CSE 565

LECTURE 28

Computational Intractability

- One more NP-complete problem

NP-completeness as a Design Guide

Sofya Raskhodnikova

Some NP-Complete Problems

- Six basic genres of NP-complete problems and paradigmatic examples.
  – Packing problems: SET-PACKING, INDEPENDENT SET.
  – Covering problems: SET-COVER, VERTEX-COVER.
  – Constraint satisfaction problems: SAT, 3-SAT.
  – Sequencing problems: HAMILTONIAN-CYCLE, Traveling Salesman.
  – Partitioning problems: 3D-MATCHING 3-COLOR.
  – Numerical problems: SUBSET-SUM, KNAPSACK.
- Practice. Most NP problems are either known to be in P or NP-complete.
  – For most search problems, if the corresponding decision problem is in P, the search problem can be solved in polynomial time.
- Notable exceptions:
  – Decision problem: Graph isomorphism.
  – Search problems: Factoring, Nash equilibrium

3D matching

- Input
  - Disjoint sets X, Y, Z of the same size (call it n)
  - Collection T in $X \times Y \times Z$ of ordered triples
- Output
  - "yes" if there exists a set of n triples in T that cover all of $X \cup Y \cup Z$.
  - "no" otherwise
  - *Note*: Equivalently, we could ask for a set of n disjoint triples in T (why?)

