Preferred Midterm Option

- Exam will be similar in format to the previous exams posted on the webpage. These were designed to take 2 hours.
- Exam will cover material up to the end of the randomized algorithms material (lecture 17).
- Exam will be “active” from 8pm Thursday 11/5 to Friday 8pm 11/6.
- Let’s decide the timing of the exam:
  1. Option 1: Exam must be completed within 3 hours (although it’s a two hour exam) from when you start it. Handwritten solutions can be scanned/photographed and uploaded.
  2. Option 2: You can start and submit anytime in the 24 hour window. Solutions should be typed.
- What do you prefer? Option 2 is more flexible but that doesn’t really give you an advantage since everyone is in the same boat. On the other hand, Option 1 requires you to have the opportunity to be distraction free for 3 hours.
There are many computational problems where it is widely believed that there does not exist a polynomial time algorithm, e.g.,

- Finding the vertex cover of minimum size.
- Finding the maximum cut in a graph.
- The knapsack problem.
- **MAX-3-SAT**: A 3-SAT formula has the following form:

\[
(x_1 \lor x_2 \lor x_3) \land (x_4 \lor \bar{x}_2 \lor x_9) \land \ldots \land (\bar{x}_2 \lor \bar{x}_4 \lor x_8)
\]

where \( \lor \) means “or”, \( \land \) means “and”, and \( \bar{\cdot} \) means “not”. Each term in a bracket is called a “clause”. What’s the maximum number of clauses that can be satisfied?
Outline

Approximation Algorithms

Vertex Cover

Max Cut

Set-Cover
Approximation Ratios

Instead of finding the absolute minimum or maximum solution, can we design a polynomial time algorithm that is guaranteed to find an “almost” minimum or maximum solution.

Definition
The performance ratio of an algorithm is

\[
\max_{x: |x| = n} \frac{C_{\text{alg}}(x)}{C_{\text{opt}}(x)} \quad \text{for a minimization problem}
\]

\[
\max_{x: |x| = n} \frac{C_{\text{opt}}(x)}{C_{\text{alg}}(x)} \quad \text{for a maximization problem}
\]

where \( C_{\text{alg}}(x) \) is the value of the algorithm solution on input \( x \) and \( C_{\text{opt}}(x) \) is the value of the optimal solution on input \( x \).
Outline

Approximation Algorithms

Vertex Cover

Max Cut

Set-Cover
Vertex Cover

- **Input:** Graph $G = (V, E)$
- **Goal:** Find the vertex cover of smallest size. Recall that $U \subseteq V$ is a vertex cover iff at least one end point of each edge is in $U$. 
2-approximation for Vertex Cover

Algorithm

1. \( S = \emptyset \)
2. While \( E \neq \emptyset \), pick an edge \( e = (u, v) \in E \)
   - \( S \leftarrow S + u + v \)
   - \( V \leftarrow V - u - v \)
3. Return \( S \)

Theorem

The above algorithm returns a 2-approximation in polynomial time.

Proof.

- Let \( E' \) be the set of edges chosen:
  
  \[
  \text{size of vertex cover found} = 2|E'| 
  \]

- For any \( (u, v) \in E' \), at least one of \( \{u, v\} \) is in any vertex cover:
  
  \[
  \text{size of optimal vertex cover} \geq |E'| 
  \]
Outline

Approximation Algorithms

Vertex Cover

Max Cut

Set-Cover
Max Cut

Input: Unweighted graph $G = (V, E)$?

Goal: Find the cut $(A, B)$ that maximizes

$$|e = (u, v) \in E : u \in A, v \in B|$$
Max Cut Approximation Algorithm

Algorithm

1. Let $A = \emptyset, B = V$

2. While $\exists v \in V$ such that switching side of $v$ increases size of cut:

   move $v$ to other side of cut

3. Return $(A, B)$

Theorem
The algorithm is a 2-approximation and runs in polynomial time.
Max-Cut Analysis

- Number of switches is at most $|E|$
- When the algorithm terminates, let

$$a(v) = \text{number of edges from } v \text{ that cross the cut}$$
$$b(v) = \text{number of edges from } v \text{ that don’t cross the cut}$$

- Note that $a(v) \geq b(v)$ and so $\sum_v a(v) \geq \sum_v b(v)$
- But $\sum_v a(v) + \sum_v b(v) = 2|E|$

$$\text{cut size} = \sum_v a(v)/2 \geq \sum_v a(v)/4 + \sum_v b(v)/4 = |E|/2$$
Outline

Approximation Algorithms

Vertex Cover

Max Cut

Set-Cover
Set-Cover

Problem:
- Input: A collection $C = \{S_1, S_2, \ldots, S_m\}$ of subsets of $U = \bigcup_{S \in C} S$ and weights $w : C \rightarrow \mathbb{R}^+$
- Output: Find $C' \subset C$ such that

$$U = \bigcup_{S \in C'} S$$

that minimizes $|C'|$.

Greedy Algorithm: Repeatedly pick that set $S$ that covers the maximum number of currently uncovered elements.
Approximation Algorithm for Set Cover: Analysis

Suppose it is possible to cover all elements with \( k \) sets. Whenever you haven’t covered all the elements, there’s a set that covers at least \( 1/k \) fraction of the uncovered elements. To see this, suppose \( T_1, \ldots, T_k \) are the optimum sets and \( U' \) are the currently uncovered elements. Then,

\[
(T_1 \cap U') \cup (T_2 \cap U') \cup \ldots \cup (T_k \cap U') = U'
\]

since every element in \( U' \) is in some \( T_i \). But then

\[
\sum_i |T_i \cap U'| \geq |(T_1 \cap U') \cup (T_2 \cap U') \cup \ldots \cup (T_k \cap U')| \geq |U'|
\]

Hence for some \( i \), \( |T_i \cap U'| \geq |U'|/k \).

After \( t \) sets have been chosen the number of uncovered elements is

\[
n(1 - 1/k)^t < ne^{-t/k}
\]

For \( t = \lceil k \ln n \rceil \) this is less than 1, i.e., all elements have been covered.