Design and Analysis of Algorithms
Approximation Algorithms

Si Wu

School of CSE, SCUT
cswusi@scut.edu.cn

TA: 1684350406@qq.com

Topics

- **Load Balancing**
- **Center Selection**
- **Weighted Vertex Cover: Pricing Method**
- **Weighted Vertex Cover: LP Rounding**

Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.

- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

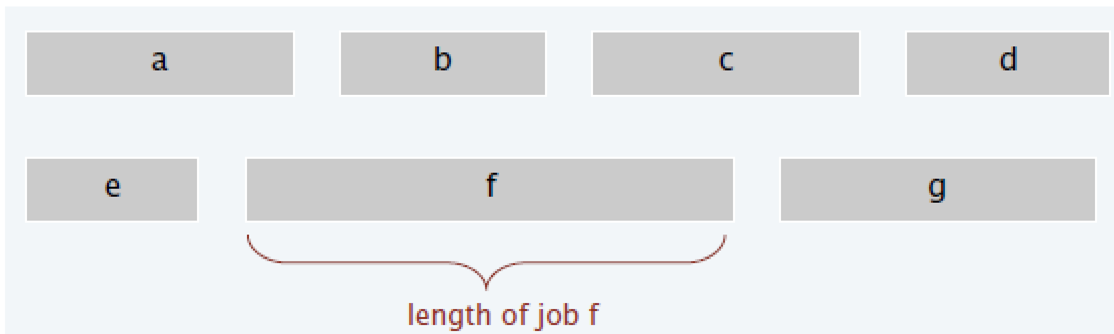
Def. Let $S[i]$ be the subset of jobs assigned to machine i.
The load of machine i is $L[i] = \sum_{j \in S[i]} t_j$.

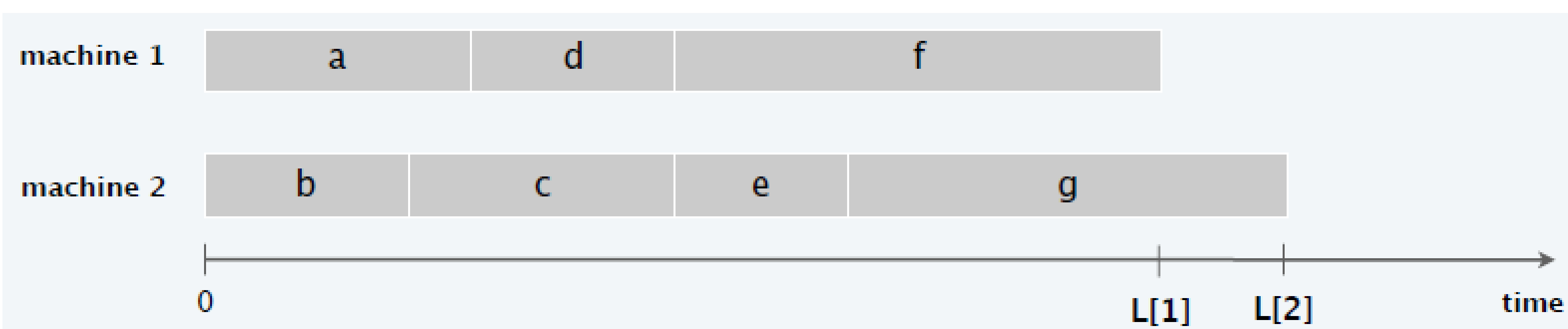
Def. The makespan is the maximum load on any machine $L = max_i L[i]$.

Load balancing. Assign each job to a machine to minimize makespan.

Load Balancing on 2 Machines is NP-Hard

Claim. Load balancing is hard even if $m = 2$ machines.





Load Balancing: List Scheduling

List-scheduling algorithm.

- Consider n jobs in some fixed order.
- Assign job j to machine i whose load is smallest so far.

List-Scheduling $(m, n, t_1, \ldots, t_n)$

For $i = 1$ to m

$L[i] = 0.$

$S[i] \leftarrow \emptyset.$

For $j = 1$ to n

$i \leftarrow argmin_k L[k].$

$S[i] \leftarrow S[i] \cup \{j\}.$

$L[i] \leftarrow L[i] + t_j.$

Return $S[1], S[2], \ldots, S[m].$

Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan L^*.

Lemma 1. The optimal makespan $L^* \geq max_j t_j$.

Pf.

Some machine must process the most time-consuming job.

Lemma 2. The optimal makespan $L^* \geq \frac{1}{m} \sum_j t_j$.

Pf.

- The total processing time is $\sum_j t_j$.
- One of m machines must do at least a $\frac{1}{m}$ fraction of total work.

