Lecture: Approximation Algorithms

Jannik Matuschke

October 29, 2018

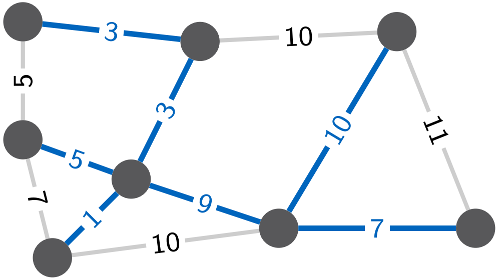
Greedy Algorithms and Local Search

Example I: Minimum Spanning Trees

The Minimum Spanning Tree Problem

Input: graph $G = (V, E)$, distances $d : E \to \mathbb{R}_+$

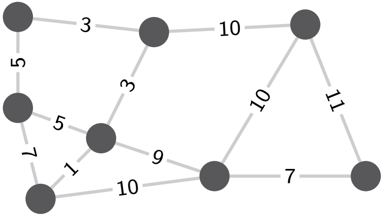
Task: find a spanning tree T in G
minimizing $\sum_{e \in T} d(e)$



Kruskal's algorithm

Algorithm:

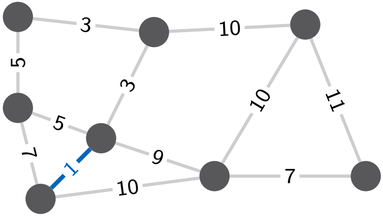
1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \ : \ T \cup \{e\}$ contains no cycle$\}$ minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

Algorithm:

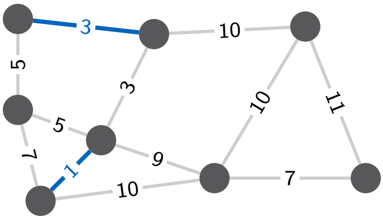
1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \ : \ T \cup \{e\}$ contains no cycle$\}$
   minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

Algorithm:

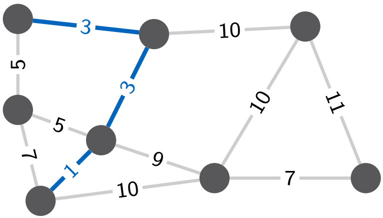
1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \ : \ T \cup \{e\}$ contains no cycle$\}$
   minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

Algorithm:

1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \;:\; T \cup \{e\} \text{ contains no cycle}\}$
   minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

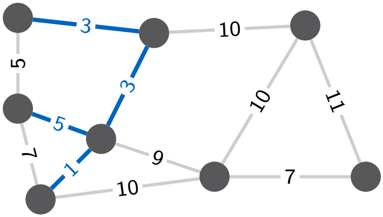
Algorithm:

1. $T := \emptyset$
2. while (T is not a spanning tree)

   Choose $e' \in \{e \in E \setminus T \;:\; T \cup \{e\} \text{ contains no cycle}\}$

   minimizing $d(e')$.

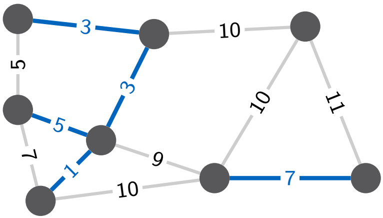
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

Algorithm:

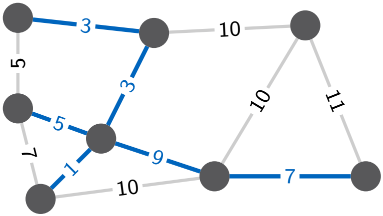
1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \ : \ T \cup \{e\}$ contains no cycle$\}$ minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

Algorithm:

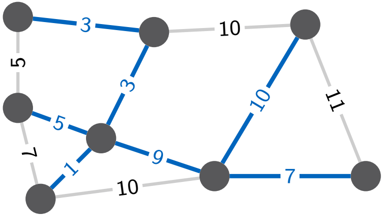
1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \ : \ T \cup \{e\}$ contains no cycle$\}$
   minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

Algorithm:

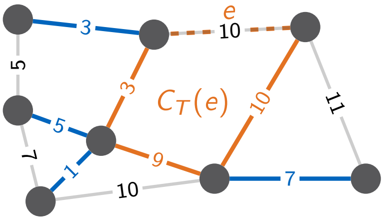
1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \ : \ T \cup \{e\}$ contains no cycle$\}$
   minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Kruskal's algorithm

Algorithm:

1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \ : \ T \cup \{e\}$ contains no cycle$\}$
   minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.

