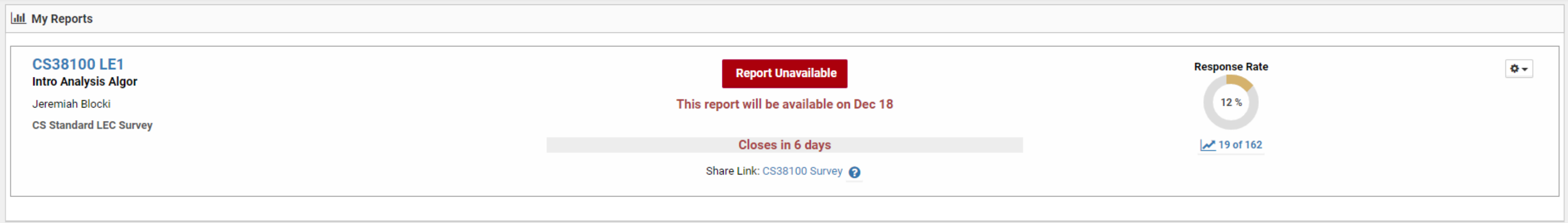
CS 381 – FALL 2019

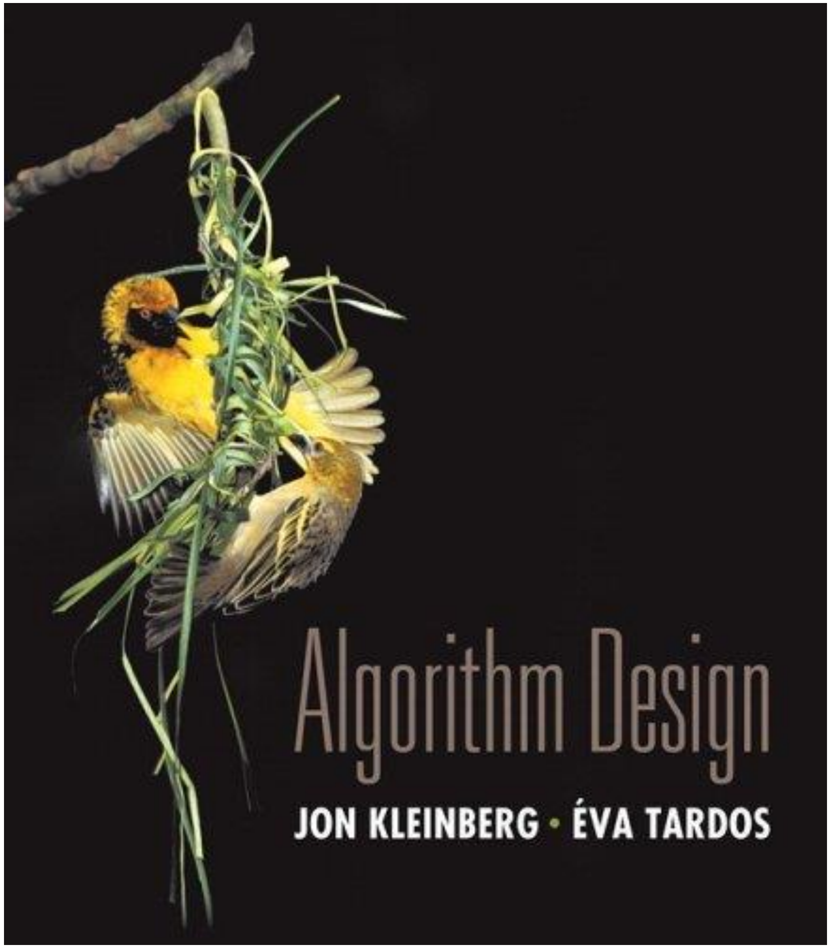
Week 16.1, Monday, December 2nd

Review for Final Exam: Wednesday, December 4th
PSOs This Week: Review for Final Exam
No Class on Friday, December 6th

Reminder: Course Evaluation

- Please let me know what you liked and what could be improved
  - **http://www.purdue.edu/idp/courseevaluations/CE_Students.html**
  - **"NP is too hard"**
- Closes December 8th at 11:59PM
- Feedback is anonymous and will have no impact on final grades





Extending the Limits of Tractability

Slides by Kevin Wayne.

Coping With NP-Completeness

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)

Knapsack Problem: Dynamic Programming II

Def. OPT(i, v) = min weight subset of items 1, …, i that yields value exactly v.

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

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

$V^* \leq n\, v_{max}$

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

- V^* = optimal value = maximum v 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.

Item	Value	Weight
1	934,221	1
2	5,956,342	2
3	17,810,013	5
4	21,217,800	6
5	27,343,199	7

W = 11

original instance

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

rounded instance

iClicker Question

Assume P≠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.

iClicker Question

Assume P≠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

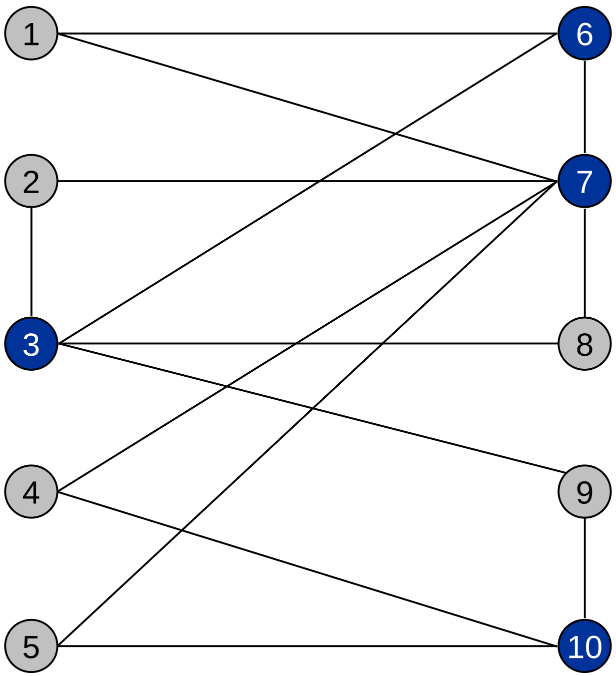
D. Claims B and C are both true

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.



$k = 4$

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

Finding Small Vertex Covers

Q. What if k is small?

Brute force. $O(k\, n^{k+1})$.

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

Goal. 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.

Finding Small Vertex Covers

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.

delete v and all incident edges

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. ▪

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. ▪

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.

```
boolean Vertex-Cover(G, k) {
   if (G contains no edges)    return true
   if (G contains ≥ 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. ▪