Lecture 2: *Greedy Approximation Algorithms*

*Advanced Algorithms*

Sid Chi-Kin Chau

Australian National University

sid.chau@anu.edu.au

October 5, 2022
The simplest algorithm is to make the most improvement as much as possible in each step.
Set Cover
Let the objective function of a NP-hard minimization problem be $\text{Cost}(\cdot)$

**Definition (SetCover)**

Consider a set $U$ with $n$ items and a family of subsets (called covers) $\mathcal{K} \subseteq 2^U$

Each cover has a non-negative cost: $\text{Cost}(S)$ for $S \in \mathcal{K}$

Select the a subset of covers $\tilde{\mathcal{K}} \subseteq \mathcal{K}$ such that

- Minimizing the total cost:
  $$\min_{\tilde{\mathcal{K}} \subseteq \mathcal{K}} \sum_{S \in \tilde{\mathcal{K}}} \text{Cost}(S)$$

- Subject to the constraint of covering all items:
  $$\bigcup_{S \in \tilde{\mathcal{K}}} S = U$$
Set Cover Problem

<table>
<thead>
<tr>
<th>Sensor Networks</th>
<th>Covers</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistics</td>
<td>Sensors</td>
<td>Targets of Interest</td>
</tr>
<tr>
<td>Cloud Computing</td>
<td>Service Depots</td>
<td>Clients</td>
</tr>
<tr>
<td>Testing</td>
<td>Cloud Servers</td>
<td>Users</td>
</tr>
<tr>
<td></td>
<td>Tests</td>
<td>Properties</td>
</tr>
</tbody>
</table>
Example (Antivirus Scanner)

- Some special features can be detected in the boot sector of a computer, if a computer virus is present.
- Let the items of SetCover be the known boot sector viruses (∼150 at the time).
- Let each cover be a three-byte sequence in the boot sector, if viruses are present in a computer (∼21,000 such sequences).
- Each cover contains all the boot sector viruses that have the corresponding three-byte sequence detected in the boot sector.
- Goal: Find a minimum number of such sequences (∝150) that are useful for an Antivirus scanner.
Example: DNA Sequencing

DNA sequencing is also a set cover problem
When sequencing DNA, it is not achieved by one sequential operation
Instead, many short segments of DNA sequences can be identified
Need to reconstruct the original long DNA sequence from these short identified segments
Goal: Find the shortest sequence that is a superstring of all short identified segments
What is a good strategy to solve SetCover?

Intuitive approach is to use a greedy algorithm

Find the most improvement in each step. But how?
- Select the cover with the lowest cost?
- Select the cover with the most uncovered items?
- Select the cover with the lowest cost per uncovered item?

Which one of them works?

If not, why doesn’t it work?
How bad is a Greedy Algorithm?

- Select the cover with the lowest cost?

- Select the cover with the most uncovered items?

- Select the cover with the lowest cost per uncovered item?
Better Greedy Algorithm for SetCover

- Let the set of items that are already covered before the $k$-th step be $C_k$
- Define the price for each cover $S \in \mathcal{K}$ by $\text{Price}(S, C_k) \triangleq \frac{\text{Cost}(S)}{|S \setminus C_k|}$

**Algorithm $A_{setcover}$**

- $\tilde{\mathcal{K}} \leftarrow \emptyset; \quad C_1 \leftarrow \emptyset; \quad k \leftarrow 1$
- **While** $C_k \neq \mathcal{U}$
  - Find $S \in \mathcal{K}$ with the least $\text{Price}(S, C_k)$
  - $\tilde{\mathcal{K}} \leftarrow \tilde{\mathcal{K}} \cup \{S\}$
  - $C_{k+1} \leftarrow C_k \cup S$
  - $k \leftarrow k + 1$
- **Return** $\tilde{\mathcal{K}}$
Let $\text{Opt}$ be the optimal cost.

Let $S_k$ be the selected cover by $\mathcal{A}_{\text{setcover}}$ at the $k$-th step.

Let $S^*_k$ be the selected cover by the optimal solution at the $k$-th step.
Better Greedy Algorithm for SetCover

**Theorem**

The approximation ratio of $A_{\text{setcover}}$ is $O(\log(n))$

**Proof:**

- The number of items being not covered at the $k$-th step is $n - |C_k|$
- We show that $\text{Price}(S_k, C_k) \leq \frac{\text{Opt}_k}{n - |C_k|}$
  - $A_{\text{setcover}}$ always selects the least-price cover (i.e. the least cost per uncovered item)
  - Let $\text{Opt}_k$ be the optimal cost on $U \setminus C_k$ and the corresponding optimal covers be $K^*_k$
  - The price of $S_k$ must be lower than the overall price of $\text{Opt}_k$

$$\text{Price}(S_k, C_k) = \frac{\text{Cost}(S_k)}{|S_k \setminus C_k|} \leq \frac{\text{Opt}_k}{n - |C_k|}$$
Better Greedy Algorithm for SetCover

Proof (Cont.):

• Why $\text{Price}(S_k, C_k) \leq \frac{\text{Opt}_k}{n - |C_k|}$?
  