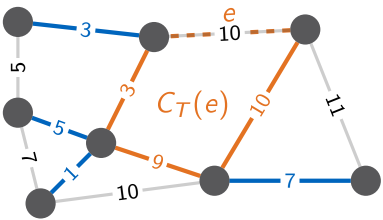


Definition

- For $e \in E \setminus T$, let $C_T(e)$ be the unique cycle in $T \cup \{e\}$.

Algorithm:

1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \,:\, T \cup \{e\}$ contains no cycle$\}$ minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



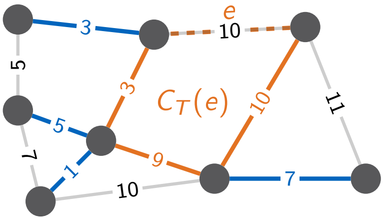
Definition

- For $e \in E \setminus T$, let $C_T(e)$ be the unique cycle in $T \cup \{e\}$.
- A tree T is swap-optimal if $d(e) \geq d(f)$ for all $e \in E \setminus T$, $f \in C_T(e)$.

Kruskal's algorithm

Algorithm:

1. $T := \emptyset$
2. while (T is not a spanning tree)
   Choose $e' \in \{e \in E \setminus T \; : \; T \cup \{e\}$ contains no cycle$\}$
   minimizing $d(e')$.
   Set $T := T \cup \{e'\}$.
3. Return T.



Definition

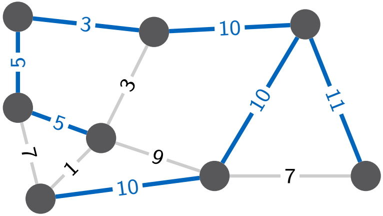
- For $e \in E \setminus T$, let $C_T(e)$ be the unique cycle in $T \cup \{e\}$.
- A tree T is swap-optimal if $d(e) \geq d(f)$ for all $e \in E \setminus T$, $f \in C_T(e)$.

Lemma 3.1

Kruskal's algorithm returns a swap-optimal tree.

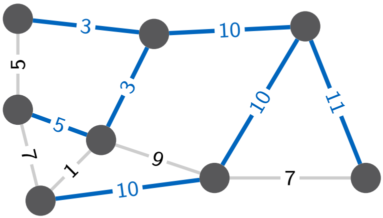
Algorithm:

1. $T :=$ some spanning tree
2. while $(\exists\, e \in E \setminus T,\ f \in C_T(e) : d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.



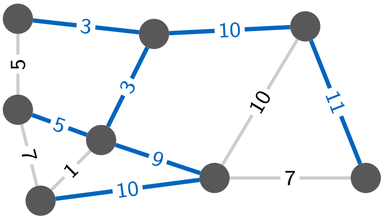
Algorithm:

1. T := some spanning tree
2. while $(\exists\, e \in E \setminus T,\ f \in C_T(e)\ :\ d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.



Algorithm:

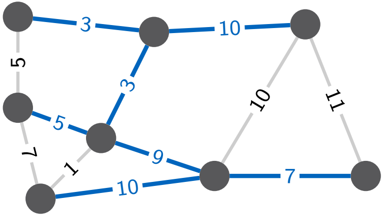
1. $T :=$ some spanning tree
2. while $(\exists\, e \in E \setminus T,\ f \in C_T(e)\ :\ d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.



Local Search

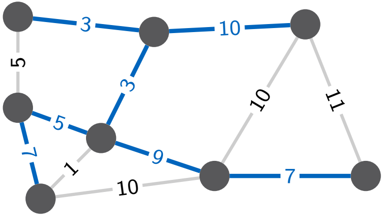
Algorithm:

1. $T :=$ some spanning tree
2. while $(\exists\, e \in E \setminus T,\ f \in C_T(e)\ :\ d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.



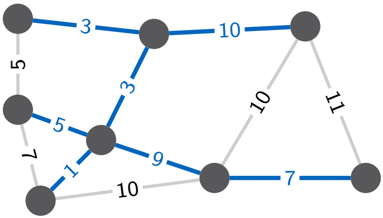
Algorithm:

1. $T :=$ some spanning tree
2. while $(\exists\, e \in E \setminus T,\ f \in C_T(e) \,:\, d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.



Algorithm:

1. $T :=$ some spanning tree
2. while $(\exists\, e \in E \setminus T,\; f \in C_T(e) \,:\, d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.

