Review for Final Exam: Wednesday, December 4\textsuperscript{th}
PSOs This Week: Review for Final Exam
No Class on Friday, December 6\textsuperscript{th}
Please let me know what you liked and what could be improved
  - “NP is too hard”
Closes December 8th at 11:59PM
Feedback is anonymous and will have no impact on final grades
Chapter 10
Extending the Limits of Tractability
Q. Suppose I need to solve an NP-complete problem. What should I do?
A. Theory says you're unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to **optimality**.
- Solve problem in **polynomial time**.
- Solve **arbitrary instances** of the problem.

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

**Example:** Knapsack is NP-Hard
- Can find solutions that are very close to optimal in polynomial time
- Can efficiently solve instances when all weights are small
- Can also efficiently solve instances when all values are small...(next slide)
Def. $OPT(i, v) = \text{min weight subset of items } 1, \ldots, i \text{ that yields value exactly } v$.

- **Case 1:** $OPT$ does not select item $i$.
  - $OPT$ selects best of $1, \ldots, i-1$ that achieves exactly value $v$.

- **Case 2:** $OPT$ selects item $i$.
  - adds weight $w_i$, new value needed = $v - v_i$.
  - $OPT$ selects best of $1, \ldots, i-1$ that achieves value exactly $v - v_i$.

$$
OPT(i, v) = \begin{cases} 
0 & \text{if } v = 0 \\
\infty & \text{if } i = 0, \ v > 0 \\
OPT(i - 1, v) & \text{if } v_i > v \\
\min \{OPT(i - 1, v), w_i + OPT(i - 1, v - v_i)\} & \text{otherwise}
\end{cases}
$$

Running time. $O(n V^*) = O(n^2 v_{\text{max}})$.

- $V^*$ = optimal value $= \text{maximum } v \text{ such that } OPT(n, v) \leq W$.
- Not polynomial in input size!
Knapsack: FPTAS

Polynomial Time Approximation Scheme (PTAS): \((1 + \varepsilon)\)-approximation algorithm for any constant \(\varepsilon > 0\).

Intuition for approximation algorithm.
- Round all values up to lie in smaller range.
- Run dynamic programming algorithm on rounded instance.
- Return optimal items in rounded instance.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>934,221</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5,956,342</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>17,810,013</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>21,217,800</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>27,343,199</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

\(W = 11\)
Assume $P \neq NP$. Which of the following claims are necessarily true? (Let $n$ denote the number of variables in a 3SAT formula)

A. There is no algorithm that solves 3SAT in time $O(1.5^n)$

B. There is no algorithm that solves arbitrary 3SAT instances in time $O(n^{20000000})$

C. There is no algorithm running in time $O(n^2)$ that successfully solves most 3SAT instances (and occasionally outputs “I don’t know” for hard instances that the algorithm cannot solve)

D. Claims B and C are both true

E. Claims A, B and C are all true.
Assume $P \neq NP$. Which of the following claims are necessarily true? (Let $n$ denote the number of variables in a 3SAT formula)

A. There is no algorithm that solves 3SAT in time $O(1.5^n)$
   - A. [KS10] deterministic $O(1.439^n)$ algorithm for 3SAT
   - B. [HMS11]: randomized time $O(1.321^n)$

B. There is no algorithm that solves arbitrary 3SAT instances in time $O(n^{20000000})$
   - A. this would imply $P=NP$ as $O(n^{20000000})$ is still polynomial time.

C. There is no algorithm running in time $O(n^2)$ that successfully solves most 3SAT instances (and occasionally outputs “I don’t know” for hard instances that the algorithm cannot solve)
   - A. Heuristic solvers often work quite quickly in practice

D. Claims B and C are both true

E. Claims A, B and C are all true.
10.1 Finding Small Vertex Covers
**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.

![Graph Diagram]

$k = 4$

$S = \{ 3, 6, 7, 10 \}$
Finding Small Vertex Covers

Q. What if $k$ is small?

Brute force. $O(kn^{k+1})$.
- Try all $\binom{n}{k} = \mathcal{O}(n^k)$ subsets of size $k$.
- Takes $O(kn)$ time to check whether a subset is a vertex cover.

Goal. Limit exponential dependency on $k$, e.g., to $O(2^k kn)$.

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

Brute. $kn^{k+1} = 10^{34} \Rightarrow$ infeasible.
Better. $2^k kn = 10^7 \Rightarrow$ feasible.

Remark. If $k$ is a constant, algorithm is poly-time; if $k$ is a small constant, then it's also practical.
Claim. 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 $\leq k-1$.

Pf. $\Rightarrow$

- Suppose $G$ has a vertex cover $S$ of size $\leq k$.
- $S$ contains either $u$ or $v$ (or both). Assume it contains $u$.
- $S - \{u\}$ is a vertex cover of $G - \{u\}$.

Pf. $\Leftarrow$

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

Claim. If $G$ has a vertex cover of size $k$, it has $\leq k(n-1)$ edges.

Pf. Each vertex covers at most $n-1$ edges. $\blacksquare$
Finding Small Vertex Covers: Algorithm

Claim. The following algorithm determines if $G$ has a vertex cover of size $\leq k$ in $O(2^k kn)$ time.

```java
boolean Vertex-Cover(G, k) {
    if (G contains no edges) return true
    if (G contains $\geq kn$ edges) return false

    let (u, v) be any edge of G
    a = Vertex-Cover(G - {u}, k-1)
    b = Vertex-Cover(G - {v}, k-1)
    return a or b
}
```

Pf.
- Correctness follows from previous two claims.
- There are $\leq 2^{k+1}$ nodes in the recursion tree; each invocation takes $O(kn)$ time. □