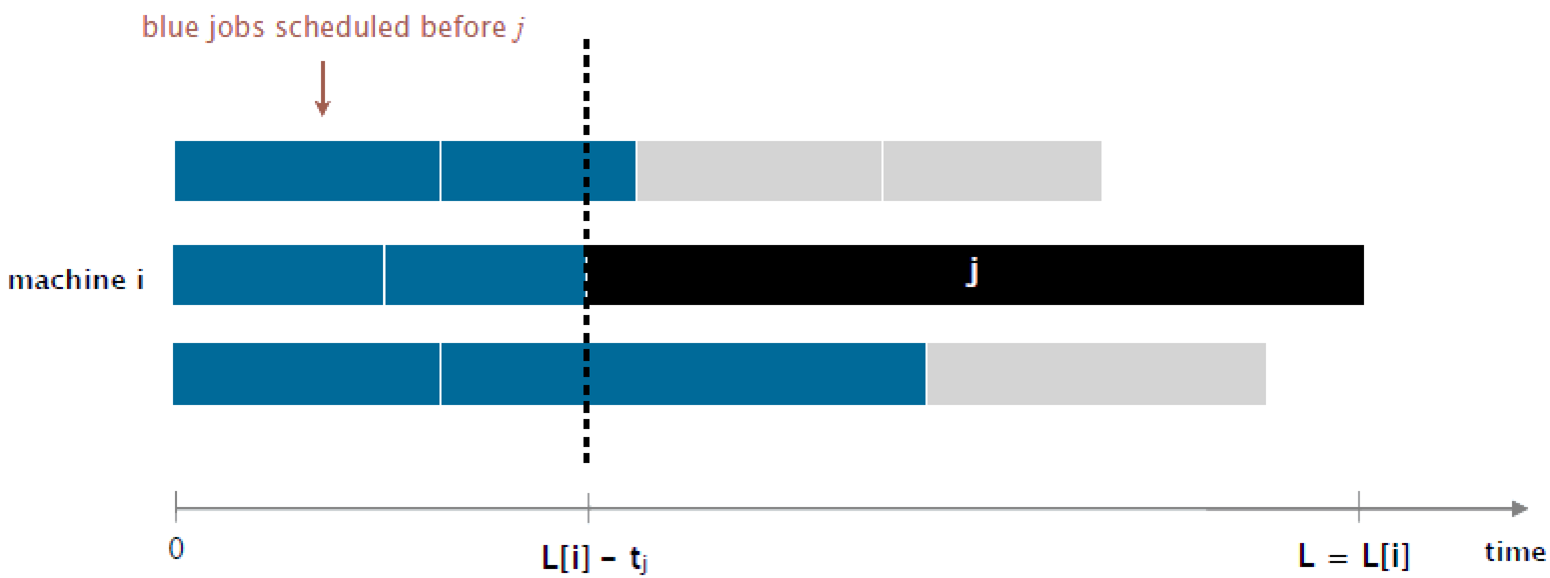
Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load $L[i]$ of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i has smallest load.

Its load before assignment is $L[i] - t_j \Rightarrow L[i] - t_j \le L[k]$ for all $1 \le k \le m$.



Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load $L[i]$ of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i has smallest load.

Its load before assignment is $L[i] - t_j \Longrightarrow L[i] - t_j \leq L[k]$ for all $1 \leq k \leq m$.

- Sum inequalities over all k and divide by m:

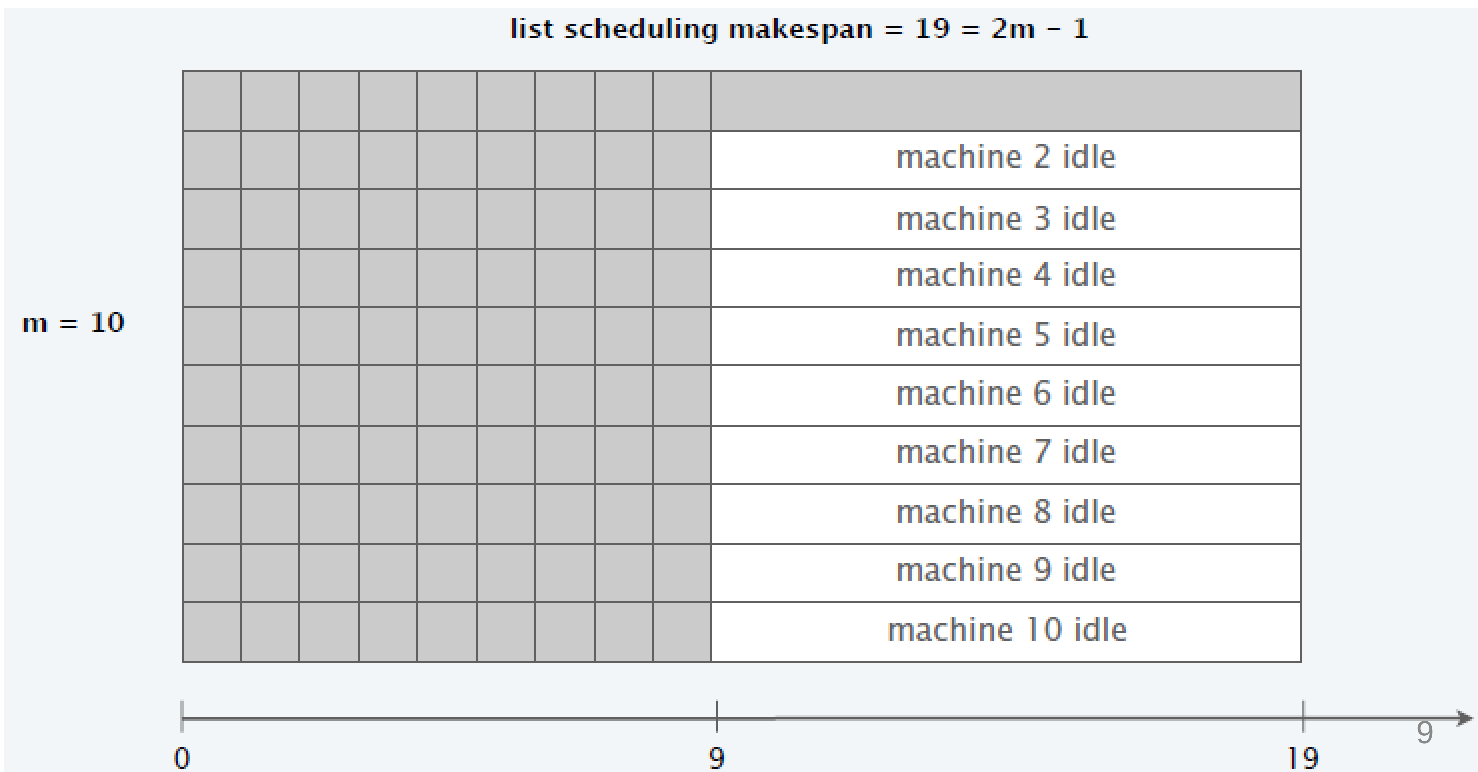
$$L[i] - t_j \leq \frac{1}{m} \sum_k L[k] = \frac{1}{m} \sum_j t_j \leq L^*$$

