Lecture 7
Tuesday, 4/6/10

Approximation Algorithms
Approximation Algorithms

Chapter 35
Motivation: Some Techniques for Treating NP-Complete Problems

- **Approximation Algorithms**
- **Heuristic Upper or Lower Bounds**
  - Greedy, Simulated Annealing, Genetic “Alg”, AI
- **Mathematical Programming**
  - Linear Programming for part of problem
  - Integer Programming
  - Quadratic, Convex Programming...
- **Search Space Exploration:**
  - Gradient Descent, Local Search, Pruning, Subdivision
- **Randomization, Derandomization**
- **Leverage/Impose Problem Structure**
- **Leverage Similar Problems**
Basic Concepts

Definitions
Definitions

• **Approximation Algorithm**
  - produces “near-optimal” solution
  - Algorithm has *approximation ratio* $\rho(n)$ if:
    $$\max \left( \frac{C}{C^*}, \frac{C^*}{C} \right) \leq \rho(n)$$
    
    - $C$ = cost of algorithm’s solution
    - $C^*$ = cost of optimal solution
    - $n$ = number of inputs = size of instance

• **Approximation Scheme**
  - $(1+\varepsilon)$-approximation algorithm for fixed $\varepsilon$
    - $\varepsilon > 0$ is an input
  - **Polynomial-Time Approximation Scheme**
    - time is polynomial in $n$
  - **Fully Polynomial-Time Approximation Scheme**
    - time is also *polynomial* in $(1/\varepsilon)$
    - constant-factor decrease in $\varepsilon$ causes only constant-factor running-time increase

source: 91.503 textbook Cormen et al.
Resources beyond textbook…

• UMass Lowell course taught by Prof. Jie Wang.
Overview

- VERTEX-COVER
  - Polynomial-time 2-approximation algorithm
- TSP
  - TSP with triangle inequality
    - Polynomial-time 2-approximation algorithm
  - TSP without triangle inequality
    - Negative result on polynomial-time $\rho(n)$-approximation algorithm
- MAX-3-CNF Satisfiability
  - Randomized $\rho(n)$-approximation algorithm
- SET-COVER
  - Polynomial-time $\rho(n)$-approximation algorithm
    - $\rho(n)$ is a logarithmic function of set size
- SUBSET-SUM
  - Fully polynomial-time approximation scheme
Vertex Cover

Polynomial-Time 2-Approximation Algorithm
**Vertex Cover** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both

**NP-Complete**

(via reduction from CLIQUE in Ch. 34 of Cormen et al.)

[GT1] **VERTEX COVER**

**INSTANCE:** Graph $G = (V, E)$, positive integer $K \leq |V|$.  
**QUESTION:** Is there a vertex cover of size $K$ or less for $G$, i.e., a subset $V' \subseteq V$ with $|V'| \leq K$ such that for each edge $(u, v) \in E$ at least one of $u$ and $v$ belongs to $V'$?

**Reference:** [Karp, 1972]. Transformation from 3SAT (see Chapter 3).  
**Comment:** Equivalent complexity to INDEPENDENT SET with respect to restrictions on $G$. Variation in which the subgraph induced by $V'$ is required to be connected is also NP-complete, even for planar graphs with no vertex degree exceeding 4 [Garey and Johnson, 1977a]. Easily solved in polynomial time if $V'$ is required to be both a vertex cover and an independent set for $G$. The related EDGE COVER problem, in which one wants the smallest set $E' \subseteq E$ such that every $v \in V$ belongs to at least one $e \in E'$, can be solved in polynomial time by graph matching (e.g., see [Lawler, 1976a]).

source: Garey & Johnson
**Approximate Vertex Cover**

**Approximate Vertex Cover**

1. $C \leftarrow \emptyset$
2. $E' \leftarrow E[G]$
3. **while** $E' \neq \emptyset$
4. **do** let $(u, v)$ be an arbitrary edge of $E'$
   5. \[ C \leftarrow C \cup \{u, v\} \]
6. **remove** from $E'$ every edge incident on either $u$ or $v$
7. **return** $C$ (vertex cover)

**Figure 35.1** The operation of **Approximate Vertex Cover**. (a) The input graph $G$, which has 7 vertices and 8 edges. (b) The edge $(b, c)$, shown heavy, is the first edge chosen by **Approximate Vertex Cover**. Vertices $b$ and $c$, shown lightly shaded, are added to the set $C$ containing the vertex cover being created. Edges $(a, b)$, $(c, e)$, and $(c, d)$, shown dashed, are removed since they are now covered by some vertex in $C$. (c) Edge $(e, f)$ is chosen; vertices $e$ and $f$ are added to $C$. (d) Edge $(d, g)$ is chosen; vertices $d$ and $g$ are added to $C$. (e) The set $C$, which is the vertex cover produced by **Approximate Vertex Cover**, contains the six vertices $b, c, d, e, f, g$. (f) The optimal vertex cover for this problem contains only three vertices: $b, d,$ and $e$. 

source: 91.503 textbook Cormen et al.
**Theorem:** APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm.