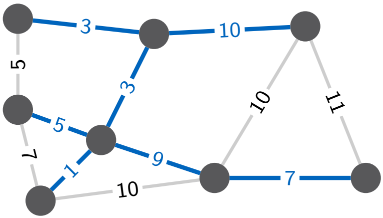


Algorithm:

1. $T :=$ some spanning tree
2. while $(\exists\, e \in E \setminus T,\ f \in C_T(e)\ :\ d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.

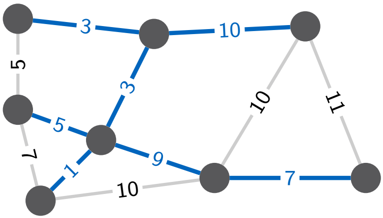
Lemma 3.2

The local search algorithm returns a swap-optimal tree.



Algorithm:

1. $T :=$ some spanning tree
2. while $(\exists\, e \in E \setminus T,\ f \in C_T(e)\ :\ d(e) < d(f))$
   Set $T := T \setminus \{f\} \cup \{e\}$.
3. Return T.



Lemma 3.2

The local search algorithm returns a swap-optimal tree.

Theorem 3.3

A tree is a minimum spanning tree if and only if it is swap-optimal.

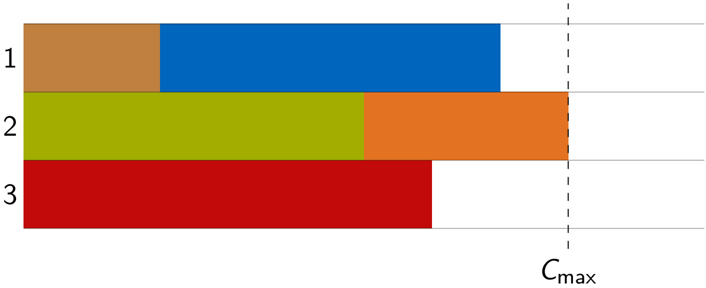
Greedy Algorithms and Local Search

Example II: Scheduling on Identical Parallel Machines

Scheduling on Parallel Machines

Input: m identical machines,
n jobs with processing times $p_1, \ldots, p_n$

Task: assign each job $j \in [n]$ to a machine $\sigma(j) \in [m]$
minimizing $C_{\max} := \max_{i \in [m]} \sum_{j : \sigma(j) = i} p_j$



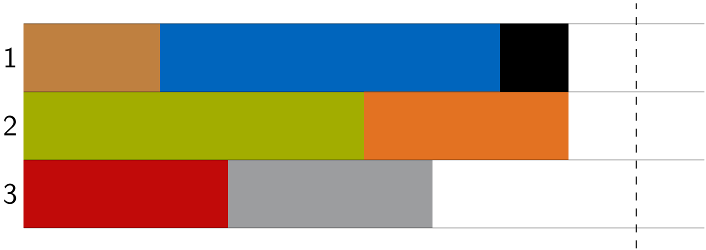
Algorithm:

1. Let σ be some assignment.
2. while $(\exists\, i \in [m],\ j \in [n] : \text{load}_\sigma(i) + p_j < \text{load}_\sigma(\sigma(j)))$
   Set $\sigma(j) := i$.
3. Return σ.

Algorithm:

1. Let σ be some assignment.
2. while $(\exists\, i \in [m],\ j \in [n] : \text{load}_\sigma(i) + p_j < \text{load}_\sigma(\sigma(j)))$

   Set $\sigma(j) := i$.
3. Return σ.



Algorithm:

1. Let σ be some assignment.
2. while $(\exists\, i \in [m],\ j \in [n] : \text{load}_\sigma(i) + p_j < \text{load}_\sigma(\sigma(j)))$
   Set $\sigma(j) := i$.
3. Return σ.

Algorithm:

1. Let σ be some assignment.
2. while $(\exists\, i \in [m],\ j \in [n] : \mathrm{load}_\sigma(i) + p_j < \mathrm{load}_\sigma(\sigma(j)))$
   Set $\sigma(j) := i$.
3. Return σ.

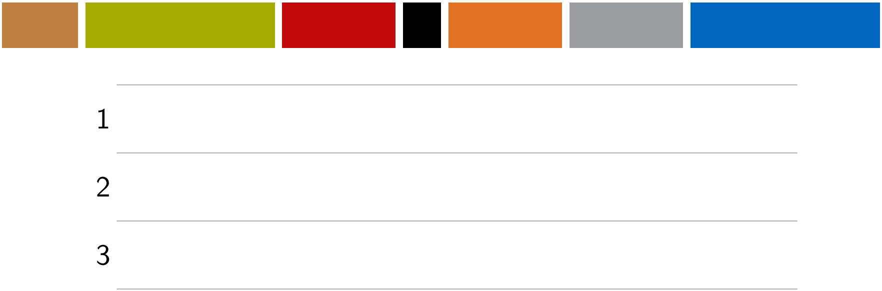
Theorem 3.4

Local search is a 2-approximation for $P||C_{\max}$.*

List Scheduling

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.