- Now, $L = L[i] = \left(L[i] - t_j\right) + t_j \leq 2L^*$.

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, $m(m-1)$ jobs length 1, one job of length m.

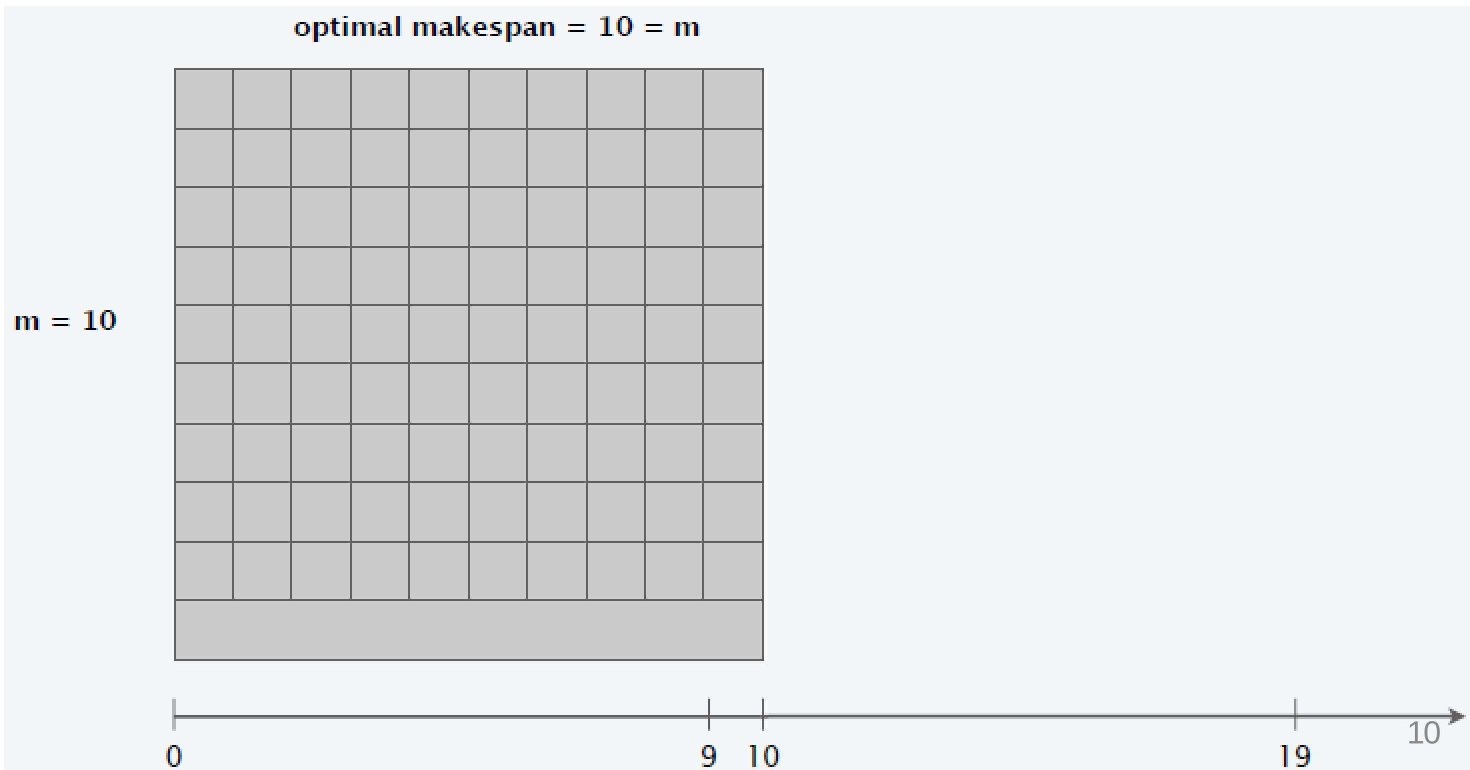


Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, $m(m-1)$ jobs length 1, one job of length m.