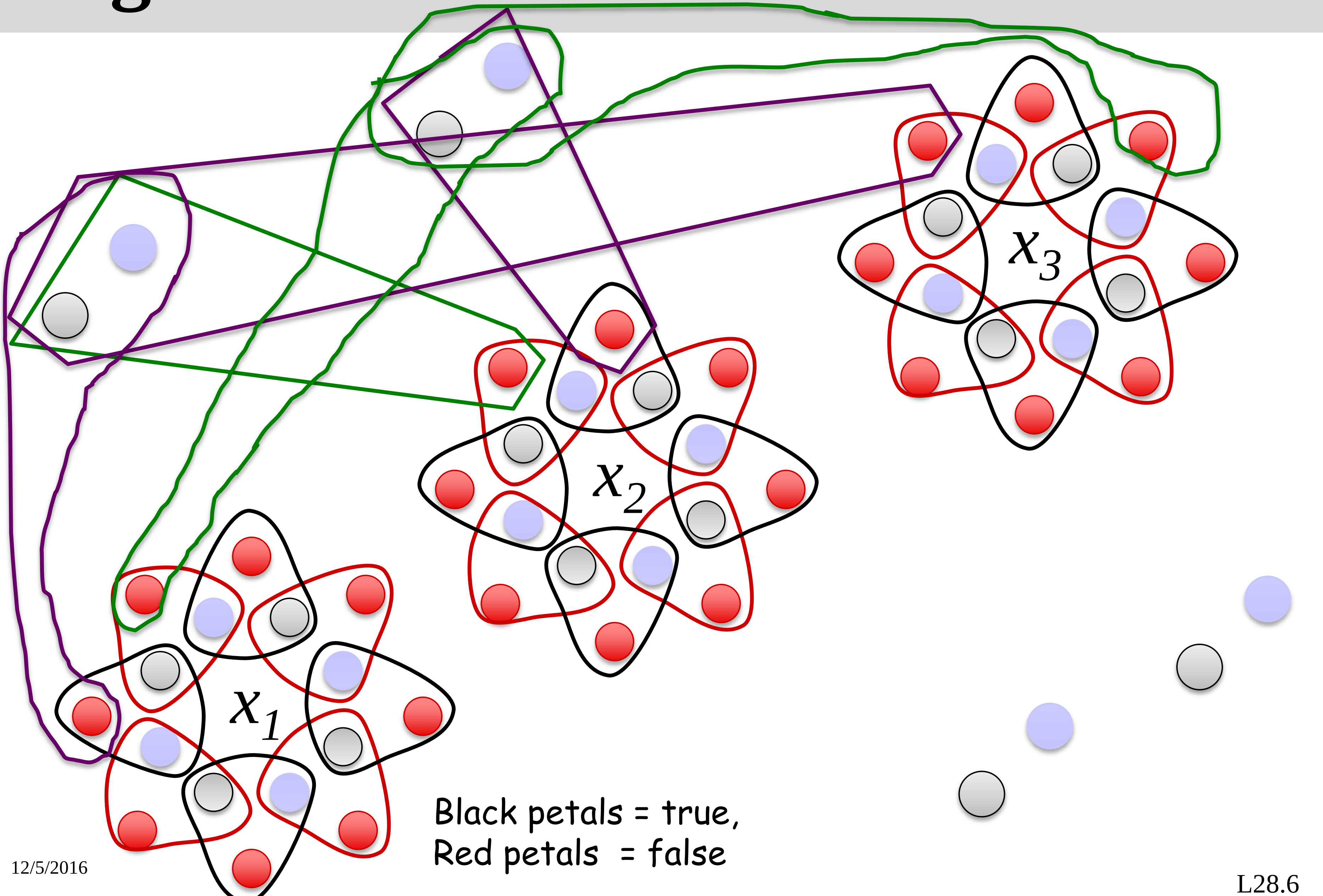
Review

- We wish to prove 3D-matching is NP-complete
  - We need to give 2 algorithms:
    - what are their inputs and outputs
    - what guarantees do they need to satisfy?

Reduction from 3-SAT to 3D matching

- Input: 3-CNF formula φ. Let m = #vars and k = #clauses
- Output: 3 sets X, Y, Z with $|X| = |Y| = |Z| = 2mk$ and a set of $2mk + 3k + 2m(m-1)k^2$ triples $T \subseteq X \times Y \times Z$
- Variable gadgets: $4k$ items for each variable
  - Core: ring of $2k$ items $a_{i,1}, \dots, a_{i,2k}$
  - $2k$ free tips $b_{i,1}, \dots, b_{i,2k}$
  - Triples: $(a_{i,j}, a_{i,j+1}, b_{i,j})$ for every $j = 1, \dots, 2k$
- Clause gadgets:
  - Pair $p_{t,1}, p_{t,2}$ for $t = 1, \dots, k$
  - For each literal (say, x_i or $\neg x_i$), add a triple $\left(p_{t,1}, p_{t,2}, b_{i,j}\right)$ where $b_{i,j}$ has not yet appeared in a similar triple, and j is even for x_i and odd for $\neg x_i$
- Cleanup gadgets:
  - $(m-1)k$ pairs of items $c_{\ell,1}, c_{\ell,2}$,
  - For each ℓ, add all possible triples $(c_{\ell,1}, c_{\ell,2}, b_{i,j})$.
  - (These allow you to cover unused triples.)

E.g. $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3})$



Black petals = true,
Red petals = false

Sets X,Y,Z and T

- On input ϕ, output:
  - $Z = \left\{ b_{i,j} : \begin{matrix} i = 1, \dots, m \\ j = 1, \dots, 2k \end{matrix} \right\}$
  - $X = \left\{ a_{i,j} : \begin{matrix} i = 1, \dots, m \\ j \ odd \end{matrix} \right\} \cup \{ p_{t,1} : t = 1, \dots, k \} \cup \{ c_{\ell,1} : \ell = 1, \dots, (m-1)k \}$
  - $Y = \left\{ a_{i,j} : \begin{matrix} i = 1, \dots, m \\ j \ even \end{matrix} \right\} \cup \{ p_{t,2} : t = 1, \dots, k \} \cup \{ c_{\ell,2} : \ell = 1, \dots, (m-1)k \}$
  - Triples T as on previous slide