  ▶ Since $\text{Opt}_k = \sum_{S \in K_k^*} |S \setminus C_k| \cdot \text{Price}(S, C_k)$, there always exists $S \in K_k^*$, such that

  $$\text{Price}(S, C_k) \leq \frac{\text{Opt}_k}{\sum_{S \in K_k^*} |S \setminus C_k|}$$

  ▶ Note that $\sum_{S \in K_k^*} |S \setminus C_k| \geq n - |C_k|$. Hence, we obtain $\text{Price}(S_k, C_k) \leq \frac{\text{Opt}_k}{n - |C_k|}$

• Because $\text{Opt}_k \leq \text{Opt}$,

  $$\text{Cost}(S_k) = \text{Price}(S_k, C_k) \cdot (|C_{k+1}| - |C_k|) \leq \frac{|C_{k+1}| - |C_k|}{n - |C_k|} \cdot \text{Opt}_k \leq \frac{|C_{k+1}| - |C_k|}{n - |C_k|} \cdot \text{Opt}$$
**Better Greedy Algorithm for SetCover**

**Proof (Cont.):**

- Because $\text{Cost}(S_k) \leq \frac{|C_{k+1}| - |C_k|}{n - |C_k|} \text{Opt}$,

\[
\text{Cost}(A_{\text{setcover}}) = \sum_{k=1}^{|	ilde{\mathcal{C}}|} \text{Cost}(S_k)
\]

\[
\leq \text{Opt} \cdot \sum_{k=1}^{|	ilde{\mathcal{C}}|} \frac{|C_{k+1}| - |C_k|}{n - |C_k|}
\]

\[
\leq \text{Opt} \cdot \sum_{i=1}^{n} \frac{1}{i} \quad \text{(since } n - |C_k| \geq n - i + 1, \text{ if the } i\text{-th item } \in C_{k+1} \setminus C_k\text{)}
\]

\[
= \text{Opt} \cdot O(\log(n))
\]

- Note that the sum $\sum_{i=1}^{n} \frac{1}{i}$ is called the harmonic number.

- Therefore, the approximation ratio is $\alpha_n(A_{\text{setcover}}) = O(\log(n))$
Theorem

For any polynomial-time algorithm $A$, the approximation ratio is $\alpha_n(A) = \Omega(\log(n))$ for SetCover, unless $P = NP$.

- Namely, $\alpha_n(A) = O(\log(n))$ is the best to achieve, unless $P = NP$.
- Hence, the greedy algorithm is already the best in terms of order magnitude.
- SetCover does not admit any polynomial-time algorithm with a constant approximation ratio.
- The proof uses PCP theorem, one of the most fundamental theorems in complexity theory.
Suppose that the covers $\mathcal{K}$ belong to different owners.
We are required to balance the costs of covers among the owners.

Example (Traffic Measurement Problem)
The items are links in a network.

A cover is a path in the networks (comprising of some links).

Several agents send probe packets along the selected paths for measurement in parallel.

Goal: Select a subset of paths that traverse all the intended links while keeping the maximum number of paths of any agent to be the minimum.
**Balanced Cover Ownership**

**Definition (BalSetCover)**

- Consider a set \( \mathcal{U} \) of \( n \) and covers \( \mathcal{K} \subseteq 2^\mathcal{U} \) with \( \text{Cost}(S) \) for \( S \in \mathcal{K} \).
- There are \( m \) owners, who split the covers by \( \{\mathcal{K}_1, ..., \mathcal{K}_m\} \), such that \( \bigcup_{i=1}^m \mathcal{K}_i = \mathcal{K} \) and \( \mathcal{K}_i \cap \mathcal{K}_j = \emptyset \) for \( i \neq j \).
- Select the a subset of covers \( \tilde{\mathcal{K}} \subseteq \mathcal{K} \) such that
  - Minimizing the maximum total cost among the owners:
    \[
    \min_{\tilde{\mathcal{K}} \subseteq \mathcal{K}} \left( \max_{i \in \{1, ..., m\}} \sum_{S \in \tilde{\mathcal{K}} \cap \mathcal{K}_i} \text{Cost}(S) \right)
    \]
  - Subject to the constraint of covering all items:
    \[
    \bigcup_{S \in \tilde{\mathcal{K}}} S = \mathcal{U}
    \]
Greedy Algorithm for Balanced Set Cover