Load Balancing: LPT Rule

Longest Processing Time (LPT). Sort n jobs in decreasing order of processing times; then run list scheduling algorithm.

LPT-List-Scheduling $(m, n, t_1, \ldots, t_n)$

Sort jobs and renumber so that $t_1 \geq t_2 \geq \cdots \geq t_n$.

For $i = 1$ to m
 $L[i] = 0.$
 $S[i] \leftarrow \emptyset.$

For $j = 1$ to n
 $i \leftarrow argmin_k L[k].$
 $S[i] \leftarrow S[i] \cup \{j\}.$
 $L[i] \leftarrow L[i] + t_j.$

Return $S[1], S[2], \ldots, S[m].$

Load Balancing: LPT Rule

Ex.

LPT-List-Scheduling $(m, n, t_1, \dots, t_n)$

Sort jobs and renumber so that $t_1 \geq t_2 \geq \cdots \geq t_n$.

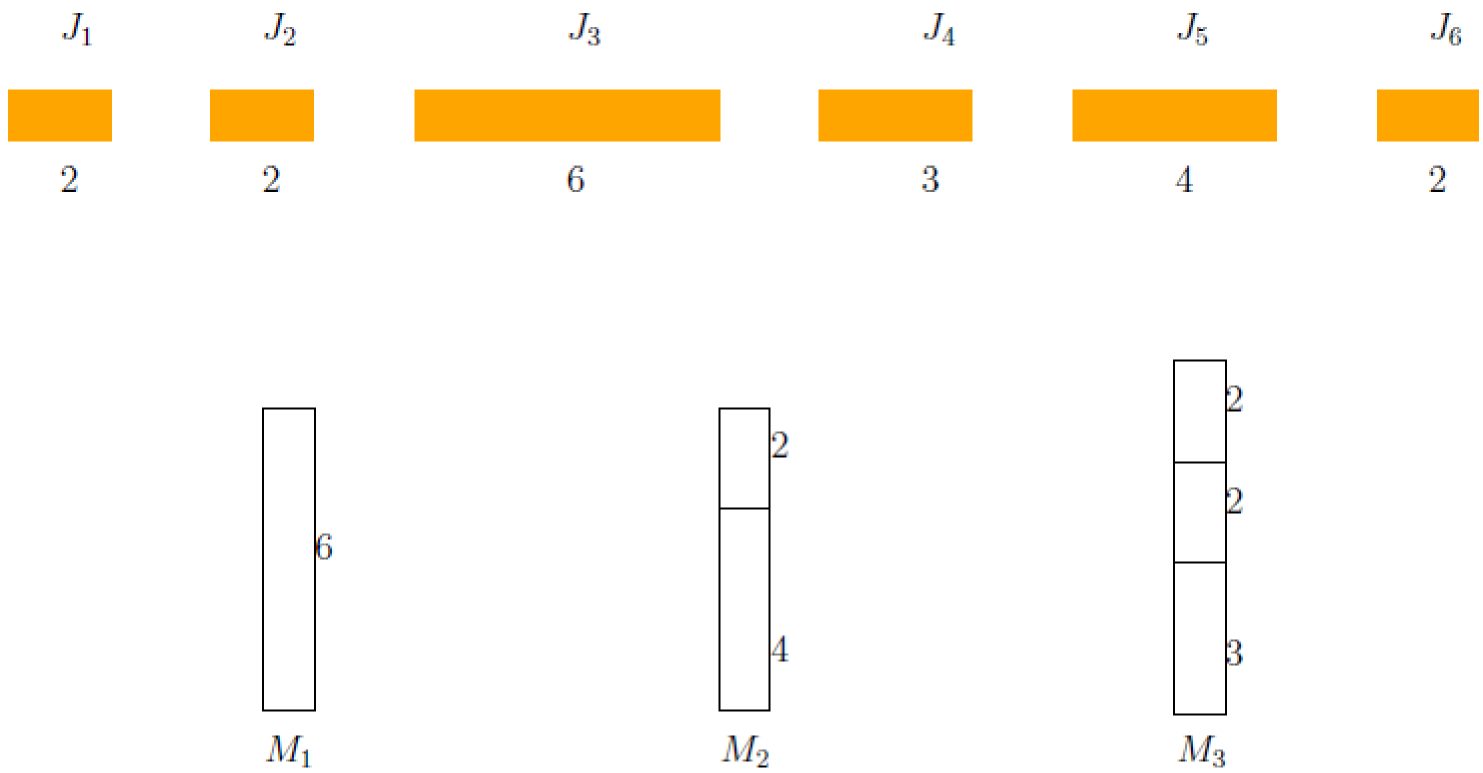
For $i = 1$ to m
  $L[i] = 0$.
  $S[i] \leftarrow \emptyset$.

For $j = 1$ to n
  $i \leftarrow argmin_k L[k]$.
  $S[i] \leftarrow S[i] \cup \{j\}$.
  $L[i] \leftarrow L[i] + t_j$.

Return $S[1], S[2], \dots, S[m]$.

Load Balancing: LPT Rule

Ex.



Load Balancing: LPT Rule

Observation. If bottleneck machine i has only 1 job, then optimal.

Pf. Any solution must schedule that job.

Lemma 3. If there are more than m jobs, $L^* \geq 2t_{m+1}$.

Pf.