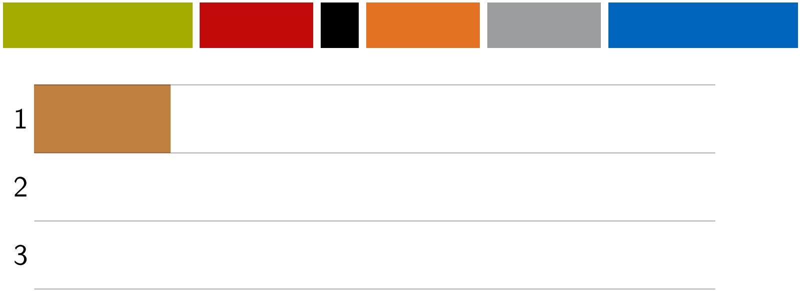
Theorem 3.5

List Scheduling is a 2-approximation for $P||C_{\max}$.

List Scheduling

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.



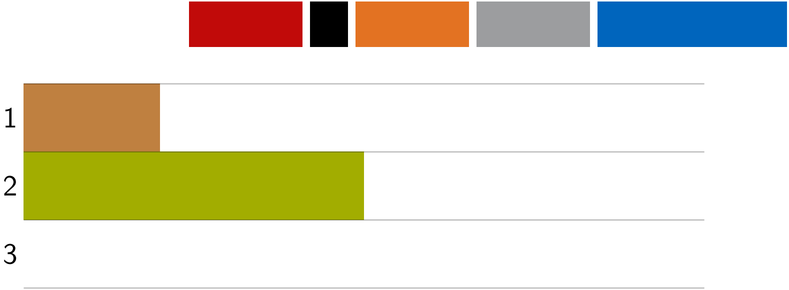
Theorem 3.5

List Scheduling is a 2-approximation for $P||C_{\max}$.

List Scheduling

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.



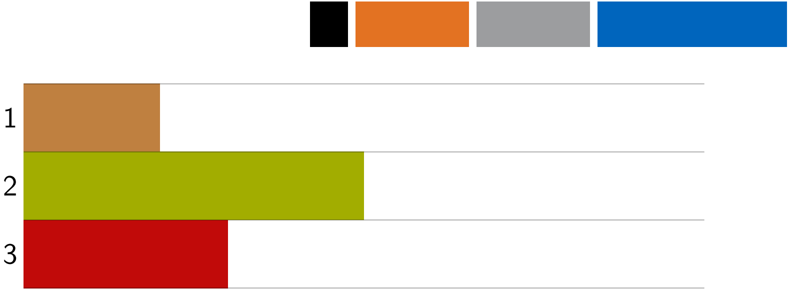
Theorem 3.5

List Scheduling is a 2-approximation for $P||C_{\max}$.

List Scheduling

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.



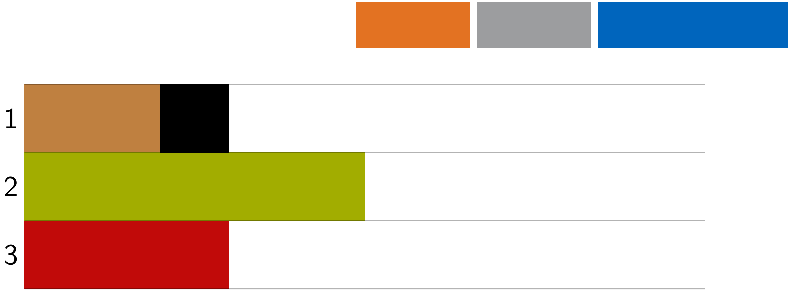
Theorem 3.5

List Scheduling is a 2-approximation for $P||C_{\max}$.

List Scheduling

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.