Algorithm $\mathcal{A}_{\text{bsetcover}}$

- $\tilde{C} \leftarrow \emptyset$
- $\tilde{\mathcal{K}} \leftarrow \emptyset$
- While $\tilde{C} \neq \mathcal{U}$
  - For each owner $i \in \{1, \ldots, m\}$
    - Find $S \in \mathcal{K}_i$ with the least Price$(S, C)$
    - $\tilde{\mathcal{K}} \leftarrow \tilde{\mathcal{K}} \cup \{S\}$
    - $\tilde{C} \leftarrow \tilde{C} \cup S$
- Return $\tilde{\mathcal{K}}$

- Loop through each owner to pick the least-price cover at each step
- Terminate when all items are covered, like the normal set cover problem
**Greedy Algorithm for Balanced Set Cover**

**Theorem**

The competitive ratio of \( \mathcal{A}_{bsetcover} \) is \( O(m \log(n)) \)

**Proof:**

- Consider a sub-problem by focusing on a particular owner \( i \)
- Let \( U_i = \bigcup_{S \in \mathcal{K}_i} S \)
- Run \( \mathcal{A}_{setcover} \) over \( (U_i, \mathcal{K}_i) \)
- At each step the price of selected cover in \( \mathcal{A}_{setcover} \) cannot be lower than \( \mathcal{A}_{bsetcover} \) for the corresponding owner \( i \)
- Hence, the approximation ratio is at most \( O(\log(n)) \) for each owner, and \( O(m \log(n)) \) for all \( m \) owners
Application: Survivable Network

- Some problems may not appear as set cover problem
- But we can transform them into set cover problem

**Definition (Survivable Network)**

- Given a graph $G$, there are some pairs of sources and destinations
- There are a set paths $P$ connecting every pair of source and destination
- Find a minimal subset $\tilde{P} \subseteq P$, such that there exists at least one path in $\tilde{P}$ for each pair of source and destination that can survive, even though any one link in $G$ fails

Basic idea:

- Items are all the links in $G$
- Covers are the paths in $P$
- A path $P$ covers link $e$, if $P$ does not traverse $e$
Steiner Tree

Definition

- Given a network $\mathcal{G} = (V, E)$
- A subset of nodes $\mathcal{R} \subseteq V$ are called terminals
- Goal: Connect the terminals using the minimum network in $\mathcal{G}$
  - Possibly using vertices not in $\mathcal{R}$ that called Steiner nodes
- Two versions:
  - *Edge-weighted*: Minimum number of links (or weighted total cost)
  - *Node-weighted*: Minimum number of vertices (or weighted total cost)
Example (Wireless Networks)

- In a wireless network, a node can reach all the nodes within its transmission range.
- Terminals want to communicate with each other, even though they are outside the transmission range of each other.
- Our goal is to find a minimum number of relay nodes to relay all data among the set of terminals by a node-weighted Steiner tree.
Inapproximability

Theorem

Node-weighted Steiner tree problem is inapproximable with an approximation ratio $\Omega(\log |\mathcal{R}|)$

Proof:

- Reduce any set cover problem to a node-weighted Steiner tree problem
- The solution of set cover problem has one-to-one correspondence to the solution node-weighted Steiner tree problem
- Given SetCover with $\mathcal{U}$ and $\mathcal{K}$, we create a graph $\mathcal{G}$ such that
  - The set of terminals $\mathcal{R}$ is $\mathcal{U}$
  - Each cover $S \in \mathcal{K}$ is a non-terminal vertex in $\mathcal{G}$
  - Add a link between the cover $S \in \mathcal{K}$ to all covered items by $S$
  - Connect all non-terminal vertices in $\mathcal{G}$ by a complete graph
- The cost of an optimal solution in SetCover is the same as an optimal node-weighted Steiner tree in $\mathcal{G}$
Steiner Tree: Spider Decomposition

**Definition (Spider in a Graph)**

*Spider:*  
- A tree with at most one vertex of degree larger than two

*Foot of Spider:*  
- Center of spider (when three or more leave nodes) or one of the leave nodes

*Non-trivial Spider:*  
- A spider with at least two leave nodes

*Spider Decomposition:*  
- Disjoint union of non-trivial spiders whose feet contains all the terminals in $\mathcal{R}$
Lemma

Given a connected graph $G$ and a subset of vertices $\mathcal{R}$ (where $|\mathcal{R}| \geq 2$), $G$ always contains a spider decomposition of $\mathcal{R}$.

