Outline

More Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness

Approximation Algorithms

Linear Programming
Formulating Vertex Cover as a Linear (?) Program

- Given graph $G = (V, E)$, for each node $v \in V$, create variable $x_v$
- For each edge $(u, v) \in E$, create constraint $x_v + x_u \geq 1$

Minimize $\sum_{v \in V} x_v$ subject to

$$x_v + x_u \geq 1 \quad \text{for all } (u, v) \in E$$
$$x_v \leq 1 \quad \text{for all } v \in V$$
$$x_v \geq 0 \quad \text{for all } v \in V$$

Does this mean we can solve Vertex Cover in poly-time? No, need to constraints $x_v \in \{0, 1\}$ and program is linear integer program.
LP Relaxation

- Vertex cover can be expressed as the following integer program
- Minimize $\sum_{v \in V} x_v$ subject to

\[
\begin{align*}
    x_v + x_u & \geq 1 \quad \text{for all } (u, v) \in E \\
    x_v & \leq 1 \quad \text{for all } v \in V \\
    x_v & \geq 0 \quad \text{for all } v \in V
\end{align*}
\]

where each $x_v \in \{0, 1\}$.

- **Relax**: Replace $x_v \in \{0, 1\}$ constraint by $0 \leq x_v \leq 1$
- **Solve**: Let $\hat{x}_v$ be optimal solution.
- **Round**: Let $x'_v = 1$ if $\hat{x}_v \geq 1/2$ and 0 otherwise.
- **Final solution is feasible for the original ILP and is a 2-approx.**
Approx Algorithms and Reductions: Cautionary Tale!

Suppose \( \Pi' \leq_P \Pi \) and we have an polynomial time \( \alpha \)-approximation for a \( \Pi \), do we necessarily have an \( \alpha \) approximation for \( \Pi \)?

**Problem:** Independent Set

- **Input:** An undirected graph \( G = (V, E) \).
- **Output:** A set \( U \subset V \) of maximum size such that no two vertices in \( U \) are connected by a single edge.

**Lemma**

\( \text{Independent-Set} \leq_P \text{Vertex-Cover} \)

**Proof.**

\( U \subset V \) is an independent set iff \( V - U \) is a vertex cover. \( \square \)

But using a factor 2-approximation for Vertex-Cover may give a factor \( \Omega(n) \) approximation for Independent-Set.
Problem: Max-3-SAT

- Input: A 3-CNF formula with \( m \) clauses and \( n \) variables
- Output: Maximum number of clauses that can be satisfied.

*Recall reduction from 3-SAT:* Given formula \( \phi \), construct graph \( G_{\phi} \) such that each clique corresponds to a set of simultaneously satisfiable clauses. Hence an \( \alpha \) approx. for CLIQUE gives an \( \alpha \) approx. for MAX-3-SAT.

**Theorem**

*Unless P=NP, for all \( \epsilon > 0 \), there's no \((8/7 - \epsilon)\) approx. for Max-3-SAT.*

**Corollary**

*Unless P=NP, for all \( \epsilon > 0 \), there's no \((8/7 - \epsilon)\) approx. for CLIQUE.*

(What’s an easy randomized 8/7-approximation for Max-3-SAT?)
Summary of Approximation Algorithms

- Algorithms:
  - 2-approximation for vertex cover
  - 2-approximation for max-cut
  - $3/2$-approximation for metric traveling salesperson
  - $O(\log n)$-approximation for weighted set-cover
  - FPTAS for knapsack

- A poly-time reduction may not be “approximation preserving”

- For a reference of what approximation factors are known check out:
  [http://www.csc.kth.se/~viggo/wwwcompendium/]
Alternative Approaches to NP-hard problems

- Restrict the input:
  - Finding a clique in graph that is acyclic, of bounded degree, or planar
  - Solving metric TSP where the points are in Euclidean space
- Assume a probability distribution over input: *Average case analysis*
- Assume all integers in the input are polynomial in the input size...

**Definition**
An algorithm runs in *pseudo-polynomial time* if the running time is polynomial in the input size and any integer in the input.

**Definition**
A problem is *strongly NP-complete* if it remains NP-complete even when all integers in an input of length $n$ are polynomial in $n$
Outline

More Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness

Approximation Algorithms

Linear Programming
Outline

More Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness

Approximation Algorithms

Linear Programming
Divide and Conquer Methodology

- Goal: Solve problem $P$ on an instance $I$ of “size” $n$.
- Divide & Conquer Method:
  - Transform $I$ into smaller instances $I_1, \ldots, I_a$ each of “size” $n/b$
  - Solve problem $P$ on each of $I_1, \ldots, I_a$ by recursion
  - Combine the solutions to get a solution of $I$
- Examples: Merge Sort, Strassen’s Algorithm, Minimum Distance, Fourier Transform.
Analyzing Divide and Conquer Algorithms

Let $T(n)$ be running time of algorithm on instance of size $n$. Then

$$T(1) = \Theta(1), \quad T(n) = aT(n/b) + \Theta(n^\alpha)$$

where $\Theta(n^\alpha)$ is time to make new instances and combine solutions.