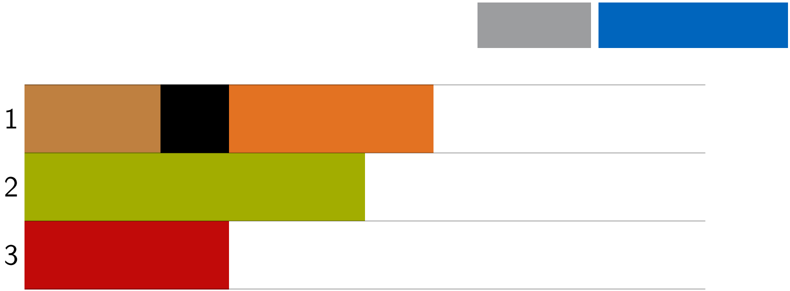
Theorem 3.5

List Scheduling is a 2-approximation for $P||C_{\max}$.

List Scheduling

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.

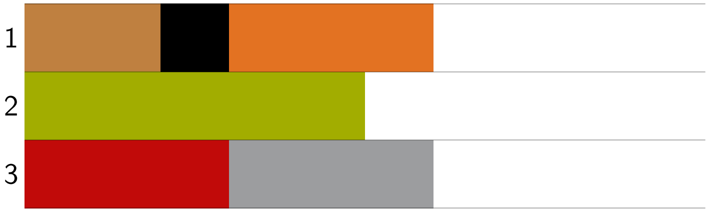


Theorem 3.5

List Scheduling is a 2-approximation for $P||C_{\max}$.

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.



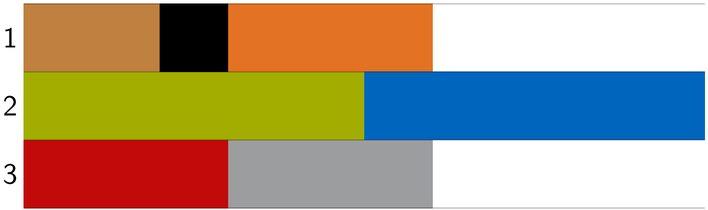
Theorem 3.5

List Scheduling is a 2-approximation for $P||C_{\max}$.

List Scheduling

Algorithm:

1. For $j := 1$ to n
   Assign j to machine i with lowest load.



Theorem 3.5

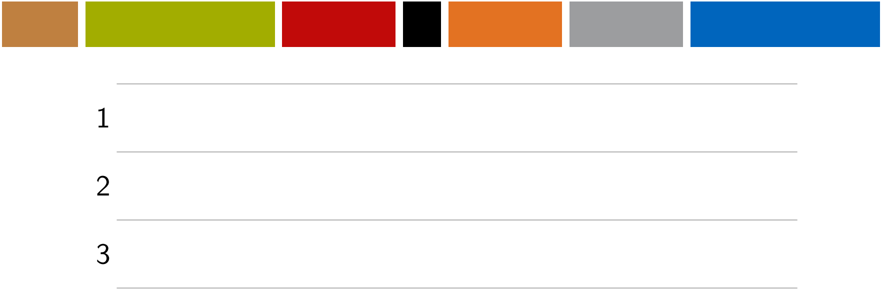
List Scheduling is a 2-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n

   Assign j to machine i with lowest load.



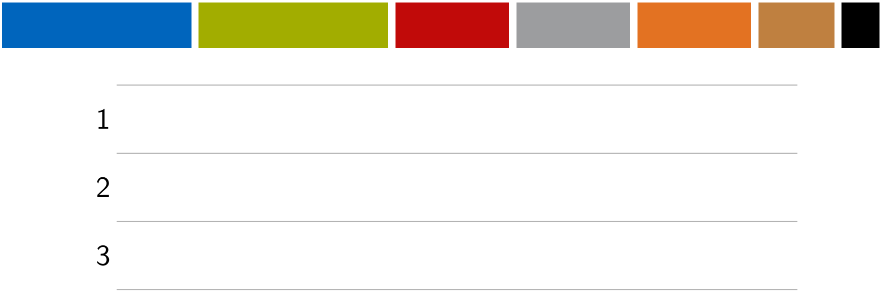
Theorem 3.6

LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n
   Assign j to machine i with lowest load.



Theorem 3.6

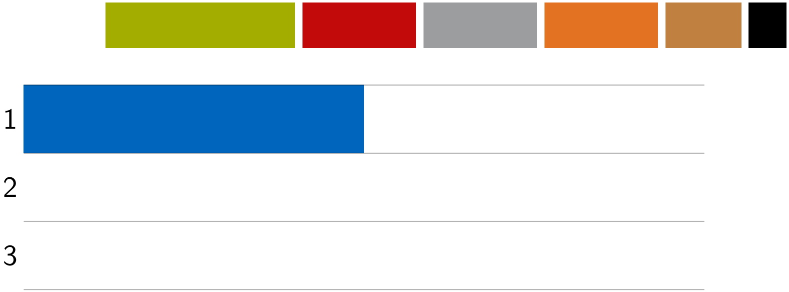
LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n

   Assign j to machine i with lowest load.



