Approximation and Randomized Algorithms

Lecture 3
28-Jan-2014
Last time

- A small tour of problems that cannot be solved efficiently
  - NP, NP-completeness
- We have started the approximation algorithms topic
  - Load balancing algorithms
  - Center selection problem
Today

Weighted set cover
- greedy algorithm

Weighted vertex cover
- pricing method

Maximum Disjoint Paths problem
- pricing method
Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you're unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to optimality.
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

\( \rho \)-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio \( \rho \) of true optimum.

Challenge. Need to prove a solution's value is close to optimum, *without even knowing what optimum value is!*
Some general techniques for designing approximation algorithms

- *Greedy algorithms and bounds on the optimum*
- Pricing method
- Linear programming and rounding
- Arbitrarily good approximations
  - Dynamic programming on a rounded version of the input
Weighted Set Cover

Weighted set cover. Given a set $U$, $|U|=n$ 
$S_1,\ldots,S_m \subseteq U$

- weight of each $S_i$: $w_i \geq 0$, $i = 1,\ldots,m$
- $C=\{S_{j_1},\ldots,S_{j_k}\}$ cover when $S_{j_1} \cup \ldots \cup S_{j_k} = U$

- **Goal**: find a cover $C$ with minimum weight: \[ \sum_{S_j \in C} w_j \]

- NP-complete problem
The algorithm

Greedy

- builds cover one set at a time
- the set \( S_i \) that makes the most progress towards goal
  - Small weight: \( w_i \)
  - Cover lots of elements: \( |S_i| \)
  - \( \Rightarrow \) Covers \( |S_i| \) elements with price \( w_i \)
  - \( R \)- set of remaining elements
  - We choose set \( S_i \) that minimizes \( \frac{w_i}{|S_i \cap R|} \)
Greedy-Set-Cover

**Greedy Set Cover.** Builds one set at a time, choosing set that minimizes "cost"

```
Greedy-Set_Cover(U,S_i,w_i) {
    R := U
    C := ∅
    While R ≠ ∅ {
        Select S_i that minimizes \( \frac{w_i}{|S_i \cap R|} \)
        R := R \ S_i
        C := C \cup S_i
    }
    Return C
}
```
Set Cover Example
Set Cover Analysis

Greedy-Set_Cover(U,Si,wi) {
R := U
C := ∅
While R≠∅ {
    Select Si that minimizes $\frac{w_i}{|S_i \cap R|}$
    For each $s \in S_i \cap R$
        $c_s := \frac{w_i}{|S_i \cap R|}$
    R := R \ S_i
    C := C ∪ S_i
} 
Return C
}
Harmonic upper bound

Lemma 1. For every set $S_k$:

$$
\sum_{s \in S_k} c_s \leq H(\lfloor \frac{|S_k|}{d} \rfloor) \cdot w_k
$$

Pf. Assume $S_k = \{s_1, ..., s_d\}$. Consider iteration when $s_j$ is covered by algorithm, $j \leq d \Rightarrow s_j, s_{j+i}, ..., s_d \in R$ => $|S_k \cap R| \geq d - j + 1$ =>

$$
\frac{w_k}{|S_k \cap R|} \leq \frac{w_k}{d - j + 1}
$$

We have $c_{s_j} = \frac{w_i}{|S_i \cap R|} \leq \frac{w_k}{|S_k \cap R|} \leq \frac{w_k}{d - j + 1}$

And thus

$$
\sum_{s \in S_k} c_s = \sum_{j=1}^{d} c_{s_j} \leq \sum_{j=1}^{d} \frac{w_k}{d - j + 1} = \frac{w_k}{d} + \frac{w_k}{d - 1} + ... + \frac{w_k}{1} = H(d) \cdot w_k
$$
Approximation factor

Lemma 2. If \( C \) is cover returned by algorithm then

\[
\sum_{S_i \in C} w_i = \sum_{s \in U} c_s
\]

Claim. The set cover \( C \) selected by algorithm has weight at most \( H(d^*) \) times optimal weight \( w^* \).

\( d^* = \max_i |S_i| \)

Pf. \( C^* \) optimum set cover

\[
w^* = \sum_{S_i \in C^*} w_i \geq \sum_{S_i \in C^*} \frac{1}{H(d^*)} \sum_{s \in S_i} c_s \geq \frac{1}{H(d^*)} \sum_{s \in U} c_s = \frac{1}{H(d^*)} \sum_{S_i \in C} w_i
\]

\( \text{def} \)

Lemma 1

\( S_i \) cover

Lemma 2
**Vertex Cover**

$G = (V,E)$ graph;

$S \subseteq V$ is vertex cover if each edge has an end in $S$.