**Proof:** Let $C^*$ be an optimal cover

$C$ be cover from APPROX - VERTEX - COVER

$A \leftarrow$ edges chosen by APPROX - VERTEX - COVER

**Observe:** no 2 edges of $A$ share any vertices in $C$ due to removal of incident edges

Any vertex cover must include $\geq 1$ endpoint of each edge in $A$ \[ |C^*| \geq |A| \]

APPROX-VERTEX-COVER adds both endpoints of each edge of $A$ \[ |C| = 2|A| \leq 2|C^*| \]

transitivity \[ |C| \leq 2|C^*| \] \[ |C| / |C^*| \leq 2 \]

Algorithm runs in time polynomial in $n$. 
Traveling Salesman

**TSP with triangle inequality**
Polynomial-time 2-approximation algorithm

**TSP without triangle inequality**
Negative result on polynomial-time $\rho(n)$-approximation algorithm
A Hamiltonian Cycle of an undirected graph $G=(V,E)$ is a simple cycle that contains each vertex in $V$.

**NP-Complete** (via reduction from VERTEX-COVER in Ch. 34 of Cormen et al.)

Schlegl diagram of dodecahedron: 12 sides (one of Platonic solids)

Figure 34.2 (a) A graph representing the vertices, edges, and faces of a dodecahedron, with a Hamiltonian cycle shown by shaded edges. (b) A bipartite graph with an odd number of vertices. Any such graph is non-Hamiltonian.
Traveling Salesman Problem (TSP)

**TSP Tour** of a complete, undirected, **weighted** graph \(G=(V,E)\) is a Hamiltonian Cycle with a designated starting/ending vertex.

**TSP Decision Problem:**

\[\{(G, c, k) : \text{cost } c(V \times V) \rightarrow Z \land k \in Z \land G \text{ has TSP - tour of cost } \leq k\}\]

**NP-Complete** (via reduction from HAM-CYCLE in Ch. 34 of Cormen et al.)
**Minimum Spanning Tree: Greedy Algorithms**

**Time:**
- $O(|E| \log |E|)$ given fast FIND-SET, UNION

**Invariant:** Minimum weight spanning forest

**MST-Kruskal** ($G, w$)

1. $A \leftarrow \emptyset$
2. for each vertex $v \in V[G]$
3.   do **MAKE-SET**($v$)
4. sort the edges of $E$ by nondecreasing weight $w$
5. for each edge $(u, v) \in E$, in order by nondecreasing weight
6.   do if **FIND-SET**($u$) $\neq$ **FIND-SET**($v$)
7.     then $A \leftarrow A \cup \{(u, v)\}$
8.     **UNION**($u,v$)
9. return $A$

**Produces minimum weight tree of edges that includes every vertex.**

**Become single tree at end**

**Time:**
- $O(|E| \log |V|) = O(|E| \log |E|)$ slightly faster with fast priority queue

**Invariant:** Minimum weight tree

**MST-Prim** ($G, w, r$)

1. $Q \leftarrow V[G]$
2. for each $u \in Q$
3.   do $key[u] \leftarrow \infty$
4. $key[r] \leftarrow 0$
5. $\pi[r] \leftarrow \text{NIL}$
6. while $Q \neq \emptyset$
7.   do $u \leftarrow \text{EXTRACT-MIN}(Q)$
8.     for each $v \in Adj[u]$
9.     do if $v \in Q$ and $w(u, v) < key[v]$
10.    then $\pi[v] \leftarrow u$
11.   $key[v] \leftarrow w(u, v)$

**Spans all vertices at end**

for Undirected, Connected, Weighted Graph $G=(V,E)$

source: 91.503 textbook Cormen et al.
TSP with Triangle Inequality

**Approx-TSP-Tour** \((G, c)\)

1. Select a vertex \(r \in V[G]\) to be a “root” vertex.
2. Compute a minimum spanning tree \(T\) for \(G\) from root \(r\) using \(\text{MST-PRIM}(G, c, r)\) (Why Prim?)
3. Let \(L\) be the list of vertices visited in a preorder tree walk of \(T\).
4. Return the Hamiltonian cycle \(H\) that visits the vertices in the order \(L\).

