# COL758: Advanced Algorithms

Ragesh Jaiswal, CSE, IITD

# (Greedy) Approximation Algorithms

# (Greedy) Approximation Algorithms

- For some problems, even though an efficient algorithm does not give an optimal solution, it might give a solution that is provably close to the optimal solution.
- Such algorithms are called *approximation algorithms*.

### Minimum Vertex Cover Problem

Given a graph $G = (V, E)$, find the smallest subset of nodes such that for every edge $(u, v) \in E$, at least one of $u, v$ is in the subset.

### Algorithm

For a Maximal matching of the given graph, pick both nodes of every edge in the matching.

### Theorem

Let $S$ be the subset of nodes returned by our algorithm for an input graph $G$. Then $|S| \leq 2 \cdot OPT$.

**Minimum Vertex Cover Problem**

Given a graph $G = (V, E)$, find the smallest subset of nodes such that for every edge $(u, v) \in E$, at least one of $u, v$ is in the subset.

**Algorithm**

For a Maximal matching of the given graph, pick both nodes of every edge in the matching.

**Theorem**

Let $S$ be the subset of nodes returned by our algorithm for an input graph $G$. Then $|S| \leq 2 \cdot OPT$.

- *Proof sketch.* The optimal solution contains at least one node from every edge in a maximal matching.

- Covering set: Let $S$ be a set containing $n$ elements. A set of subsets $\{S_1, ..., S_m\}$ of $S$ is called a covering set if each element in $S$ is present in at least one of the subsets $S_1, ..., S_m$.

**Problem**

Set Cover: Given a set $S$ containing $n$ elements and $m$ subsets $S_1, ..., S_m$ of $S$. Find a covering set of $S$ of minimum cardinality.

- Example
  - $S = \{a, b, c, d, e, f\}$
  - $S_1 = \{a, b\}$, $S_2 = \{a, c\}$, $S_3 = \{b, c\}$, $S_4 = \{d, e, f\}$, $S_5 = \{e, f\}$
  - $\{S_1, S_2, S_3, S_4\}$ is a covering set.
  - A covering set of minimum cardinality:?

- Covering set: Let $S$ be a set containing $n$ elements. A set of subsets $\{S_1, ..., S_m\}$ of $S$ is called a covering set if each element in $S$ is present in at least one of the subsets $S_1, ..., S_m$.

**Problem**

Set Cover: Given a set $S$ containing $n$ elements and $m$ subsets $S_1, ..., S_m$ of $S$. Find a covering set of $S$ of minimum cardinality.

- Example
  - $S = \{a, b, c, d, e, f\}$
  - $S_1 = \{a, b\}$, $S_2 = \{a, c\}$, $S_3 = \{b, c\}$, $S_4 = \{d, e, f\}$, $S_5 = \{e, f\}$
  - $\{S_1, S_2, S_3, S_4\}$ is a covering set.
  - A covering set of minimum cardinality: $\{S_1, S_2, S_4\}$

**Problem**

Set Cover: Given a set $S$ containing $n$ elements and $m$ subsets $S_1, ..., S_m$ of $S$. Find a covering set of $S$ of minimum cardinality.

- Application: There are $n$ villages, and the government is trying to figure out which villages to open schools at so that it has to open a minimum number of schools. The constraint is that no children should walk more than 3 miles to get to a school.

**Problem**

Set Cover: Given a set $S$ containing $n$ elements and $m$ subsets $S_1, ..., S_m$ of $S$. Find a covering set of $S$ of minimum cardinality.

- Greedy strategy: Give preference to the subsets that covers the most number of (remaining) elements.

**Algorithm**

`GreedySetCover`$(S, S_1, ..., S_m)$
- $T \leftarrow \{\}$; $R \leftarrow S$
- While $R$ is not empty:
  - Pick a subset $S_i$ that covers the maximum number of elements in $R$
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

**Problem**

Set Cover: Given a set $S$ containing $n$ elements and $m$ subsets $S_1, ..., S_m$ of $S$. Find a covering set of $S$ of minimum cardinality.

- Greedy strategy: Give preference to the subsets that covers the most number of (remaining) elements.

**Algorithm**

GreedySetCover($S, S_1, ..., S_m$)
- $T \leftarrow \{\}$; $R \leftarrow S$
- While $R$ is not empty:
  - Pick a subset $S_i$ that covers the maximum number of elements in $R$
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

- Is this greedy algorithm guaranteed to output an optimal solution?
- Counterexample: $S = \{a, b, c, d, e, f, g, h\}, S_1 = \{a, b, c, d, e\}, S_2 = \{a, b, c, f\}, S_3 = \{d, e, g, h\}$.

