Outline

Linear Programs

Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness
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

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

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

Aside: When the graph is bipartite, something magical happens: the optimal solution will automatically be integral.
LP Relaxation

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

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

$$
x_v \leq 1 \quad \text{for all } v \in V
$$

$$
x_v \geq 0 \quad \text{for all } v \in V
$$

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.
Linear Programming: Review

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 $\text{OPT}_{\text{primal}}$.

Applications of duality include a) max flow equals min cut and b) the max matching size equals the min vertex cover size in a bipartite graph.

LPs can be solved in poly-time but adding integral constraints makes the problem NP-hard.
Outline

Linear Programs

Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness
Approximation Ratios

Definition
An algorithm for a minimization problem is an $\alpha$-approximation if for all instances,

\[
\frac{\text{value returned by the algorithm}}{\text{optimal value}} \leq \alpha.
\]

For a maximization problem, we want the reciprocal to be at most $\alpha$.

Examples:
- 2-approx for max-cut (local search technique)
- 3/2-approx for metric traveling salesperson
- 2-approx for metric $k$-center clustering (in homework)
- $O(\log n)$-approx for weighted set-cover (charging technique)
- 2-approx for vertex cover (LP relaxation technique)

A reference of what approximation factors are known check out:

http://www.csc.kth.se/~viggo/wwwcompendium/
One Final Approximation Technique: Approximate Input

Definition
A problem has a fully polynomial time approximation scheme (FPTAS) if and only if for all $\epsilon > 0$ it has $(1 + \epsilon)$ approximation where the run time is polynomial in $1/\epsilon$ and polynomial in the size of the input.

General Knapsack Problem:
1. **Input:** A set of items numbered 1, 2, \ldots, $n$, where each the $i$-th item has weight $w_i$ and value $v_i$. $C$ is the capacity of your knapsack.
2. **Goal:** Find a subset $B$ of the items with maximum total value subject to $\sum_{i \in B} w_i \leq C$.

Rough idea for a FPTAS. There’s a dynamic program that solves it exactly in $O(n^2 V)$ where $V = \max_i v_i$. This would be polynomial if $V = \text{poly}(n)$. Scale down the values, e.g.,

$v_1 = 101, v_2 = 93, v_3 = 124 \ldots \rightarrow v_1' = 10, v_2' = 9, v_3' = 12 \ldots$

If we scale at the appropriate precision, solving the problem with the new values gives a good approximation in polynomial time.
Outline

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

Linear Programs

Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness
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.

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

Linear Programs
Approximation Algorithms
Divide and Conquer

Greedy Algorithms
Dynamic Programming and Shortest Paths
Network Flows
Randomized Algorithms
NP Completeness
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:

$$\text{if } B \in \mathcal{I} \text{ and } A \subseteq B \text{ then } A \in \mathcal{I}$$

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 $A \in \mathcal{I}$ such that $w(A) = \sum_{e \in A} w(e)$ is maximized.

Algorithm (Greedy)

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

Definition
A matroid is a subset system \((E, \mathcal{I})\) that satisfies the exchange property: if \(A, B \in \mathcal{I}\) such that \(|A| < |B|\), then \(A + e \in \mathcal{I}\) for some \(e \in B \setminus A\).

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

Linear Programs

Approximation Algorithms

Divide and Conquer

Greedy Algorithms

Dynamic Programming and Shortest Paths

Network Flows

Randomized Algorithms

NP Completeness
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

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

- Nodes: s, v₁, v₂, v₃, v₄, t
- Edges:
  - s to v₁: 16/11
  - s to v₃: 13/8
  - v₁ to v₂: 12/12
  - v₁ to v₃: 10/0
  - v₁ to v₄: 4/1
  - v₂ to t: 20/15
  - v₃ to v₂: 9/4
  - v₃ to v₄: 7/7
  - v₄ to t: 4/4
  - v₃ to v₁: 14/11
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.
Probability and Examples

- For arbitrary events $A$ and $B$,
  \[ P[A \text{ and } B] = P[A \text{ given } B] P[B] \]
  and $A$ and $B$ are independent if $P[A \text{ and } B] = P[A] P[B]$.

- Union Bound: $P[A \text{ or } B] \leq P[A] + P[B]$.

- Expectation: $E[X] = \sum r P[X = r]$.


- Variance random variable: $\mathbb{V}[X] = \sigma_X^2 = E[(X - E[X])^2]$.

- 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, Graph Sparsification, Count-Min Sketch...
Tail Bounds

Theorem (Markov)
Let \( Y \) be a non-negative random variable. Then, for any \( t > 0 \),
\[
\Pr [ Y \geq tE(X) ] \leq 1/t .
\]

Theorem (Chebyshev)
Let \( X \) be any random variable. Then, for any \( t > 0 \),
\[
\Pr [ |X - E(X)| \geq t ] \leq \text{Var}(X)/t^2 .
\]

Theorem
Let \( X_1, \ldots, X_n \) be independent boolean random variables and \( X = \sum_i X_i \). Then for any \( \delta > 0 \),
\[
\Pr [ X > (1 + \delta)\mu ] < e^{-\delta^2 \mu / 3} \quad \text{and} \quad \Pr [ X < (1 - \delta)\mu ] < e^{-\delta^2 \mu / 2}.
\]
Outline

Linear Programs
Approximation Algorithms
Divide and Conquer
Greedy Algorithms
Dynamic Programming and Shortest Paths
Network Flows
Randomized Algorithms

NP Completeness
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 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
3-SAT, CLIQUE, VERTEX-COVER, INDEPENDENT-SET etc. are NP-Complete.

Can sometimes show that a problem is hard to approximate within a certain factor. For example, in the homework question about picking TA you essentially showed that beating a factor 2 approximation for the problem would solve DOMINATING-SET.
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. So an instance of $(G, k)$ of INDEPENDENT-SET is a “yes” instance iff the instance $(G, n - k)$ of VERTEX-COVER is a “yes” instance. □

But using a factor 2-approximation for Vertex-Cover may give a factor $\Omega(n)$ approximation for Independent-Set.
And finally... 

Good luck with the exam!