$\forall i \in V: w_i \geq 0$ weight of node $i$

$w(S) = \sum_{i \in S} w_i$ weight of set $S$

**Goal.** Find vertex cover of minimum weight
Polynomial-Time Reduction

• Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

• Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:
  • Polynomial number of standard computational steps, plus
  • Polynomial number of calls to black box that solves problem Y.

• Notation. \( X \leq_P Y \).
Polynomial-Time Reduction

- **Purpose.** Classify problems according to relative difficulty.

- **Design algorithms.** If $X \leq_p Y$ and $Y$ can be solved in polynomial-time, then $X$ can also be solved in polynomial time.

- **Establish intractability.** If $X \leq_p Y$ and $X$ cannot be solved in polynomial-time, then $Y$ cannot be solved in polynomial time.

- **Establish equivalence.** If $X \leq_p Y$ and $Y \leq_p X$, we use notation $X \equiv_p Y$, up to cost of reduction.
Independent Set

- INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in $S$?

- Exp. Is there an independent set of size $\geq 6$? Yes.
- Exp. Is there an independent set of size $\geq 7$? No.
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, at least one of its endpoints is in $S$?

- Ex. Is there a vertex cover of size $\leq 4$? Yes.
- Ex. Is there a vertex cover of size $\leq 3$? No.
Vertex Cover and Independent Set

- **Claim.** $\text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET}$. 

- **Pf.** We show $S$ is an independent set iff $V - S$ is a vertex cover.
Vertex Cover and Independent Set

• **Claim.** $\text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET}$.  

• **Pf.** We show $S$ is an independent set iff $V - S$ is a vertex cover.

• $\Rightarrow$
  • Let $S$ be any independent set.
  • Consider an arbitrary edge $(u, v)$.
  • $S$ independent $\Rightarrow u \notin S$ or $v \notin S$ $\Rightarrow u \in V - S$ or $v \in V - S$.
  • Thus, $V - S$ covers $(u, v)$.

• $\Leftarrow$
  • Let $V - S$ be any vertex cover.
  • Consider two nodes $u \in S$ and $v \in S$.
  • Observe that $(u, v) \notin E$ since $V - S$ is a vertex cover.
  • Thus, no two nodes in $S$ are joined by an edge $\Rightarrow S$ independent set. ▪
Set Cover

- **Set Cover**: Given a set $U$ of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$, does there exist a collection of $\leq k$ of these sets whose union is equal to $U$?

- Sample application.
  - $m$ available pieces of software.
  - Set $U$ of $n$ capabilities that we would like our system to have.
  - The $i$th piece of software provides the set $S_i \subseteq U$ of capabilities.
  - **Goal**: achieve all $n$ capabilities using fewest pieces of software.

- Ex:
  
  $U = \{1, 2, 3, 4, 5, 6, 7\}$  
  $k = 2$  
  $S_1 = \{3, 7\}$  
  $S_4 = \{2, 4\}$  
  $S_2 = \{3, 4, 5, 6\}$  
  $S_5 = \{5\}$  
  $S_3 = \{1\}$  
  $S_6 = \{1, 2, 6, 7\}$
Vertex Cover Reduces to Set Cover

- Claim. VERTEX-COVER \(\leq_p\) SET-COVER.
- Pf. Given a VERTEX-COVER instance \(G = (V, E), k\), we construct a set cover instance whose size equals the size of the vertex cover instance.

- Construction.
  - Create SET-COVER instance:
    - \(k = k, U = E, S_v = \{e \in E : e \text{ incident to } v\}\)
  - Set-cover of size \(\leq k\) iff vertex cover of size \(\leq k\).
Approximations via reductions

We have:

\[\text{Vertex cover} \leq_p \text{Set cover}\]

"if we had poly-time algorithm for SC, then we could use this algorithm to solve VC in poly-time"

For SC we have a $H(d)$-approximation poly-time algorithm
**H(d)-approximation for VC**

**Claim.** We can use the approximation algorithm for SC to give an H(d)-approximation algorithm for VC

(d - maximum degree of graph)

**Pf.** Consider an instance of the weighted VC. We define an instance of the SC:

\[ U := E, \quad \forall i \in V, \quad S_i := \{e | e \text{ incident to } i \}, \quad w_i \]

set cover for \( U \equiv \text{vertex cover} \)

max size of any \( S_i = \text{maximum degree } d \)
Weighted Vertex Cover

Weighted vertex cover. Given a graph $G$ with vertex weights, find a vertex cover of minimum weight.

