Lecture 24

Want:
1. hard problems (NP-complete)
2. fast algorithms (poly. time)
3. exact solutions (correctness)

PICK ANY TWO

2 & 3: most of this class
1: dealing with NP-complete problems (this week)

1 & 2: approximation algorithms (L25)
- allow suboptimal solutions
- bound the error (e.g. within 5%)

1 & 3: fixed-parameter algorithms (today)
- allow exponential time
- contain exponential to parameter, not \( n \)
- improve understanding of exponential behavior
- get fast solutions when parameter is small
**Fixed-parameter algorithms** [Downey & Fellows 1997]

For some computational problem $\pi$, a **parameter** is a function $P$ mapping (inputs for $\pi$) to (nonnegative integers). Normally write $k = P(\text{input})$.

$(\pi, P) = "\pi\text{ with respect to } P\text{"}$ is a **parameterized problem**.

- most decision problems come equipped with a "natural parameter":

**CLIQUE**
- **input**: graph $G$, nonneg. integer $k$
- **output**: does $G$ have a clique of size $k$?

Goal: confine exponential behavior to $k$ instead of $n$
**Vertex cover:**
- in an undirected graph $G = (V, E)$,
  a **vertex cover** is a subset $S \subseteq V$
  such that every edge has an endpt. in $S$
  i.e. $(u, v) \in E \Rightarrow u \in S$ or $v \in S$
- e.g. $V$ is a vertex cover

![Vertex cover graph]

**$|S| = 3$**

**VERTEX COVER problem:**
- **input:** graph $G$, nonneg. integer $k$
- **output:** does $G$ have a vertex cover $S$
  of size $|S| = k$? (i.e. $|S| \leq k$)

**Note:** can have $n$ large, $k$ small

![Vertex cover graph]

$|S| = 1$
Trivial algorithm: (BAD)
- try all \((k)\) subsets \(S\) of \(k\) vertices
- for each \(S\), test coverage in \(O(V+E)\) time
  \(\Rightarrow O(V^k (V+E))\) time
- is this any good?
  - polynomial for any fixed \(k\)
  - but not the same polynomial
    e.g., not \(O(n^{100})\) for any fixed \(k\)
  - inefficient in most cases in practice
  \(\Rightarrow\) define \(n_f(k)\) to be BAD
Smarter algorithm: (GOOD)

- Pick arbitrary edge \( e = (u, v) \)
- Know that either \( u \in S \) or \( v \in S \) (or both)
  - but don't know which
- Guess: try both possibilities
  1. Add \( u \) to \( S \)
     - Delete \( u \) & incident edges from \( G \)
     - Recurse with \( k-1 \)
  2. Ditto with \( v \) instead of \( u \)

[This actually covers \( 3^{rd} \) case where \( u, v \in S \)]

- Like guessing in dynamic programming
- Except memoization doesn't help here
- Recursion tree:

\[
\begin{align*}
&\text{FAIL} & &\text{YES} \\
&\begin{array}{c}
\quad u \\
\quad u' \\
\quad v \\
\quad v' \\
\end{array} \\
&k
\end{align*}
\]

- At leaf, when \( k = 0 \):
  - If \( |E| = 0 \) then return \text{YES} else \text{FAIL}
- \( \text{answer = YES} \iff \text{any leaf returns YES} \)

\( \Rightarrow O(2^k V) \) time
- \( O(V) \) for any fixed \( k \)
- Polynomial degree is independent of \( k \)
- Also polynomial for \( k = O(lg n) \)
- Practical for e.g. \( k \approx 32 \)
Fixed-parameter tractability (FPT): a parameterized problem is FPT if it has a (correct) algorithm whose running time is \( \leq f(k) \cdot n^{O(1)} \) for some function \( f : \mathbb{N} \to \mathbb{N} \) independent of \( n \) & \( k \).

**Question:** Why \( f(k) \cdot n^{O(1)} \), not \( f(k) + n^{O(1)} \)?

**Theorem:** there's an \( f(k) \cdot n^{O(1)} \)-time algorithm \( \iff \) there's an \( f'(k) + n^{O(1)} \)-time algorithm

**Proof:**

(\( \Rightarrow \)) trivial (assuming \( f(k) \cdot n^{O(1)} \geq 1 \))

(\( \Leftarrow \)) if \( n \leq f(k) \) then \( f(k) \cdot n^c \leq f(k)^{c+1} \)
if \( f(k) \leq n \) then \( f(k) \cdot n^c \leq n^{c+1} \)

so \( f(k) \cdot n^c \leq \max \{ f(k)^{c+1}, n^{c+1} \} \leq \frac{f(k)^{c+1} + n^{c+1}}{f'(k)} \).

**Example:** \( O(2^k n) \Rightarrow O(4^k + n^2) \).
**Kernelization:** a simplifying self-reduction

An $f(k)$-kernelization algorithm converts an input $(x, k)$ into a small, equivalent input $(x', k')$:
- $\text{answer}(x) = \text{answer}(x')$;
- $|x'| \leq f(k)$;
- $k' \leq k$.

**Theorem:** every FPT parameterized problem has an (exponential) kernelization.

**Proof:** given $f(k) \cdot n^c$ algorithm

if $n \leq f(k)$ then already kernelized
if $f(k) \leq n$ then
- run FPT algorithm in $n^{c+1}$ time
- output canonical $O(1)$-size YES or NO input accordingly.

⇒ Fixed-parameter algorithms are all about how small you can reduce the input size using polynomial-time preprocessing.

Current research is in finding small (polynomial or even linear) kernelizations.
Polynomial kernel for vertex cover:

1. remove loops \( \bullet \) & multi-edges \( \bullet \bullet \)
   - any vertex \( v \) of degree \( >k \) must be in \( S \)
     (otherwise we would need \( >k \) vertices in \( S \)
     to cover the edges incident to \( v \) )
2. while such a vertex \( v \) exists
   do add \( v \) to \( S \)
     delete \( v \) and its incident edges
     \( k \leftarrow k - 1 \)
   resulting graph has maximum degree \( \leq k \)
3. each further vertex we add to \( S \)
   covers \( \leq k \) edges
4. if \( |E| > k^2 \) then answer is NO:
   return canonical NO instance
   \( \text{else } |E| \leq k^2 \)
5. delete degree-0 vertices
   \( |V| \leq 2k^2 \), \( |V| + |E| \leq 3k^2 \)
   reduced input size to \( O(k^2) \) in \( O(V+E) \) time

Now apply

- trivial solution \( \Rightarrow O((2k^2)^k \times k^2 + V+E) \) time
  \( = O(2^{2k^2 k} + k^2 + 2k + 2k \times k^2 + V+E) \)
- smarter solution \( \Rightarrow O(2^k \times k^2) \) time