- Consider processing times of first $m+1$ jobs $t_1 \geq t_2 \geq \cdots \geq t_{m+1}$.
- Each takes at least t_{m+1} time.
- There are $m+1$ jobs and m machines, so at least one machine gets two jobs.

Theorem. LPT rule is a 3/2-approximation algorithm.

Pf. [similar to proof for list scheduling]

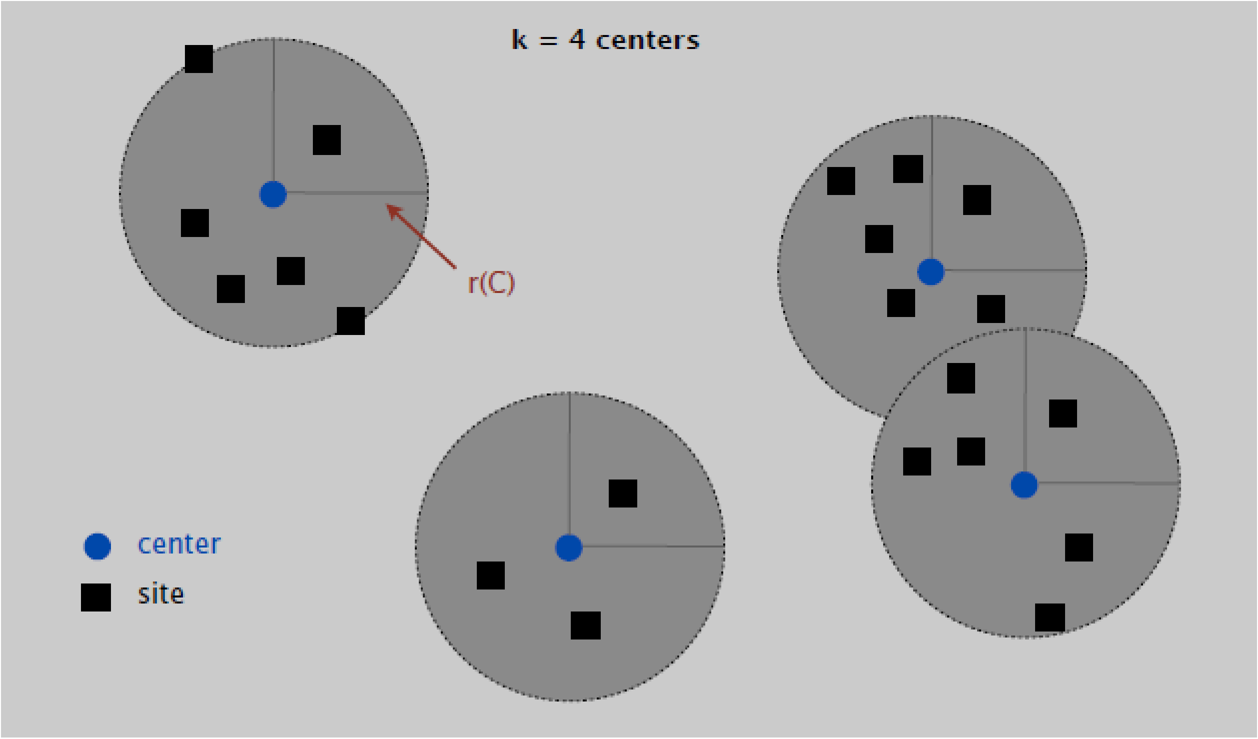
- Consider load $L[i]$ of bottleneck machine i.
- Let j be the last job scheduled on machine i.

$$L = L[i] = \left(L[i] - t_j\right) + t_j \leq \frac{3}{2} L^*$$

Center Selection Problem

Input. Set of n sites $s_1, s_2, \ldots, s_n$ and an integer $k > 0$.

Center selection problem. Select set of k centers C so that maximum distance $r(C)$ from a site to nearest center is minimized.



Center Selection Problem

Input. Set of n sites $s_1, s_2, \ldots, s_n$ and an integer $k > 0$.

Center selection problem. Select set of k centers C so that maximum distance $r(C)$ from a site to nearest center is minimized.

Notation.

- $dist(x, y) =$ distance between sites x and y.
- $dist(s_i, C) = min_c dist(s_i, c) =$ distance from s_i to closest center.
- $r(C) = max_i dist(s_i, C) =$ smallest covering radius.

Goal. Find set of centers C that minimizes $r(C)$, subject to $|C| = k$.