Proof of correctness outline

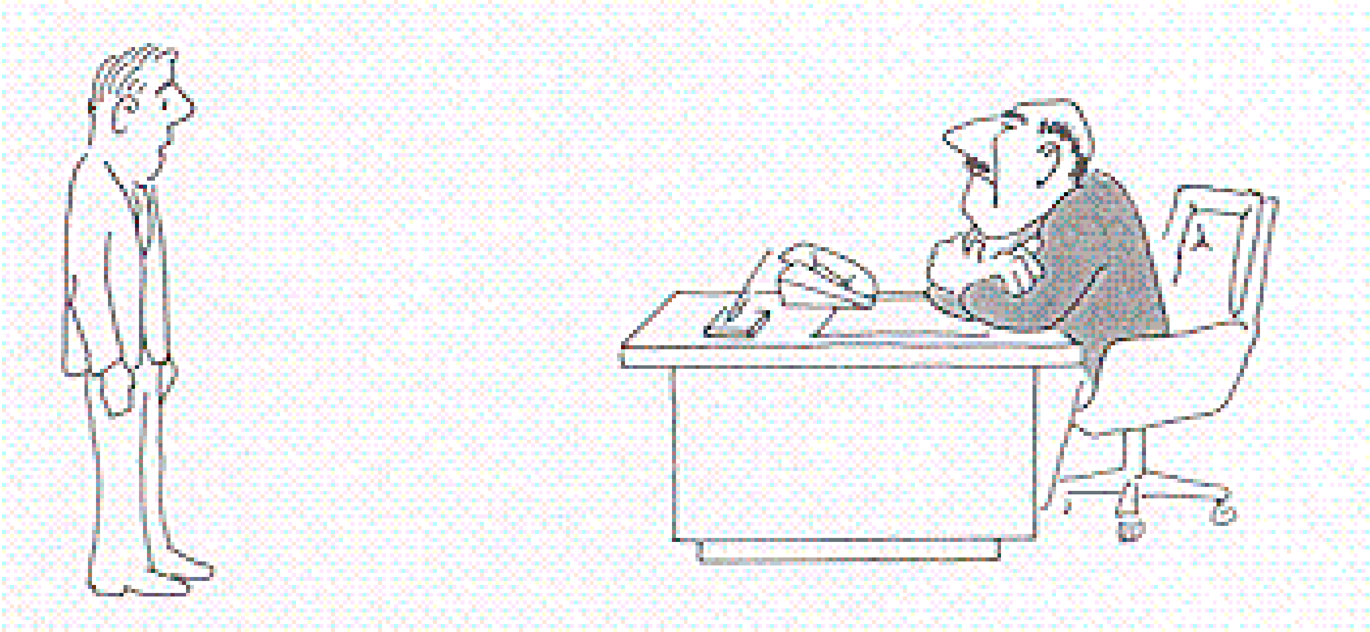
- Useful lemma: For each variable, solution contains either
  - all odd triples and no even ones, or
  - all even triples and no odd ones.

1. Reduction runs in polynomial time $O\left((mk)^2\right)$
2. If φ is satisfiable, then X, Y, Z, T have a 3D perfect matching
   - For each clause, use 1 satisfied literal to find triple
3. If X, Y, Z have a 3D matching, then φ satisfiable.
   - Each clause covered by one tripe corresponding to satisfied literal.