Implications:

- Node-weighted Steiner tree problem can be transformed into set cover problem, through spider decomposition
  - A spider is like a cover
  - The feet of a spider are like the items of a cover
- Goal of constructing a Node-weighted Steiner tree becomes to select the least-price spiders to connect all the terminals
Greedy Algorithm for Steiner Tree

- A spider is like merging a set of trees \( \{T_1, ... T_m\} \)
- Define \( \text{Price}(v, \{T_1, ..., T_m\}) \triangleq \text{cost of } v + \frac{\text{total distance cost to } \{T_1, ..., T_m\}}{m} \)

Algorithm \( \mathcal{A}_\text{nwsteiner} \)

\[
\begin{align*}
\tilde{\mathcal{C}} &\leftarrow \emptyset; \quad \tilde{\mathcal{K}} \leftarrow \emptyset \\
\text{Trees} &\leftarrow \{\{v\} \mid v \in \mathcal{R}\} \\
\text{While } \tilde{\mathcal{C}} \neq \mathcal{R} \\
\quad \text{Find } v \in \mathcal{V} \setminus \tilde{\mathcal{K}} \text{ and } \tilde{T} \subseteq \text{Trees with the least } \text{Price}(v, \tilde{T}) &\quad \text{//Find the least-price spider} \\
\quad \tilde{\mathcal{K}} &\leftarrow \tilde{\mathcal{K}} \cup \{v\} &\quad \text{//To form a new spider with center at } v \\
\quad \tilde{\mathcal{C}} &\leftarrow \{t \mid t \in \mathcal{R} \cap \tilde{T}\} &\quad \text{//Count the covered terminals by the new spider} \\
\quad \text{Trees} &\leftarrow \text{Trees} \setminus \tilde{T} \cup \{\text{Merging } \tilde{T} \text{ as single tree by a spider at } v\} &\quad \text{//Collapse trees into a terminal} \\
\text{Return } \tilde{\mathcal{K}}
\end{align*}
\]
Merging Trees in Greedy Algorithm for Steiner Tree

Merging Trees

Collapsing Trees into Terminals
Theorem

The competitive ratio of $A_{nwsteiner}$ is $O(\log(|R|))$

Proof:

- Let the number of trees at the $k$-th step be $\phi_k \triangleq |Trees_k|
- Let the number of trees merged at the $k$-th step be $m_k = \phi_{k-1} - \phi_k + 1$
- Let $C_k$ be the total cost of adding the spider at the $k$-th step by $A_{nwsteiner}$
- Since $\frac{C_k}{m_k} = \text{Price}(v_k, \tilde{T}_k) \leq \frac{\text{Opt}}{\phi_{k-1}}$, we obtain

$$\frac{C_k \cdot \phi_{k-1}}{\text{Opt}} \leq m_k = \phi_{k-1} - \phi_k + 1 \leq 2(\phi_{k-1} - \phi_k) \quad \text{(since } \phi_{k-1} > \phi_k) \quad (1)$$

$$\phi_k \leq \phi_{k-1}(1 - \frac{C_k}{2 \cdot \text{Opt}}) \text{ telescoping } \phi_k \leq \phi_0 \prod_{j=1}^{k} (1 - \frac{C_j}{2 \cdot \text{Opt}}) \quad (2)$$
Greedy Algorithm for Steiner Tree

Proof (Cont.):

Noting that $1 + x \leq e^x$,

\[
\frac{\phi_k}{\phi_0} \leq \prod_{j=1}^{k} \left(1 - \frac{C_j}{2 \cdot \text{Opt}}\right) \leq \prod_{j=1}^{k} e^{-\frac{C_j}{2 \cdot \text{Opt}}} = e^{-\sum_{j=1}^{k} \frac{C_j}{2 \cdot \text{Opt}}}
\]

(3)

\[
\Rightarrow \sum_{j=1}^{k} C_j \leq 2 \ln \left(\frac{\phi_0}{\phi_k}\right) \cdot \text{Opt} \Rightarrow \text{Cost}(A_{nwsteiner}) \leq 2 \ln(|\mathcal{R}|) \cdot \text{Opt}
\]

(4)
References

Reference Materials
- Approximation Algorithms (V. Vazirani), Springer
  - Chapter 2
- Design of Approximation Algorithms (Williamson, Shmoys), Cambridge University Press
  - Chapter 1

Recommended Materials