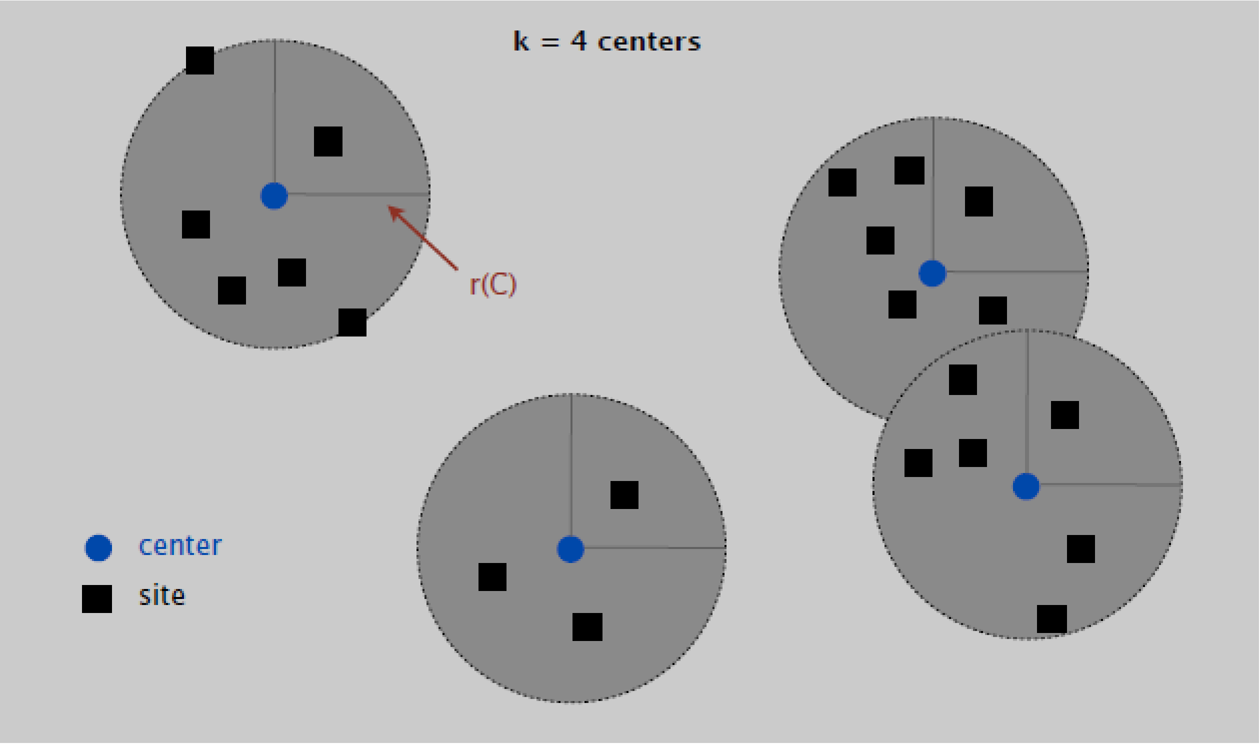
Distance function properties.

- $dist(x, y) = 0$ [identity]
- $dist(x, y) = dist(y, x)$ [symmetry]
- $dist(x, y) \leq dist(x, z) + dist(z, y)$ [triangle inequality]

Center Selection Example

Ex: each site is a point in the plane, a center can be any point in the plane, $dist(x, y) =$ Euclidean distance.

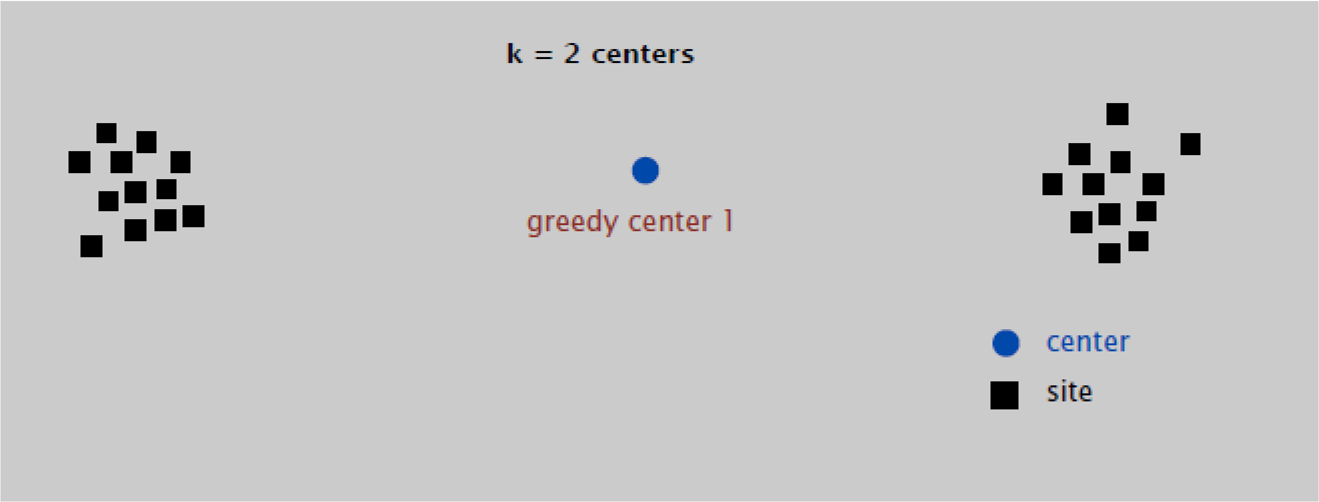
Remark: search can be infinite!



Greedy Algorithm: A False Start

Greedy algorithm. Put the first center at the best possible location for a single center, and then keep adding centers so as to reduce the covering radius each time by as much as possible.

Remark: arbitrarily bad!



Center Selection: Greedy Algorithm

Repeatedly choose next center to be site farthest from any existing center.

Greedy-Center-Selection $(k, n, s_1, \dots, s_n)$

$C \leftarrow \emptyset.$

Repeat k times

  Select a site s_i with maximum distance $dist(s_i, C)$.