Exercise: Decision vs Search

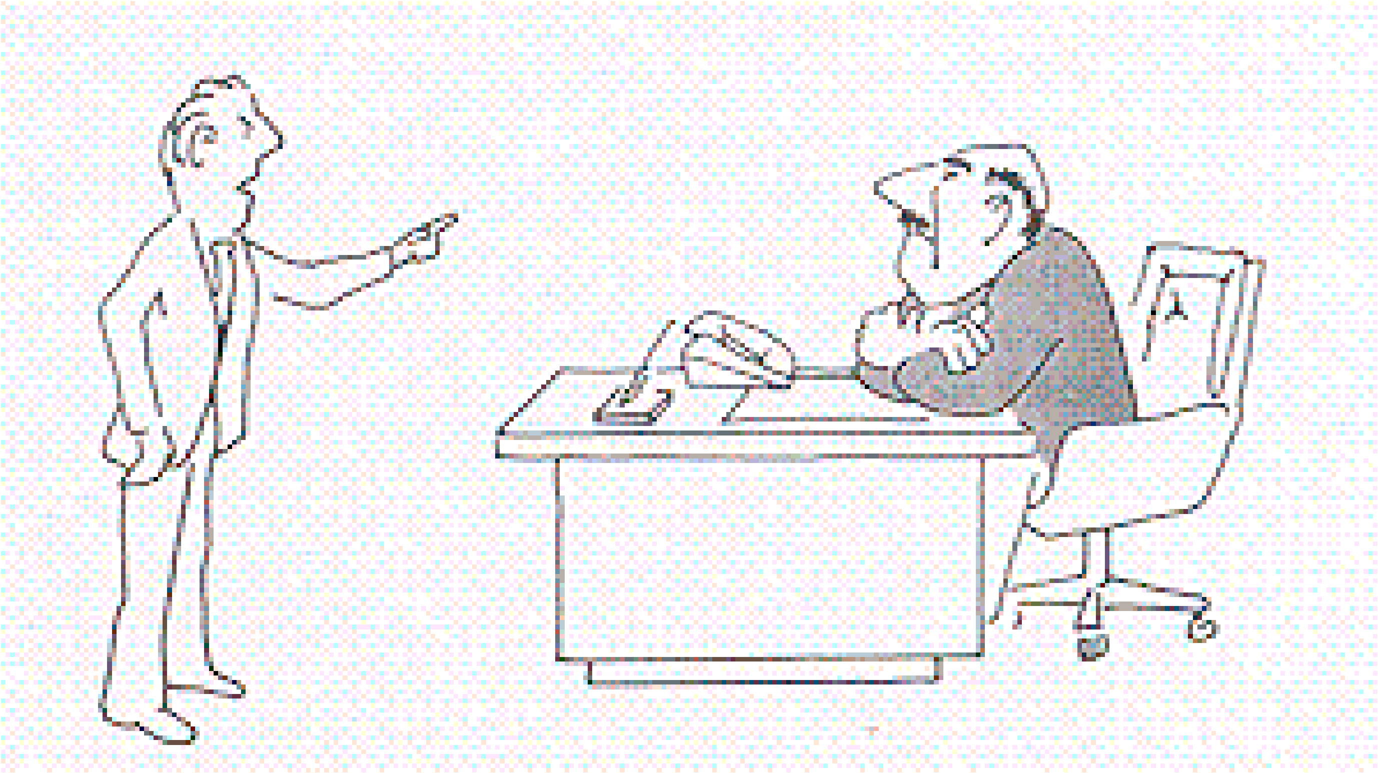
- The Matching fairy has given you a magic box that solves 3D matching in unit time.
  - How can you use it to find a matching?
  - Give an algorithm that uses O(n^2) calls to the magic box