Theorem 3.6

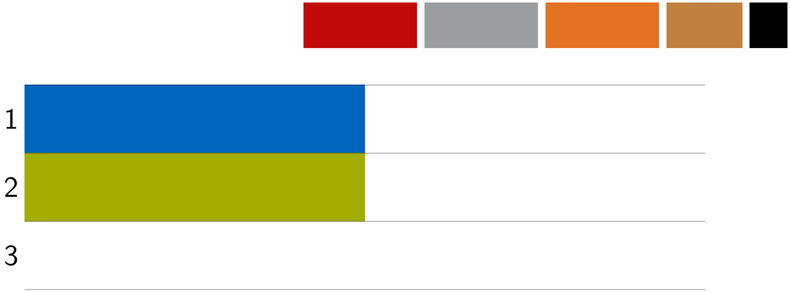
LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n

   Assign j to machine i with lowest load.



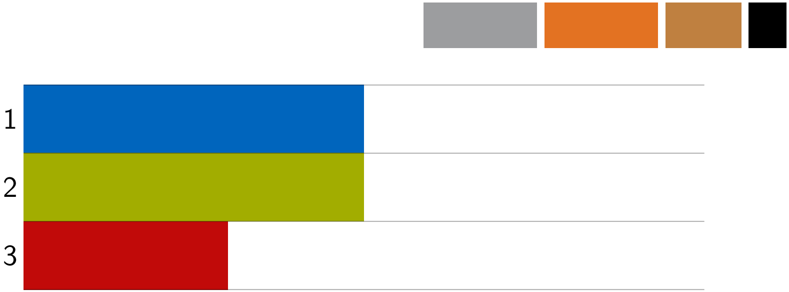
Theorem 3.6

LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n
   Assign j to machine i with lowest load.



Theorem 3.6

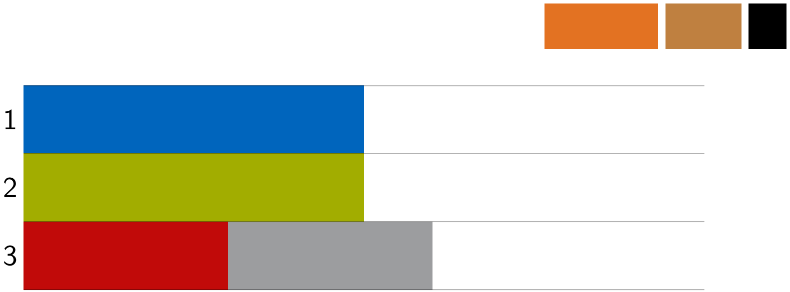
LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n

   Assign j to machine i with lowest load.



Theorem 3.6

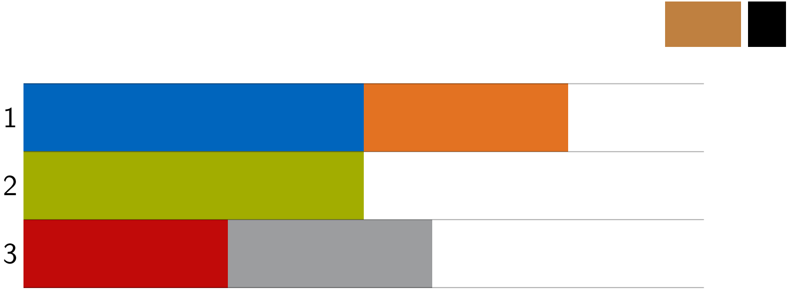
LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n

   Assign j to machine i with lowest load.



Theorem 3.6

LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \cdots \geq p_n$.
2. For $j := 1$ to n

   Assign j to machine i with lowest load.

Theorem 3.6

LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

Longest Processing Time First

Algorithm:

1. Order jobs such that $p_1 \geq p_2 \geq \dots \geq p_n$.
2. For $j := 1$ to n

   Assign j to machine i with lowest load.

Theorem 3.6

LPT List Scheduling is a 4/3-approximation for $P||C_{\max}$.