**Cost Function Satisfies Triangle Inequality**

\[ \forall u, v, w \in V \quad c(u, w) \leq c(u, v) + c(v, w) \]

**Final approximate tour** (removes vertex duplication) vs. **optimal tour** (not necessarily found by approximation algorithm).

**“full walk” = abcbbadefegeda**
Theorem: APPROX-TSP-TOUR is a polynomial-time 2-approximation algorithm for TSP with triangle inequality.

Proof: Algorithm runs in time polynomial in \( n = |V| \).

Let \( H^* \) be an optimal tour and \( T \) be a MST
\[
c(T) \leq c(H^*) \quad \text{(since deleting 1 edge from a tour creates a spanning tree)}
\]

Let \( W \) be a full walk of \( T \) (lists vertices when they are first visited and when returned to after visiting subtree)
\[
c(W) = 2c(T) \quad \text{(since full walk traverses each edge of \( T \) twice)}
\]
\[
c(W) \leq 2c(H^*)
\]

Now make \( W \) into a tour \( H \) using triangle inequality.
\[
c(H) \leq c(W) \quad \text{(New inequality holds since \( H \) is formed by deleting duplicate vertices from \( W \))}
\]
\[
c(H) \leq 2c(H^*)
\]
**Theorem:** If \( P \neq NP \), then for any constant \( \rho \geq 1 \), there is no polynomial-time approximation algorithm with ratio \( \rho \) for TSP without triangle inequality.

**Proof:** (by contradiction) Suppose there is one --- call it \( A \).

Showing how to use \( A \) to solve NP-complete Hamiltonian Cycle problem \( \rightarrow \) contradiction!

Convert instance \( G \) of Hamiltonian Cycle into instance of TSP (in polynomial time):

\[
E' = \{(u, v) : u, v \in V, u \neq v\} \quad (G'=(V,E') \text{ is complete graph on } V)
\]

\[
c(u,v) = \begin{cases} 
1 & (u,v) \in E \\
\rho |V| + 1 & \text{otherwise}
\end{cases}
\]

(assign integer cost to each edge in \( E' \))

For TSP problem \( (G',c) \):

- \( G \) has Hamiltonian Cycle \( \rightarrow (G',c) \) contains tour of cost \( |V| \)
- \( G \) does not have Hamiltonian Cycle \( \rightarrow \) Tour of \( G' \) must use some edge not in \( E \)

Cost of that tour of \( G' \) \( \geq (\rho |V| + 1) + (|V| - 1) = \rho |V| + |V| > \rho |V| \)

Can use \( A \) on \( (G',c) \)! A finds tour of cost at most \( \rho \) (length of optimal tour of \( G' \))

- \( G \) has Hamiltonian Cycle \( \rightarrow A \) finds tour of cost at most \( \rho |V| \) \( \rightarrow \) cost = \( |V| \)
- \( G \) does not have Hamiltonian Cycle \( \rightarrow A \) finds tour of cost \( > \rho |V| \)

*If \( P \neq NP \), solving NP-complete Hamiltonian cycle in polynomial time is a contradiction!*
MAX-3-CNF Satisfiability

3-CNF Satisfiability Background
Randomized Algorithms
Randomized MAX-3-CNF SAT Approximation Algorithm
Background on Boolean Formula Satisfiability

- Boolean Formula Satisfiability: Instance of language SAT is a boolean formula $\phi$ consisting of:
  - $n$ boolean variables: $x_1, x_2, \ldots, x_n$
  - $m$ boolean connectives: boolean function with 1 or 2 inputs and 1 output
    - e.g. AND, OR, NOT, implication, iff
    - truth, satisfying assignments notions apply

$SAT = \{ \langle \phi \rangle : \phi$ is a satisfiable boolean formula$\}$

NP-Complete (via reduction from CIRCUIT-SAT in Ch. 34 of Cormen et al.)
MAX-3-CNF Satisfiability
(continued)