Garey and Johnson's cartoon



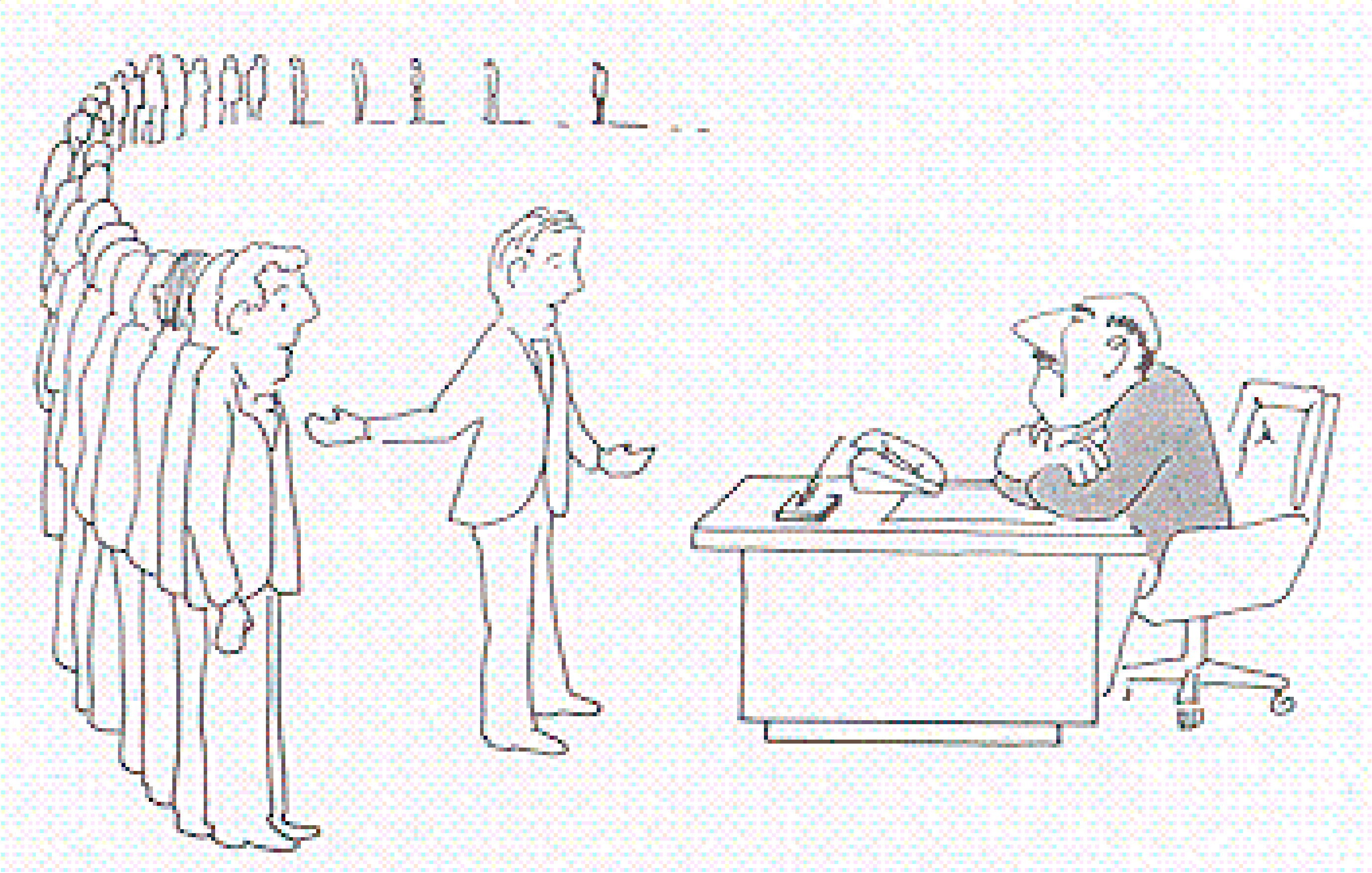
"I can't find an efficient algorithm,
I guess I'm just too dumb.“

Garey and Johnson's cartoon



"I can't find an efficient algorithm, because no such algorithm is possible! "

Garey and Johnson’s cartoon



“I can't find an efficient algorithm,
but neither can all these famous people.“

NP-Completeness as a Design Guide

Q. Suppose I need to solve an NP-complete problem. What should I do?
A. You are unlikely to find poly-time algorithm that works on all inputs.

Must sacrifice one of three desired features.

- Solve problem in polynomial time (→ *e.g.*, fast exponential algorithms)
- Solve arbitrary instances of the problem
- Solve problem to optimality (→ approximation algorithms)

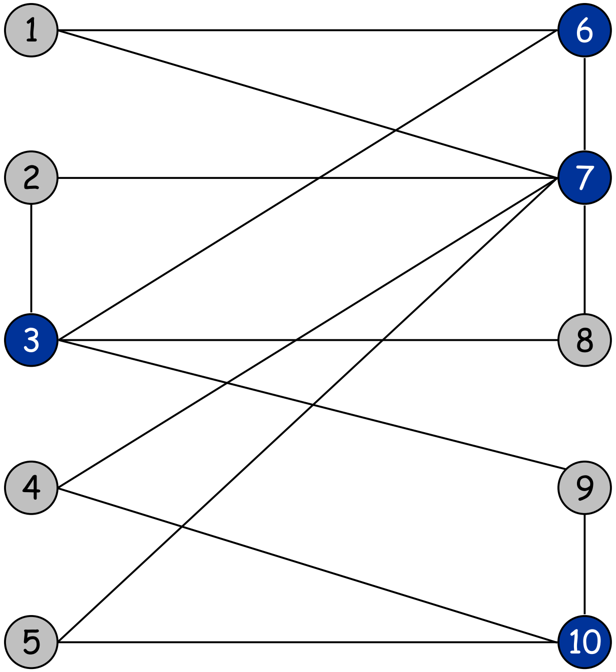
Today. Solve some special cases of NP-complete problems that arise in practice.