  $C \leftarrow C \cup s_i$.

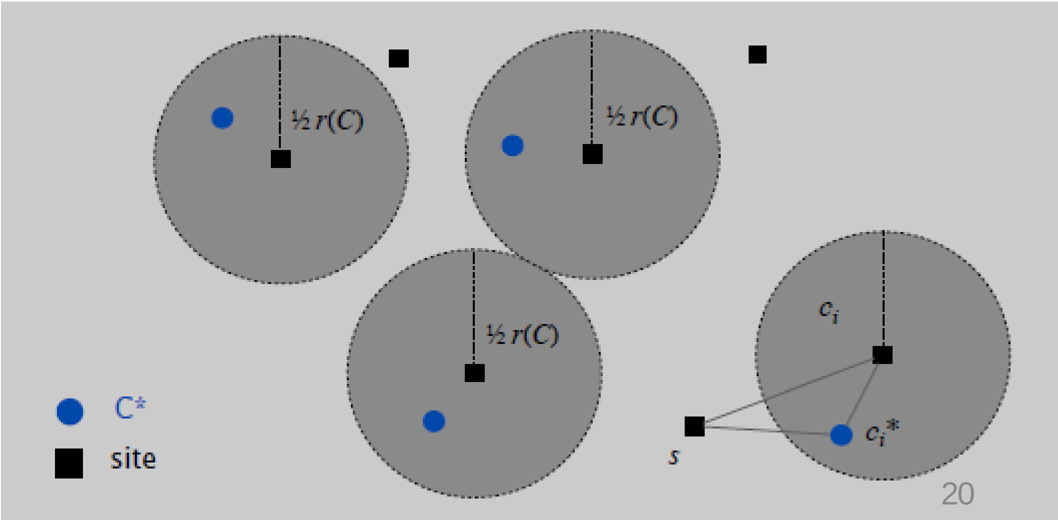
Return C.

Center Selection: Analysis of Greedy Algorithm

Lemma. Let C^* be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

Pf. [by contradiction] Assume $r(C^*) \leq \frac{1}{2} r(C)$.

- For each site $c_i \in C$, consider ball of radius $\frac{1}{2} r(C)$ around it.
- Exactly one c_i^* in each ball; let c_i be the site paired with c_i^*.
- Consider any site s and its closest center $c_i^* \in C^*$.
- $dist(s, C) \leq dist(s, c_i) \leq dist(s, c_i^*) + dist(c_i^*, c_i) \leq 2r(C^*)$.
- Thus, $r(C) \leq 2r(C^*)$.



Center Selection

Lemma. Let C^* be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

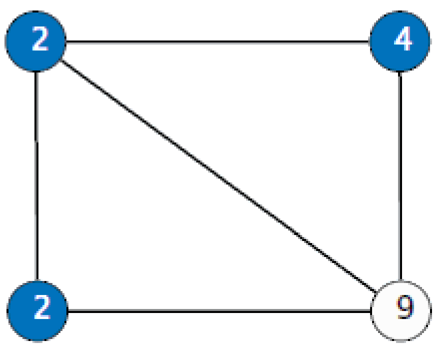
Theorem. Greedy algorithm is a 2-approximation for center selection problem.

Remark. Greedy algorithm always places centers at sites, but is still within a factor of 2 of best solution that is allowed to place centers anywhere.

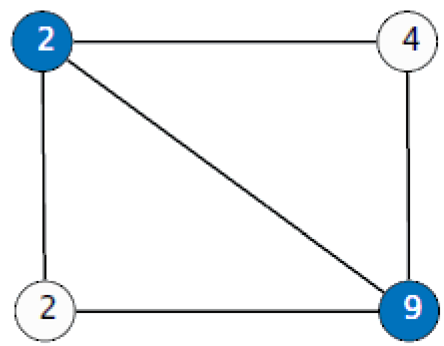
Weighted Vertex Cover

Definition. Given a graph $G = (V, E)$, a vertex cover is a set of $S \subseteq V$ such that each edge in E has at least one end in S.

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



weight = 2 + 2 + 4



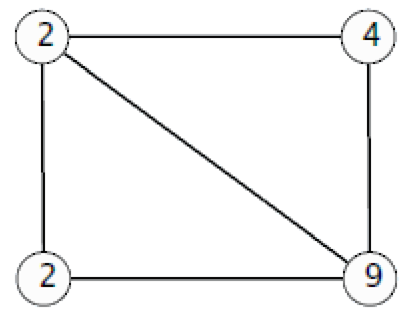
weight = 11

Pricing Method

Pricing method. Each edge must be covered by some vertex. Edge $e = (i, j)$ pays price $p_e \geq 0$ to use both vertex i and j.

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

$$\text{for each vertex } i: \sum_{e=(i,j)} p_e \leq w_i$$



Fairness lemma. For any vertex cover S and any fair prices p_e: $\sum_{e \in E} p_e \leq w(S)$.

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

Pricing Method

Set prices and find vertex cover simultaneously.

Weighted-Vertex-Cover (G, w)

$S \leftarrow \emptyset$.

For each $e \in E$

$p_e \leftarrow 0$.

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

While (there exists an edge (i, j) such that neither i nor j is tight)

Select such an edge $e = (i, j)$.

