clauses.

  Thus, if there is an assignment for (F, k) there must also be one for (Fs, k – L)

Observation 3:

- (F, k) is a yes-instance **if and only if** (Fs, k − L) is a yes-instance.

<- if (Fs , k − L) is a yes-instance of MaxSat:

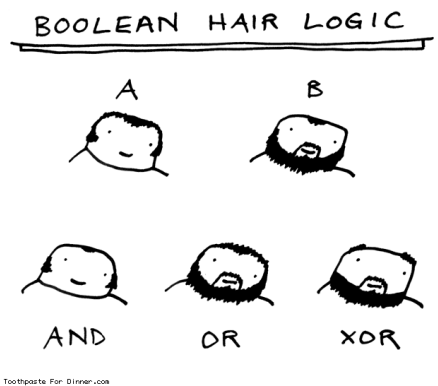
1. To satisfy k − L clauses at most k − L variables (at most one variable per satisfied clause) are needed to be assigned, leaving L variables unassigned.

2. we can use the trick before and satisfy all the large clauses and get (F, k) of MaxSat

Conclusion

- $|Fs| = O(k^2)$ because :
  - There are $m - L \leq m$ small clauses, each containing at most k literals
  - $m < 2k$
  - The total number of literal occurrences in Fs is bounded from above by $2k \cdot k = 2k^2$.

- MAXIMUM SATISFIABILITY has a problem kernel of size $O(k^2)$



The connection: Kernels and FPT

- **Theorem**: Consider a decidable parameterized problem. Then the problem belongs to FPT, **if and only if** it has a kernel

<= Build the kernel and then solve the problem on the kernel

=> A little trickier …

FPT -> Kernel

⇒ assume that the given fixed-parameter algorithm has running time $f(k) \cdot n^c$ for some positive constant c.

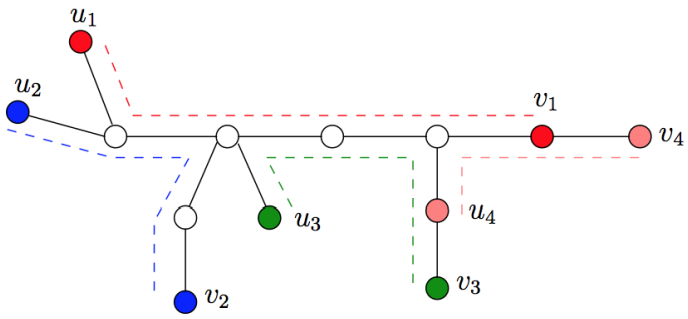
- Run this algorithm on the problem for at most n^{c+1} steps
- two cases are possible:
  1. The algorithm has finished its task within that time, now we have a kernelization algorithm running in polynomial time n^{c+1}, which simply outputs either a trivial "no"-instance or a trivial "yes"-instance.
  2. The algorithm is not finished, but we can argue that $n < f(k)$. Thus, our problem kernel is the original input instance itself.

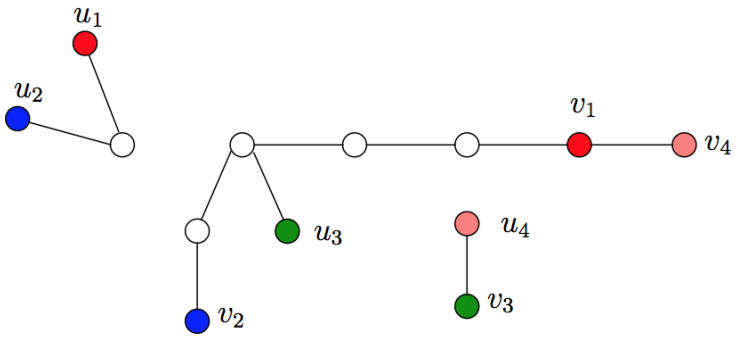
Out of the material but beautiful: Multicut in trees