weight = 2 + 2 + 4
Weighted Vertex Cover

**Pricing method.** Each edge must be covered by some vertex $i$. Edge $e$ pays price $p_e \geq 0$ to use vertex $i$.

**Fairness.** Edges incident to vertex $i$ should pay $\leq w_i$ in total.

For each vertex $i$:

$$\sum_{e=(i,j)} p_e \leq w_i$$

**Claim.** For any vertex cover $S$ and any fair prices $p_e$: $\sum_e p_e \leq w(S)$.

**Pf.**

$$\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S).$$

- each edge $e$ covered by at least one node in $S$
- sum fairness inequalities for each node in $S$
Pricing Method

Pricing method. Set prices and find vertex cover simultaneously.

\[
\text{Weighted-Vertex-Cover-Approx}(G, w) \{ \\
\text{foreach } e \text{ in } E \\
\quad p_e = 0 \\
\quad \text{while } (\exists \text{ edge } i-j \text{ such that neither } i \text{ nor } j \text{ are tight}) \\
\quad \quad \text{select such an edge } e \\
\quad \quad \text{increase } p_e \text{ without violating fairness} \\
\} \\
S \leftarrow \text{set of all tight nodes} \\
\text{return } S
\]
Pricing Method

![Graph Diagrams]

Figure 11.8

- **Price of edge a-b**
- **Vertex weight**

(a) 

(b) 

(c) 

(d)
Pricing Method: Analysis

Claim. Pricing method is a 2-approximation.

Pf.

• Algorithm terminates since at least one new node becomes tight after each iteration of while loop.

• Let $S =$ set of all tight nodes upon termination of algorithm. $S$ is a vertex cover: if some edge $i$-$j$ is uncovered, then neither $i$ nor $j$ is tight. But then while loop would not terminate.

• Let $S^*$ be optimal vertex cover. We show $w(S) \leq 2w(S^*)$. ■

\[
w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e \leq 2w(S^*).
\]
Maximum disjoint paths

Network routing. $G$ directed graph; $(s_i, t_i)$, $i=1,...,k$ routing requests; $c \geq 1$ capacity

A solution. $I \subseteq \{1,...,k\}$ s.t. $\forall i \in I$, $\exists P_i$ that connects $s_i$ to $t_i$ and each edge used at most $c$ times

Our goal. Find $I$ that is maximal

Problem is NP-complete.
Context

Common in network applications
- Find paths connecting designated pairs of terminal nodes

Paths on Internet that carry streaming media, web data

Paths through the phone network, carry voice traffic

Paths sharing edges interfere with each other
- Too many of those => quality problems
- Different $c$ for different applications
- $c=2$ -> better approximation algorithms
Algorithm 1: Greedy

- c=1; greedy, prefers short paths
- \(O(\sqrt{m})\), \(m=|E|\)
- hard to approximate
  - essentially the best factor unless \(P=NP\)

```python
Greedy-Disjoint-Paths(G, (s_i, t_i)) {
    I := ∅
    Until no new path can be found {
        choose \(P_i\) as the shortest path (if one exists) that is edge-disjoint from previously selected paths and connects some \((s_i, t_i)\) pair not yet connected
        I := I \cup \{i\}
        Select path \(P_i\) to connect \(s_i\) to \(t_i\)
    }
    Return I
}
```
Maximum disjoint paths (MDJ): example
MDP Analysis (1/3)

- $G$ connected $\Rightarrow$ at least one path

Notations.

- $I^*$ optimum; $P_{i^*}, i \in I^*$ selected paths
- $I$ returned by algorithm; $P_i, i \in I$ paths
- we define long path if at least $\sqrt{m}$ edges, short otherwise
- $I_{s^*} \subseteq I^*$, $I_s \subseteq I$, sets of indices of short paths
- $|E|=m \Rightarrow$ at most $\sqrt{m}$ long paths in $I^*$ (paths are disjoint)
Lemma 1. There are at most $|I_s| \cdot \sqrt{m}$ short paths that are in $I_s^*$ but not in $I$.

Pf. Consider $P_i^* \in I_s^* \setminus I \Rightarrow$ algorithm did not selected $P_i^*$ because $\exists$ edge, $e \in P_i^*$ s.t. $e \in P_j$, $P_j \in I$, $P_j$ selected earlier (e blocks $P_i^*$)

- $|P_j| \leq |P_i^*| \leq \sqrt{m} \Rightarrow |I_s^* \setminus I| \leq \sum_{j \in I_s} |P_j| \leq |I_s| \cdot \sqrt{m}$
MDP Analysis (3/3)

Claim. The algorithm is a \((2\sqrt{m} + 1)\)-approximation for MDP.