10.1 Finding Small Vertex Covers

- Suppose vertex cover describes warehouse "placement" problem,
  - e.g.: how many warehouses (placed in cities) are needed so there is one at an endpoint of every designated highway segment?
- Not interested if the answer is larger than 10
- This is vertex cover (of the highway graph) with $k < 10$

Vertex Cover

VERTEX COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge (u, v) either $u \in S$, or $v \in S$, or both.



$k = 4$
$S = \{ 3, 6, 7, 10 \}$

Finding Small Vertex Covers

Q. How fast can we solve Vertex Cover for small k?

First attempt: Brute force.

- Try all $\binom{n}{k} = \Theta(n^k)$ subsets of size k.
- Takes $O(k\,n)$ time to check whether a subset is a vertex cover.
- Crude time bound: $O(k\,n^{k+1})$.
- Slow even for k=10.

Second attempt. Limit exponential dependency on k, e.g., to $O(2^k\,k\,n)$.

Ex. n = 1,000, k = 10.

Brute. $k\,n^{k+1} = 10^{34} \Rightarrow$ infeasible.

Better. $2^k\,k\,n = 10^7 \Rightarrow$ feasible.

Remark. If k is a constant, algorithm is poly-time; if k is a small constant, then it's also practical.

Important. The algorithm is still exponential, and hence scales badly (e.g., consider k=40). However, it is better than brute force.

Finding Small Vertex Covers

Idea: Recursive solution similar to self-reducibility argument.

Claim 1. Let u-v be an edge of G. G has a vertex cover of size $\leq$ k iff at least one of $G - \{ u \}$ and $G - \{ v \}$ has a vertex cover of size $\leq$ k-1.

Proof. $\Rightarrow$

- Suppose G has a vertex cover S of size $\leq$ k.
- S contains either u or v (or both).
- Without loss of generality, assume it contains u.
- $S - \{ u \}$ is a vertex cover of $G - \{ u \}$.

Proof. $\Leftarrow$

- Suppose S is a vertex cover of $G - \{ u \}$ of size $\leq$ k-1.
- Then $S \cup \{ u \}$ is a vertex cover of G. ▪

Finding Small Vertex Covers: Algorithm

Claim 2. The following algorithm find a vertex cover of size $\leq$ k in G if it exists and runs in $O(2^k\, n)$ time.