• Background on 3-CNF-Satisfiability
  • Instance of language SAT is a boolean formula
    \( \phi \) consisting of:
      • literal: variable or its negation
      • CNF = conjunctive normal form
        • conjunction: AND of clauses
        • clause: OR of literal(s)
    • 3-CNF: each clause has **exactly 3** distinct literals

MAX-3-CNF Satisfiability: optimization version of 3-CNF-SAT
  - Maximization: satisfy as many clauses as possible
  - Input Restrictions:
    - exactly 3 literals/clause
    - no clause contains both variable and its negation (this assumption can be removed)

NP-Complete (via reduction from SAT in Ch. 34 of Cormen et al.)

source: 91.503 textbook Cormen et al.
Definition

- Randomized Algorithm has approximation ratio $\rho(n)$ if, for expected cost $C$ of solution produced by randomized algorithm:

$$\max \left( \frac{C}{C^*}, \frac{C^*}{C} \right) \leq \rho(n)$$

size of input = $n$

source: 91.503 textbook Cormen et al.
Randomized Approximation Algorithm for MAX-3-CNF SAT

**Theorem:** Given an instance of MAX-3-CNF satisfiability with $n$ variables $x_1, x_2, \ldots, x_n$ with $m$ clauses, the randomized algorithm that independently sets each variable to 1 with probability $1/2$ and to 0 with probability $1/2$ is a randomized $8/7$-approximation algorithm.

**Proof:** Independently set each variable to 0 or 1 with probability $1/2$ indicator random variable $Y_i = I\{\text{event that clause } i \text{ is satisfied}\}$

$Pr\{\text{clause } i \text{ is not satisfied}\} = (1/2)^3 = 1/8 \quad Pr\{\text{clause } i \text{ is satisfied}\} = 1 - (1/8) = 7/8$

Lemma: Given a sample space $S$ and an event $A$ in the sample space $S$,

Let $X_A = I\{A\}$. Then $E[X_A] = Pr\{A\}$.

By Lemma: $E[Y_i] = 7/8 \quad$ Let $Y = Y_1 + Y_2 + \cdots + Y_m$

$E[Y] = E\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} \frac{7}{8} = \frac{7m}{8} = \text{expected cost } C$

number of satisfied clauses $\leq m \Rightarrow C^* \leq m \Rightarrow \frac{C^*}{C} \leq \frac{m}{7m/8} = \frac{8}{7} = \rho(n)$
Set Cover

Greedy Approximation Algorithm

polynomial-time $\rho(n)$-approximation algorithm

$\rho(n)$ is a logarithmic function of set size
Set Cover Problem

Instance \((X, F)\):

- finite set \(X\) (e.g. of points)
- family \(F\) of subsets of \(X\)

\[ X = \bigcup_{S \in F} S \]

Problem: Find a minimum-sized subset \(C \subseteq F\) whose members cover all of \(X\):

\[ X = \bigcup_{S \in C} S \]

**NP-Complete** (via reduction from VERTEX-COVER as noted in Ch. 35 of Cormen et al.)

Source: 91.503 textbook Cormen et al.
Greedy Set Covering Algorithm

**Greedy-Set-Cover**$(X, F)$

1. $U \leftarrow X$
2. $C \leftarrow \emptyset$
3. while $U \neq \emptyset$
   4. do select an $S \in F$ that maximizes $|S \cap U|$  
   5. $U \leftarrow U - S$
   6. $C \leftarrow C \cup \{S\}$
4. return $C$

*Greedy*: select set that covers the most uncovered elements
**Theorem:** GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm for $\rho(n) = H(\max \{|S| : S \in F\})$

**Proof:**

The $d$th harmonic number $H_d = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$. Algorithm runs in time polynomial in $n$.

$S_i = i$th subset selected by algorithm.

selecting $S_i$ costs 1

$S_i = i$th subset selected by algorithm.

(cnotational caveat)

$\mathcal{c}_x = \text{cost of element } x \in X$

paid only when $x$ is covered for the first time

$\mathcal{c}_x = \frac{1}{\left| S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1}) \right|}$

(elements already covered by first $i-1$ chosen sets)

assume $x$ is covered for the first time by $S_i$

(spread cost evenly across all elements covered for first time by $S_i$)

Number of elements covered for first time by $S_i$
**Theorem:** \textsc{Greedy-Set-Cover} is a polynomial-time $\rho(n)$-approximation algorithm for $\rho(n) = H(\max\{|S|: S \in \mathcal{F}\})$.