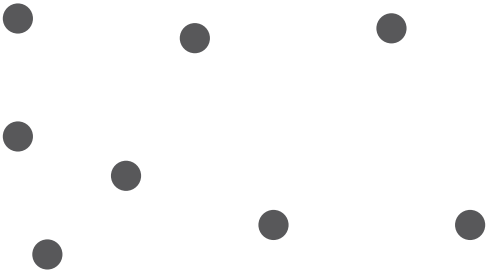
Greedy Algorithms

Example III: The k-Center Problem

The k-Center Problem

Input: clients V, metric $d : V \times V \to \mathbb{R}_+$

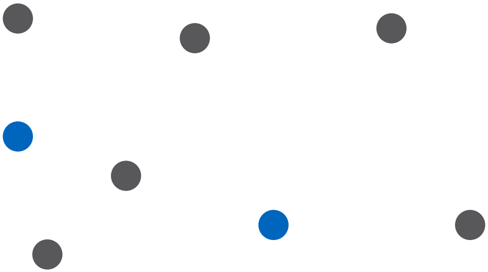
Task: find $S \subseteq V$ with $|S| \leq k$,
minimizing $\max_{v \in V} d(v, S)$
where $d(v, S) := \min_{s \in S} d(v, s)$



The k-Center Problem

Input: clients V, metric $d : V \times V \to \mathbb{R}_+$

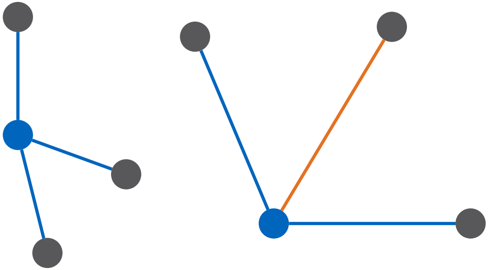
Task: find $S \subseteq V$ with $|S| \leq k$,
minimizing $\max_{v \in V} d(v, S)$
where $d(v, S) := \min_{s \in S} d(v, s)$



The k-Center Problem

Input: clients V, metric $d : V \times V \to \mathbb{R}_+$

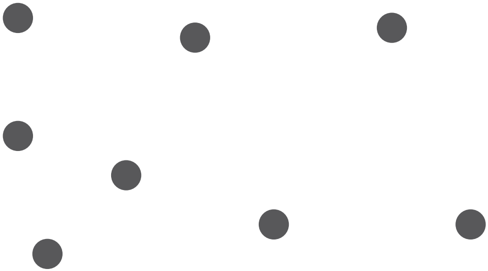
Task: find $S \subseteq V$ with $|S| \leq k$,
minimizing $\max_{v \in V} d(v, S)$
where $d(v, S) := \min_{s \in S} d(v, s)$



Greedy algorithm

Algorithm:

1. Let $S := \{v_0\}$ for some $v_0 \in V$.
2. while $(|S| < k)$
   Let $v \in \operatorname{argmax} d(v, S)$.
   Add v to S.
3. Return S.



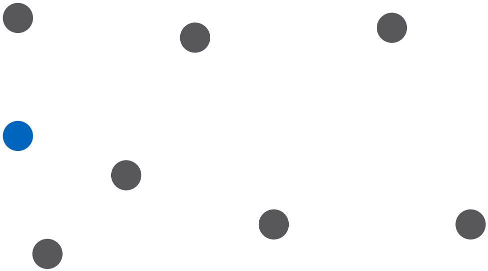
Theorem 3.7

The greedy algorithm is a 2-approximation for k-CENTER.

Greedy algorithm

Algorithm:

1. Let $S := \{v_0\}$ for some $v_0 \in V$.
2. while $(|S| < k)$
   Let $v \in \operatorname{argmax} d(v, S)$.
   Add v to S.
3. Return S.



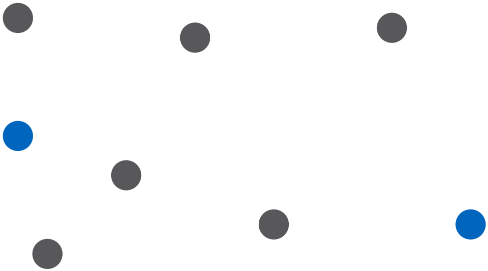
Theorem 3.7

The greedy algorithm is a 2-approximation for k-CENTER.

Greedy algorithm

Algorithm:

1. Let $S := \{v_0\}$ for some $v_0 \in V$.
2. while $(|S| < k)$
   Let $v \in \operatorname{argmax} d(v, S)$.
   Add v to S.
3. Return S.