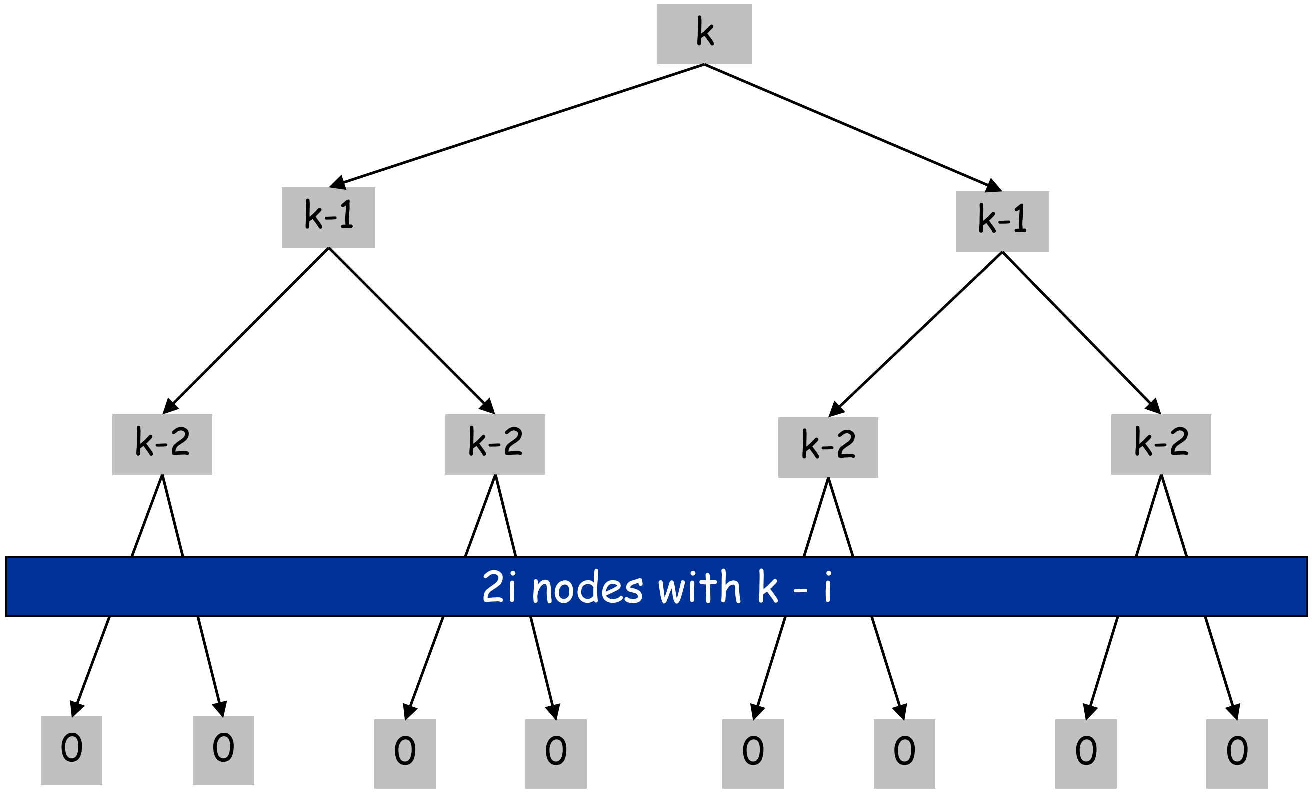
```
SmallVC(G, k) {
   if k=0
      if (G contains no edges) return ;
      else return false

   let (u, v) be any edge of G
   S-u = SmallVC(G - {u}, k-1)
   if S-u ≠ false return S-u ∪ {u}

   S-v = SmallVC(G - {v}, k-1)
   if S-v ≠ false return S-v ∪ {v}
   else return false
}
```

Proof.

- Correctness follows from Claim 1.
- There are $\leq 2^{k+1}$ nodes in the recursion tree; each invocation takes O(n) time. ▪

Finding Small Vertex Covers: Recursion Tree

$$T(n,\,k) \le \begin{cases} cn & \text{if } k = 0 \\ 2T(n, k-1) + cn & \text{if } k > 1 \end{cases} \quad \Rightarrow \quad T(n,\,k) \le 2^{k+1} c\ n$$



Another algorithm for small VC

As before. Remove or include one vertex u (as in fast exponential algorithm)

- If u is in VC, remove adjacent edges
  - T(n-1,k-1)
- If u not in VC, remove adjacent edges, add all neighbors of u to VC
  - T(n-1-deg(u) , k-deg(u))

Idea. By choosing vertex u with largest degree (at least |E|/k, otherwise no VC of size k exists), get better exponent in k

Exercise: how small an exponent in k can you get while maintaining linear scaling in n?

Summary

Often input size is too crude a measure of complexity

- e.g., VC algorithm linear in n, exponential in k

Parameterized complexity

- General theory of such problems
- Clever algorithms, hardness arguments

Take away message:

- When facing a seemingly hard problem, look for what “really” makes it hard