Pf. \(I^*\) has

- Long paths, at most \(\sqrt{m}\)
- Short paths also in \(I\)
- Short paths not in \(I\), at most \(|I_s| \cdot \sqrt{m}\)

Hence, \(|I^*| \leq \sqrt{m} + |I| + |I_s^* - I| \leq \sqrt{m} + |I| + |I_s| \cdot \sqrt{m} \leq (2\sqrt{m} + 1) \cdot |I|\)
Algorithm 2: Pricing

- Analogous to Greedy; \( c > 1 \)
- Greedy
  - all edges equal, short paths preferred
- Pricing
  - Paths have to pay for using up the edges
  - Each edge has unit cost
  - Edges are more expensive if they were already used
    - Less capacity left over
  - Algorithm spreads out the paths
**Cost**

**Notation.**

- $l(e)$ - length of edge $e$ (its cost)
- $l(P) = \sum_{e \in P} l(e)$ where $P$ path
- $\beta$ - parameter, increases the “length” of an edge every time a path uses it
Algorithm 2: pricing MDP

Greedy-Paths-With-Capacity(G,(s_i,t_i)) {
  I := ∅
  For each e ∈ E, l(e) := 1
  Until no new path can be found {
    choose P_i as the shortest path (if one exists) s.t.
    adding P_i to the already selected paths does not use
    any edge more than c times and P_i connects some
    (s_i,t_i) pair not yet connected
    I := I ∪ {i}
    Select path P_i to connect s_i to t_i
    For each e ∈ P_i, l(e) := l(e) * β
  }
  Return I
}
Pricing MDP (1/3)

For analysis, \( c = 2 \)

- \( \beta = m^{1/3} \) gives the best approximation

**Short path.** P path is *short* if \(|P| < \beta^2\)

When measure the length?

- For analysis we measure when there are no more short paths to choose

- \( \ell_a(P) = \sum_{e \in P} \ell_a(e) \)

- \( P_i \) *short* if \( \ell_a(P) < \beta^2 \), *long* otherwise
Pricing MDP (2/3)

Lemma 1. $P_i^* \in I^* \setminus I \Rightarrow \lambda_a(P_i^*) \geq \beta^2$

Pf. While short paths selected, no need to enforce $c$.
- Any edge $e$ chosen by a third path $l(e) = \beta^2 \Rightarrow$ long
- Assume no more short paths.
  $P_i^* \notin I \Rightarrow \lambda_a(P_i^*) \geq \beta^2$

Lemma 2. $\sum_e \lambda_a(e) \leq |I_s|^* \beta^3 + m$

Pf. $\sum_e \lambda_a(e) \leq m + |I_s|^* \beta \cdot \beta^2$
Pricing MDP (3/3)

**Claim.** Greedy-Paths-With-Capacity is $(4m^{1/3} + 1)$-approximation when $c=2$ and $\beta = m^{1/3}$

**Pf.**

- Lemma 1 $\Rightarrow \sum_{i \in I^* \setminus I} la(P_i^*) \geq \beta^2 \cdot |I^* \setminus I|
- $\sum_{i \in I^* \setminus I} la(P_i^*) \leq \sum_{e \in E} 2la(e)$ (each edge used by at most 2 paths in $I^*$)
- $\beta^2 |I^*| \leq \beta^2 |I^* \setminus I| + \beta^2 |I| \leq \sum_{i \in I^* \setminus I} la(P_i^*) + \beta^2 |I| \leq \sum_{e \in E} 2la(e) + \beta^2 |I| \leq 2(\beta^3 |I_s| + m) + \beta^2 |I|

$\Rightarrow |I^*| \leq (4m^{1/3} + 1)|I|$, as $|I| \geq 1$, $\beta = m^{1/3}$
General MDP, $c>0$

**Claim.** Greedy-Paths-With-Capacity is an approximation when $c>0$ and $eta = m^{1/(c+1)}$.

**Pf.** P path is short if $|P| < \beta^c$.
Algorithms from today

Weighted set cover
  • greedy algorithm: $\rho = H(d^*)$

Weighted vertex cover
  • pricing method: $\rho = H(d)$ but also $\rho = 2$

Maximum Disjoint Paths problem
  • pricing method: $\rho = 2c m^{1/(c+1)} + 1$