**Problem**

Set Cover: Given a set $S$ containing $n$ elements and $m$ subsets $S_1, ..., S_m$ of $S$. Find a covering set of $S$ of minimum cardinality.

- Greedy strategy: Give preference to the subsets that covers the most number of (remaining) elements.

**Algorithm**

`GreedySetCover`$(S, S_1, ..., S_m)$
- $T \leftarrow \{\}$; $R \leftarrow S$
- While $R$ is not empty:
  - Pick a subset $S_i$ that covers the maximum number of elements in $R$
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

- Is this greedy algorithm guaranteed to output an optimal solution? No
- Counterexample: $S = \{a, b, c, d, e, f, g, h\}, S_1 = \{a, b, c, d, e\}, S_2 = \{a, b, c, f\}, S_3 = \{d, e, g, h\}$.

**Algorithm**

GreedySetCover($S, S_1, ..., S_m$)

- $T \leftarrow \{\}$; $R \leftarrow S$
- While $R$ is not empty:
  - Pick a subset $S_i$ that covers the maximum number of elements in $R$
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

- Claim 1: Let $k$ be the cardinality of any optimal covering set. Then the greedy algorithm outputs a covering set with cardinality at most $k \cdot \ln n$.

### Algorithm

GreedySetCover($S, S_1, ..., S_m$)
- $T \leftarrow \{\}$; $R \leftarrow S$
- While $R$ is not empty:
    - Pick a subset $S_i$ that covers the maximum number of elements in $R$
    - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

- Claim 1: Let $k$ be the cardinality of any optimal covering set. Then the greedy algorithm outputs a covering set with cardinality at most $k \cdot \ln n$.

### Proof of Claim 1

- Let $N_t$ be the number of uncovered elements after $t$ iterations of the loop.
- Claim 1.1: $N_t \leq (1 - 1/k) \cdot N_{t-1}$.

### Algorithm

GreedySetCover$(S, S_1, ..., S_m)$

- $T \leftarrow \{\}$; $R \leftarrow S$
- While $R$ is not empty:
  - Pick a subset $S_i$ that covers the maximum number of elements in $R$
  - $T \leftarrow T \cup \{S_i\}$; $R \leftarrow R - S_i$

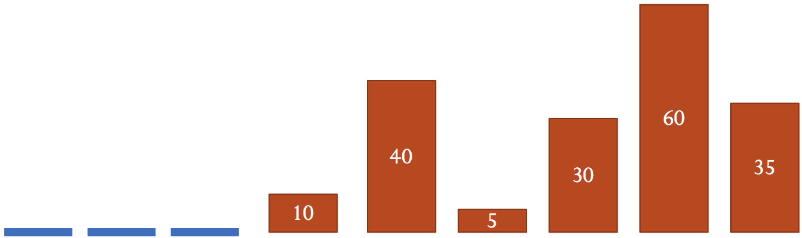
- Claim 1: Let $k$ be the cardinality of any optimal covering set. Then the greedy algorithm outputs a covering set with cardinality at most $k \cdot \ln n$.

### Proof of Claim 1

- Let $N_t$ be the number of uncovered elements after $t$ iterations of the loop.
- Claim 1.1: $N_t \leq (1 - 1/k) \cdot N_{t-1}$.
- Claim 1.2: $N_{k \cdot \ln n} < 1$.
  - Use the fact that $(1 - x) \leq e^{-x}$ and the equality holds only for $x = 0$.

**Problem**

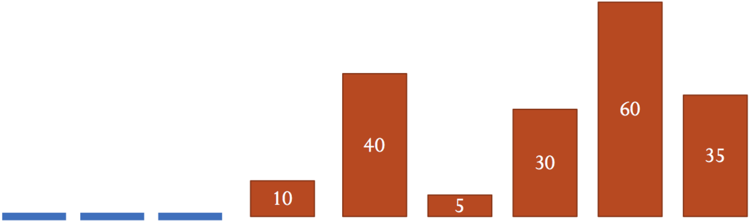
Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.



**Problem**

Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

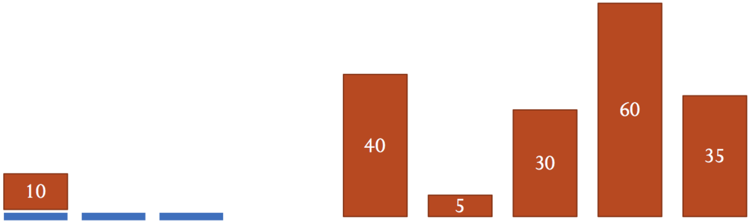
- Greedy strategy: Assign the next job to a machine with the least load.



**Problem**

Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

- Greedy strategy: Assign the next job to a machine with the least load.



**Problem**

Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

- Greedy strategy: Assign the next job to a machine with the least load.