**Theorem (Master Theorem)**

*If $a, b, \alpha$ are constants, for $\beta = \log_b a$,*

$$T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \alpha > \beta \\ \Theta(n^\beta) & \text{if } \alpha < \beta \\ \Theta(n^\alpha \log n) & \text{if } \alpha = \beta \end{cases}$$
Outline

More Approximation Algorithms
Divide and Conquer

**Greedy Algorithms**

Dynamic Programming and Shortest Paths
Network Flows
Randomized Algorithms
NP Completeness
Approximation Algorithms
Linear Programming
Generic Problem and Greedy Algorithms

Definition

A subset system $S = (E, \mathcal{I})$ is a finite set $E$ with a collection $\mathcal{I}$ of subsets $E$ such that:

$$\text{if } i \in \mathcal{I} \text{ and } i' \subset i \text{ then } i' \in \mathcal{I}$$

i.e., “$\mathcal{I}$ is closed under inclusion”

Problem Given a subset system $S = (E, \mathcal{I})$ and weight function $w : E \to \mathbb{R}^+$, find $i \in \mathcal{I}$ such that $w(i) = \sum_{e \in i} w(e)$ is maximized.

Algorithm (Greedy)

1. $i = \emptyset$
2. Sort elements of $E$ by non-increasing weight
3. For each $e \in E$: If $i + e \in \mathcal{I}$ then $i = i + e$
Matroid Definition and Theorem

Definition
A matroid is a subset system \( M = (E, \mathcal{I}) \) that satisfies the exchange property: if \( i, i' \in \mathcal{I} \) such that \( |i| < |i'| \), then there exists \( e \in i' - i \) with \( i + e \in \mathcal{I} \)

Theorem
For any subset system \( (E, \mathcal{I}) \), the greedy algorithm solves the optimization problem for \( (E, \mathcal{I}) \) if and only if \( (E, \mathcal{I}) \) is a matroid.

- A matroid can also be characterized by the cardinality theorem.
- Maximum bipartite matching can be expressed as intersection of two matroids and can therefore be solved in polynomial time.
- Solving the intersection of three matroids becomes NP-hard.
Outline

More Approximation Algorithms
Divide and Conquer
Greedy Algorithms

**Dynamic Programming and Shortest Paths**

Network Flows
Randomized Algorithms
NP Completeness
Approximation Algorithms
Linear Programming
Dynamic Programming and Shortest Paths

When to use dynamic programming...

- **Optimal Substructure**: The solution to the problem can be found using solutions to smaller sub-problems.
- **Overlap of Sub-Problems**: By taking advantage of the fact that many identical sub-problems are created, a dynamic programming algorithm may be more efficient than a divide and conquer algorithm.

Shortest path algorithms...

- **Floyd-Warshall Algorithm**: $O(|V|^3)$
- **Dijkstra’s Algorithm**: Positive weights! $O(|E| + |V| \log |V|)$.
- **Seidel’s Algorithm**: Unweighted Graphs! $O(|V|^{2.38})$ running time.
Outline

More Approximation Algorithms
Divide and Conquer
Greedy Algorithms
Dynamic Programming and Shortest Paths

Network Flows
Randomized Algorithms
NP Completeness
Approximation Algorithms
Linear Programming
Definitions

Input:
▶ Directed Graph $G = (V, E)$
▶ Capacities $C(u, v) > 0$ for $(u, v) \in E$ and $C(u, v) = 0$ for $(u, v) \notin E$
▶ A source node $s$, and sink node $t$

Output: A flow $f$ from $s$ to $t$ where $f : V \times V \to \mathbb{R}$ satisfies
▶ Skew-symmetry: $\forall u, v \in V, f(u, v) = -f(v, u)$
▶ Conservation of Flow: $\forall v \in V - \{s, t\}, \sum_{u \in V} f(u, v) = 0$
▶ Capacity Constraints: $\forall u, v \in V, f(u, v) \leq C(u, v)$

Goal: Maximize “size of the flow”, i.e., the total flow coming leaving $s$:

$$|f| = \sum_{v \in V} f(s, v)$$
Capacity/Flow

Graph representation:

- Nodes: s, v1, v2, v3, v4, t
- Edges and capacities:
  - s to v1: 16/11
  - v1 to v2: 12/12
  - v1 to v3: 10/0
  - v2 to v4: 20/15
  - v3 to v2: 9/4
  - v3 to v4: 7/7
  - v4 to t: 4/4
  - Other edges: 13/8, 14/11

Flow values are indicated on the edges.
Cut Definitions

Definition
An \( s - t \) cut of \( G \) is a partition of the vertices into two sets \( A \) and \( B \) such that \( s \in A \) and \( t \in B \).