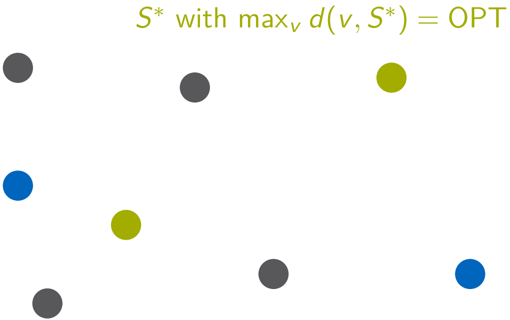
Theorem 3.7

The greedy algorithm is a 2-approximation for k-Center.

Greedy algorithm

Algorithm:

1. Let $S := \{v_0\}$ for some $v_0 \in V$.
2. while $(|S| < k)$
   Let $v \in \operatorname{argmax} d(v, S)$.
   Add v to S.
3. Return S.



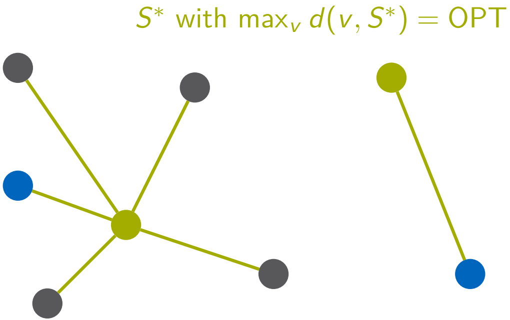
Theorem 3.7

The greedy algorithm is a 2-approximation for k-CENTER.

Greedy algorithm

Algorithm:

1. Let $S := \{v_0\}$ for some $v_0 \in V$.
2. while $(|S| < k)$
   Let $v \in \operatorname{argmax} d(v, S)$.
   Add v to S.
3. Return S.



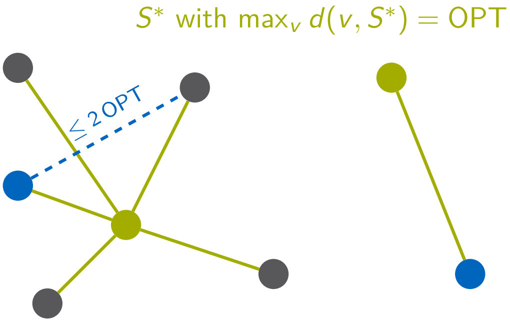
Theorem 3.7

The greedy algorithm is a 2-approximation for k-CENTER.

Greedy algorithm

Algorithm:

1. Let $S := \{v_0\}$ for some $v_0 \in V$.
2. while $(|S| < k)$
   Let $v \in \operatorname{argmax} d(v, S)$.
   Add v to S.
3. Return S.



Theorem 3.7

The greedy algorithm is a 2-approximation for k-Center.

Theorem 3.8

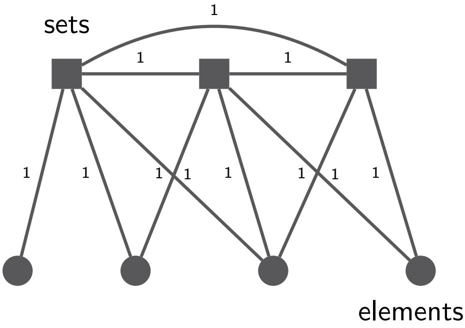
For any $\alpha < 2$, there is no α-approximation for k-CENTER, unless $P = NP$.

Hardness of Approximation

Theorem 3.8

For any $\alpha < 2$, there is no α-approximation for k-CENTER, unless $P = NP$.

Reduction from SET COVER:



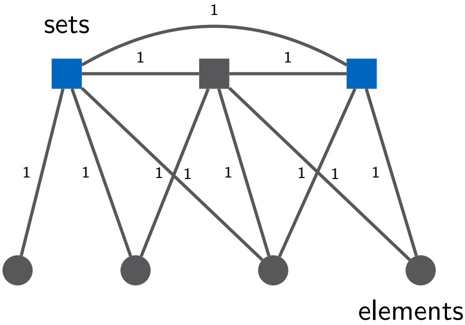
all other distances: 2

Hardness of Approximation

Theorem 3.8

For any $\alpha < 2$, there is no α-approximation for k-Center, unless $P = NP$.

Reduction from Set Cover:



all other distances: 2

OPT $= 1$ if and only if there is set cover of size $\leq k$.

Today you learnt ...

Greed can be good.*

*If you know how to analyze it.

Local Search:

If the grass is greener on the other side,

move into your neighbor's house.

(and repeat)