# (Greedy) Approximation Algorithms
## Minimum Makespan

### Problem

Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

- Greedy strategy: Assign the next job to a machine with the least load.

### Problem

Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

- Greedy strategy: Assign the next job to a machine with the least load.

**Problem**

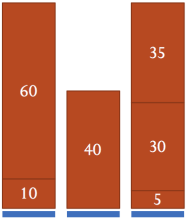
Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

- Greedy strategy: Assign the next job to a machine with the least load.

**Problem**

Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

- Greedy strategy: Assign the next job to a machine with the least load.

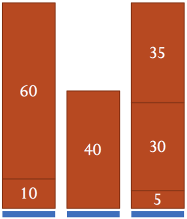



- Is this solution optimal?

**Problem**

Minimum Makespan: You have $m$ identical machines and $n$ jobs. For each job $i$, you are given the duration of this job $d(i)$ that denotes the time required by any machine to perform this job. Assign these $n$ jobs on the $m$ machine to minimise the maximum finishing time.

- Greedy strategy: Assign the next job to a machine with the least load.




- Is this solution optimal? No

**Algorithm**

`GreedyMakespan`
- While all jobs are not assigned
  - Assign the next job to a machine with the least load

- Let $OPT$ be the optimal value.
- Let $G$ denote the maximum finishing time of a machine as per the greedy assignment.
- <u>Claim 1</u>: $G \leq 2 \cdot OPT$.

### Algorithm

`GreedyMakespan`

- While all jobs are not assigned
    - Assign the next job to a machine with the least load

- Let $OPT$ be the optimal value.
- Let $G$ denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

### Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$

**Algorithm**

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with the least load

- Let $OPT$ be the optimal value.
- Let $G$ denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

**Proof of Claim 1**

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job $t$, $OPT \geq d(t)$.

### Algorithm

`GreedyMakespan`
- While all jobs are not assigned
    - Assign the next job to a machine with the least load

- Let $OPT$ be the optimal value.
- Let $G$ denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

### Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job $t$, $OPT \geq d(t)$.
- Let the $j^{th}$ machine finish last. Let $i$ be the last job assigned to machine $j$. Let $s$ be the start time of job $i$ on machine $j$.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$

**Algorithm**

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with the least load

- Let $OPT$ be the optimal value.
- Let $G$ denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

**Proof of Claim 1**

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job $t$, $OPT \geq d(t)$.
- Let the $j^{th}$ machine finish last. Let $i$ be the last job assigned to machine $j$. Let $s$ be the start time of job $i$ on machine $j$.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$
- So, $G \leq s + d(i)$
- This implies that $G \leq \frac{d(1)+...+d(n)}{m} + d(i)$ (using claim 1.3)

### Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with the least load

- Let $OPT$ be the optimal value.
- Let $G$ denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

### Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job $t$, $OPT \geq d(t)$.
- Let the $j^{th}$ machine finish last. Let $i$ be the last job assigned to machine $j$. Let $s$ be the start time of job $i$ on machine $j$.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$
- So, $G \leq s + d(i)$
- This implies that $G \leq \frac{d(1)+...+d(n)}{m} + d(i)$ (using claim 1.3)
- This implies that $G \leq OPT + d(i)$ (using claim 1.1)

### Algorithm

`GreedyMakespan`

- While all jobs are not assigned
  - Assign the next job to a machine with the least load

- Let $OPT$ be the optimal value.
- Let $G$ denote the maximum finishing time of a machine as per the greedy assignment.
- Claim 1: $G \leq 2 \cdot OPT$.

### Proof of Claim 1

- Claim 1.1: $OPT \geq \frac{d(1)+d(2)+...+d(n)}{m}$
- Claim 1.2: For any job $t$, $OPT \geq d(t)$.
- Let the $j^{th}$ machine finish last. Let $i$ be the last job assigned to machine $j$. Let $s$ be the start time of job $i$ on machine $j$.
- Claim 1.3: $s \leq \frac{d(1)+d(2)+...+d(n)}{m}$
- So, $G \leq s + d(i)$
- This implies that $G \leq \frac{d(1)+...+d(n)}{m} + d(i)$ (using claim 1.3)
- This implies that $G \leq OPT + d(i)$ (using claim 1.1)
- This implies that $G \leq OPT + OPT$ (using claim 1.2)

## Problem

<u>$k$-center</u>: Given a set $X$ of $n$ points from a Metric Space $(\mathcal{X}, D)$, find $k$ points $C$ (*called centers*) such that the maximum distance of a point in $X$ to its closest center in $C$ is minimised. In other words, find $k$ centers $C$ such that the following cost function gets minimised:

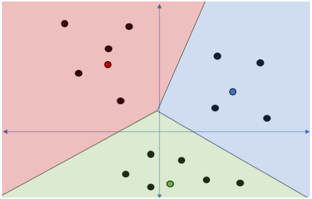
$$cost(C, X) \equiv \max_{x \in X} \{\min_{c \in C} D(x, c)\}.$$

**Problem**

$\underline{k\text{-center}}$: Given a set $X$ of $n$ points from a Metric Space $(\mathcal{X}, D)$, find $k$ points $C$ (*called centers*) such that the maximum distance of a point in $X$ to its closest center in $C$ is minimised. In other words, find $k$ centers $C$ such that the following cost function gets minimised:

$$cost(C, X) \equiv \max_{x \in X} \{\min_{c \in C} D(x, c)\}.$$

- Any set of $k$ centers, partitions the dataset $X$ into $k$ "custers" based on closest center. See the 2-D Euclidean plane example.



- So, the $k$-center problem is one way to cluster a dataset.

### Problem

<u>$k$-center</u>: Given a set $X$ of $n$ points from a Metric Space $(\mathcal{X}, D)$, find $k$ points $C$ (*called centers*) such that the maximum distance of a point in $X$ to its closest center in $C$ is minimised. In other words, find $k$ centers $C$ such that the following cost function gets minimised:

$$cost(C, X) \equiv \max_{x \in X} \{\min_{c \in C} D(x, c)\}.$$

### Algorithm

`Farthest-First`$(X, k)$

- Let $x$ be an arbitrary point in $X$
- $C = \{x\}$
- for $i = 2$ to $k$:
  - Let $c$ be the farthest point in $X$ from points in $C$
  - $C = C \cup \{c\}$
- return$(C)$

### Problem

<u>$k$-center</u>: Given a set $X$ of $n$ points from a Metric Space $(\mathcal{X}, D)$, find $k$ points $C$ (*called centers*) such that the maximum distance of a point in $X$ to its closest center in $C$ is minimised. In other words, find $k$ centers $C$ such that the following cost function gets minimised:

$$cost(C, X) \equiv \max_{x \in X} \{\min_{c \in C} D(x, c)\}.$$

### Algorithm

`Farthest-First`$(X, k)$
- Let $x$ be an arbitrary point in $X$
- $C = \{x\}$
- for $i = 2$ to $k$:
  - Let $c$ be the farthest point in $X$ from points in $C$
  - $C = C \cup \{c\}$
- return$(C)$

### Theorem

For any dataset $X$, let $C$ be the centres returned by the `Farthest-First` algorithm. Then $cost(C, X) \leq 2 \cdot OPT$.

### Algorithm

`Farthest-First`$(X, k)$
- Let $x$ be an arbitrary point in $X$
- $C = \{x\}$
- for $i = 2$ to $k$:
  - Let $c$ be the farthest point in $X$ from points in $C$
  - $C = C \cup \{c\}$
- return$(C)$

### Theorem

For any dataset $X$, let $C$ be the centres returned by the `Farthest-First` algorithm. Then $cost(C, X) \leq 2 \cdot OPT$.

### Proof sketch

- Let $o_1, ..., o_k$ be the optimal centers and let $X_1, ..., X_k$ be the corresponding Voronoi partitions of $X$.
- Case 1: Every $X_i$ has exactly one center (say $c_i$) from $C$.
- Case 2: There is an $X_i$ that has more than one center from $C$.

# (Greedy) Approximation Algorithms

## $k$-center

**Algorithm**

`Farthest-First`$(X, k)$

- Let $x$ be an arbitrary point in $X$
- $C = \{x\}$
- for $i = 2$ to $k$:
  - Let $c$ be the farthest point in $X$ from points in $C$
  - $C = C \cup \{c\}$
- return$(C)$

**Theorem**

For any dataset $X$, let $C$ be the centres returned by the `Farthest-First` algorithm. Then $cost(C, X) \leq 2 \cdot OPT$.

**Proof sketch**

- Let $o_1, ..., o_k$ be the optimal centers and let $X_1, ..., X_k$ be the corresponding Voronoi partitions of $X$.
- Case 1: Every $X_i$ has exactly one center (say $c_i$) from $C$.
  - The distance of any point $x \in X_i$ from $c_i$ is bounded by $D(x, o_i) + D(o_i, c_i) \leq 2 \cdot OPT$.
- Case 2: There is an $X_i$ that has more than one center from $C$.
  - Let $c$ and $c'$ be two centers from $C$ in $X_i$ such that $c'$ is chosen later than $c$ by our algorithm. Since $c'$ is the "*farthest*" point from $C$ at the time it was chosen, the distance of any point $x \in X$ from $C$ is bounded by the distance of $c'$ from $c$. This, in turn, is bounded by $D(c, o_i) + D(o_i, c') \leq 2 \cdot OPT$.

## End