- Input: An undirected tree $T = (V, E)$, $n := |V|$, and

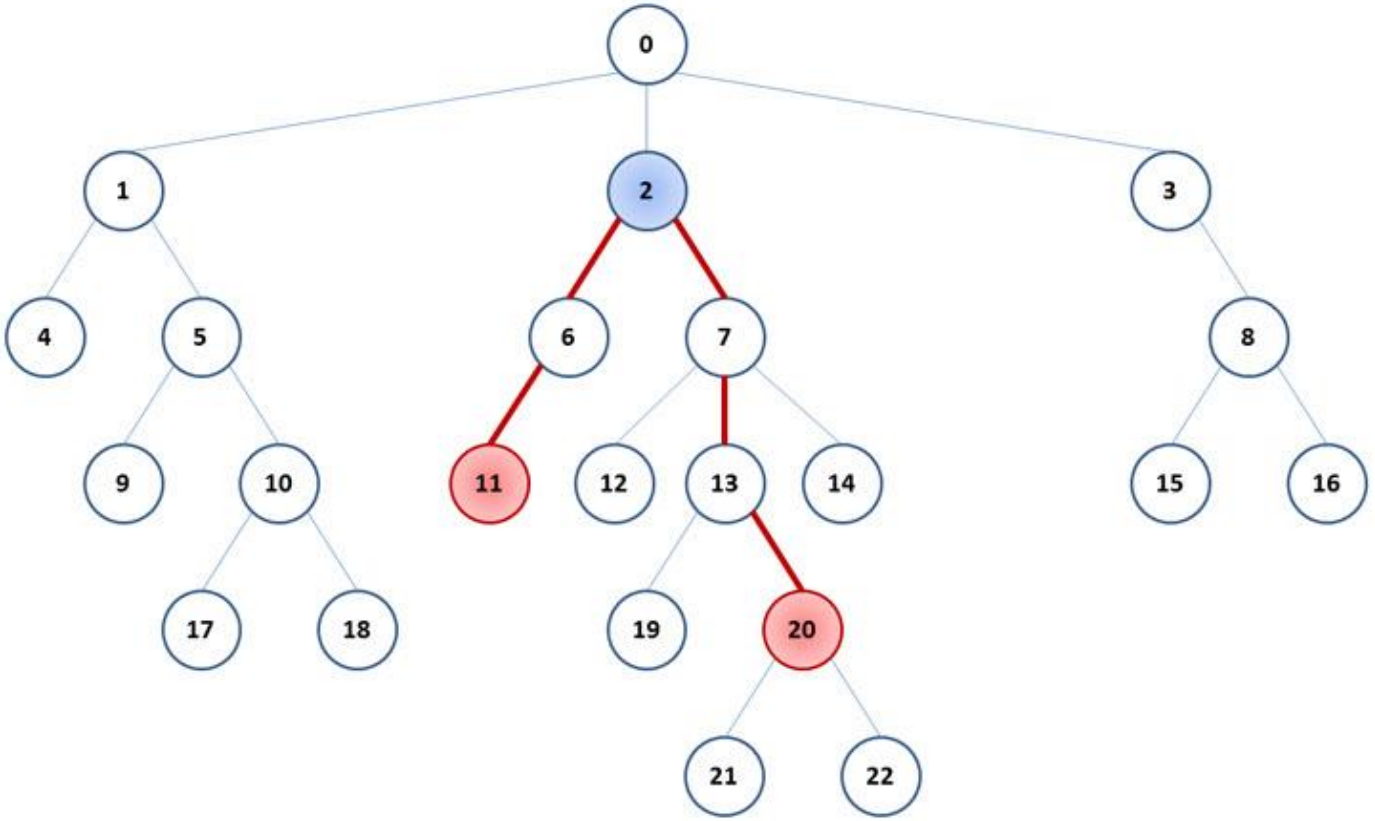
$H = \{(u_i, v_i) \mid u_i, v_i \in V,\ u_i \neq v_i, 1 \leq i \leq m\}$.

- Task: Find a minimum size set $E' \subseteq E$ such that the removal of the edges in E' separates each pair of nodes in H.





Least common ancestor in a tree



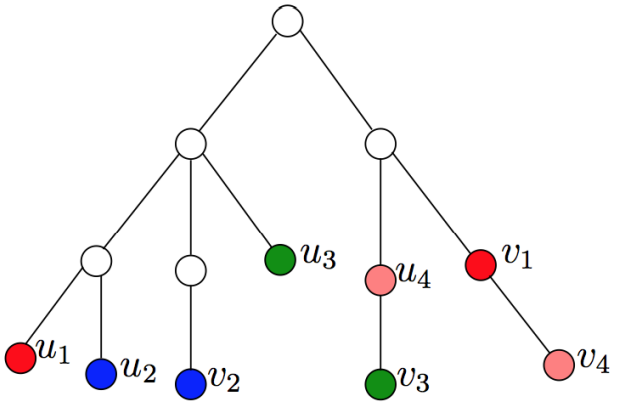
5 minute task

- What if instead of asking which edges must we remove, we ask what vertices? How can we solve this problem?

Multicut is FPT with respect to number of requests

Working bottom up, taking the NCA of each of the requests.
Search tree: 2^k (k := the number of edge deletions).

Recent work: the kernel is of $O(k^3)$



Tips and tricks for selecting the parameter

- Parameter really small
  - Example: Vertex Cover on planar graphs
- Guaranteed parameter value
  - Example: Independent set on planar graphs
- More than one obvious parameterization
  - Example :Weighted vertex cover
- Close to trivial problem instances
  - Example :Traveling Salesperson problem in the two-dimensional Euclidean plane