**Proof:** (continued)

Let $C^*$ be an optimal cover,\hfill $|C| = \sum_{x \in X} c_x$ \hspace{2cm} 1 unit is charged at each stage of algorithm

$C$ be cover from \textsc{Greedy - Set - Cover}.\hfill \sum\sum_{S \in C^*} \left( \sum_{x \in S} c_x \right)$ \hspace{2cm} Each $x$ is in $\geq 1$ $S$ in $C^*$ \hfill \sum\sum_{S \in C^*} \left( \sum_{x \in S} c_x \right) \geq \sum_{x \in X} c_x$

Cost assigned to optimal cover:

\[|C| \leq \sum\sum_{S \in C^* \setminus x \in S} c_x\]
Theorem: GREEDY-SET-COVER is a polynomial-time $\rho(n)$-approximation algorithm for $\rho(n) = H(\max \{|S| : S \in F\})$

Proof: (continued)

How does this relate to harmonic numbers??

$d$th harmonic number $H_d = \sum_{i=1}^{d} \frac{1}{i} = H(d)$

We’ll show that: $\sum_{x \in S} c_x \leq H(|S|)$ for any set $S \in F$

And then conclude that: $|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*| H(\max \{|S| : S \in F\})$
Set Cover (proof continued)

Proof of: \[ \sum_{x \in S} c_x \leq H(|S|) \]
for any set \( S \in F \)

\[ u_i = |S - (S_1 \cup S_2 \cup \cdots \cup S_i)| \]
For some set \( S \): Number of elements of \( S \) remaining uncovered after \( S_1 \ldots S_i \) selected

\[ \sum_{x \in S} c_x = \sum_{i=1}^k \left( (u_{i-1} - u_i) \frac{1}{S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})} \right) \]
\( k = \) least index for which \( u_k = 0 \).

\[ \sum_{x \in S} c_x \leq \sum_{i=1}^k \left( (u_{i-1} - u_i) \frac{1}{u_{i-1}} \right) \]
Since, due to greedy nature of algorithm:

\[ |S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \geq |S - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| = u_{i-1} \]

\[ = \sum_{i=1}^k \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}} \leq \sum_{i=1}^k \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \]
since \( j \leq u_{i-1} \)

\[ = \sum_{i=1}^k \left( \sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) \]

\[ = \sum_{i=1}^k \left( H(u_{i-1}) - H(u_i) \right) = H(u_0) - H(u_k) = H(u_0) - H(0) = H(u_0) = H(|S|) \]
telelescoping sum
Subset-Sum

Exponential-Time Exact Algorithm

Fully Polynomial-Time Approximation Scheme
Subset-Sum Problem

Instance \((S, t)\):

\[ S = \{x_1, x_2, \ldots, x_n\} \quad \text{positive integers} \]

Decision Problem:

\[ \exists S' \subseteq S : \sum_{s \in S'} s = t ? \]

**Optimization Problem** seeks subset with largest sum \( \leq t \)

NP-Complete (via reduction from 3-CNF-SAT in Ch. 34 of Cormen et al.)

source: 91.503 textbook Cormen et al.
Exponential-Time
Exact Algorithm

\[ P_i = \text{set of values obtainable by selecting subset of } \{x_1, x_2, \ldots, x_i\} \text{ and summing elements.} \]

\[ L_i \text{ is sorted list of every element of } P_i \leq t \]

\[ L_i = \text{list of sums of subsets of } \{x_1, x_2, \ldots, x_i\} \leq t \]

**MERGE-LISTS** (\( L, L' \)) returns sorted list = merge of sorted \( L, L' \) with duplicates removed.

**EXACT-SUBSET-SUM** (\( S, t \))

1. \( n \leftarrow |S| \)
2. \( L_0 \leftarrow \langle () \rangle \)
3. \( \text{for } i \leftarrow 1 \text{ to } n \)
4. \( \text{do } L_i \leftarrow \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i) \)
5. \( \text{remove from } L_i \text{ every element that is greater than } t \)
6. \( \text{return the largest element in } L_n \)

**Identity:**

\[ P_i = P_{i-1} \cup (P_{i-1} + x_i) \]
**Theorem:**

APPROX-SUBSET-SUM is a fully polynomial-time approximation scheme for subset-sum.

**Proof:** see textbook

(differs from 2nd edition)

\[ |L_i| \leq \frac{3n \ln t}{\varepsilon} + 2 \]

source: 91.503 textbook Cormen et al.