Definition
The capacity of a cut \((A, B)\) is \( C(A, B) = \sum_{u \in A, v \in B} C(u, v) \)

Definition
The flow across a cut \((A, B)\) is \( f(A, B) = \sum_{u \in A, v \in B} f(u, v) \)

Theorem (Max-Flow Min-Cut)
For any flow network and flow \( f \), the following statements are equivalent:
1. \( f \) is a maximum flow.
2. There exists an \( s - t \) cut \((A, B)\) such that \( |f| = C(A, B) \)

Went over Ford-Fulkerson Algorithm with Edmonds-Karp Heuristic to find max-flow.
Outline

More Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness

Approximation Algorithms

Linear Programming
Probability and Examples

- For arbitrary events $A$ and $B$,

$$\mathbb{P}[A \text{ and } B] = \mathbb{P}[A \text{ given } B] \mathbb{P}[B]$$

and $A$ and $B$ are independent if $\mathbb{P}[A \text{ and } B] = \mathbb{P}[A] \mathbb{P}[B]$.
- Union Bound: $\mathbb{P}[A \text{ or } B] \leq \mathbb{P}[A] + \mathbb{P}[B]$
- Expectation: $\mathbb{E}[X] = \sum_r r \mathbb{P}[X = r]$
- Linearity of expectation: $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
- Variance random variable: $\mathbb{V}[X] = \sigma_X^2 = \mathbb{E}[(X - \mathbb{E}[X])^2]$
- Linearity of variance if $X$ and $Y$ are independent:

$$\mathbb{V}[X + Y] = \mathbb{V}[X] + \mathbb{V}[Y]$$

Examples: Quicksort, Karger’s Randomized Min-Cut Algorithm, Schwartz-Zippel, Lazy Select, Balls and Bins...
Tail Bounds

**Theorem (Markov)**

Let $Y$ be a non-negative random variable and let $\mu_Y = \mathbb{E}[Y]$. Then, for all $t > 0$, $\mathbb{P}[Y \geq t\mu_Y] \leq 1/t$.

**Theorem (Chebyshev)**

Let $X$ be a random variable with expectation $\mu_X$ and standard deviation $\sigma_X$. Then for $t > 0$, $\mathbb{P}[|X - \mu_X| \geq t\sigma_X] \leq 1/t^2$.

**Theorem**

Let $X_1, \ldots, X_n$ be independent boolean random variables such that $\mathbb{P}[X_i = 1] = p_i$. Then, for $X = \sum_i X_i$, $\mu = \mathbb{E}[X]$, and $\delta > 0$,

$$
\mathbb{P}[X > (1 + \delta)\mu] < \left[ \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right]^\mu
$$
Outline

More Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness

Approximation Algorithms

Linear Programming
NP Completeness

1. \( P \): Problems for which there exists a poly-time algorithm
2. \( NP \): Problems for which there exists a poly-time algorithm taking advice:
   - If the answer should be “yes”, then there exists advice that leads the algorithm to output “yes”
   - If the answer is “no”, then there doesn’t exist advice that would lead the algorithm to output “yes”
3. A problem \( \Pi \) is NP-hard if for any \( \Pi' \in NP: \Pi' \leq_P \Pi \)
4. A problem \( \Pi \) is NP-complete if \( \Pi \in NP \) and \( \Pi \) is NP-hard

Theorem

*Clique, vertex cover, subset-sum etc. are NP-Complete.*
Outline

More Approximation Algorithms
Divide and Conquer
Greedy Algorithms
Dynamic Programming and Shortest Paths
Network Flows
Randomized Algorithms
NP Completeness

Approximation Algorithms
Linear Programming
Approximation Ratios

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 \(x\).

Definition

A problem has a PTAS iff for all \(\epsilon > 0\) it has a poly time \((1 + \epsilon)\) approx.

A problem has a FPTAS iff for all \(\epsilon > 0\) it has \((1 + \epsilon)\) approx where the run time is poly in \(1/\epsilon\) and poly in the size of the input.

Examples: 2 approx for vertex cover, 2 approx for max-cut, 1.5 approx for metric TSP, \(O(\log n)\)-approx for weighted set-cover
Outline

More Approximation Algorithms
Divide and Conquer
Greedy Algorithms
Dynamic Programming and Shortest Paths
Network Flows
Randomized Algorithms
NP Completeness
Approximation Algorithms
Linear Programming
## Linear Programming

### Primal and Dual Linear Programs:

<table>
<thead>
<tr>
<th>Primal LP</th>
<th>Dual LP</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \max c^T x )</td>
<td>( \min y^T b )</td>
</tr>
<tr>
<td>( Ax \leq b )</td>
<td>( y^T A \geq c^T )</td>
</tr>
<tr>
<td>( x \geq 0 )</td>
<td>( y \geq 0 )</td>
</tr>
</tbody>
</table>

### Theorem

Let \( \text{OPT}_{\text{primal}} \) be optimal solution of Primal LP and let \( \text{OPT}_{\text{dual}} \) be optimal solution of Dual LP: If both are bounded and feasible,

\[
\text{OPT}_{\text{primal}} = \text{OPT}_{\text{dual}}
\]

and hence, any feasible solution of the dual LP upper bounds \( \text{OPT}_{\text{primal}} \).

Can be solved in polynomial time but adding integral constraints makes the problem NP-hard.
And finally...

Good luck with the exam!