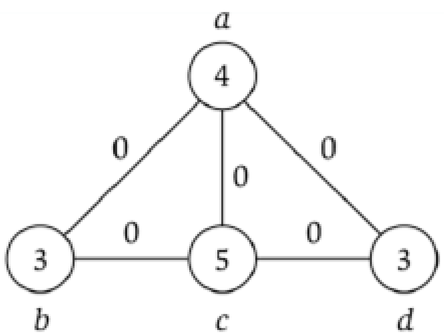
Increase p_e as much as possible until i or j tight.

$S \leftarrow$ set of all tight nodes.

Return S.

Pricing Method Example

Ex.



Weighted-Vertex-Cover (G, w)

$S \leftarrow \emptyset.$

For each $e \in E$

$p_e \leftarrow 0.$

While (there exists an edge (i, j) such that neither i nor j is tight)

Select such an edge $e = (i, j)$.

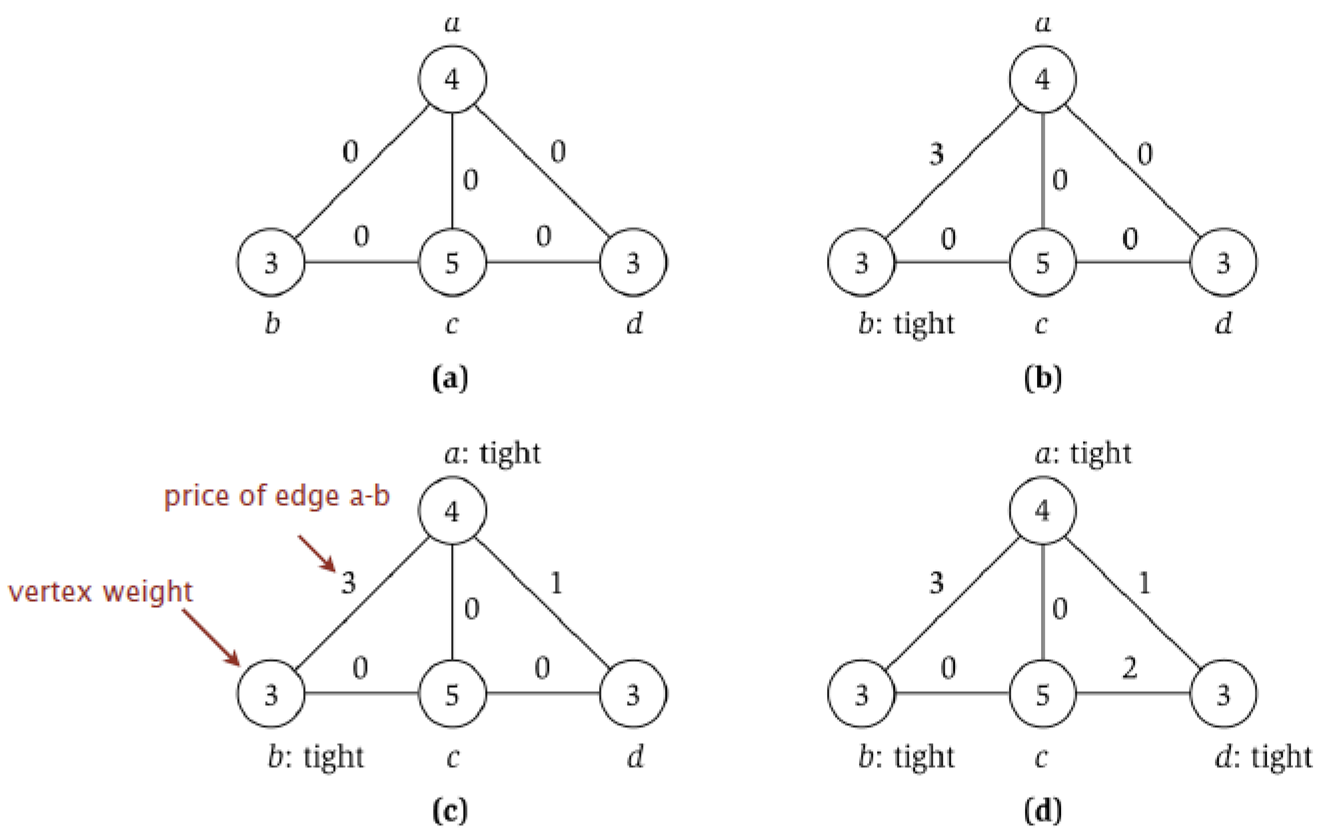
Increase p_e as much as possible until i or j tight.

$S \leftarrow$ set of all tight nodes.

Return S.

Pricing Method Example

Ex.



Pricing Method: Analysis

Theorem. Pricing method is a 2-approximation for Weighted-Vertex-Cover.

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 or j is tight. But then "while" loop would not terminate.

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

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e$$
$$\leq 2w(S^*)$$

Weighted Vertex Cover: ILP Formulation

Given a graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a min-weight subset of vertices $S \subseteq V$ such that every edge is incident to at least one vertex in S.

Integer Linear Programming (ILP) formulation.

- Model inclusion of each vertex i using a 0/1 variable x_i.

$$x_i = \begin{cases} 0, if \text{ vertex } i \text{ is not in vertex cover} \\ 1, if \text{ vertex } i \text{ is in vertex cover} \end{cases}$$

Vertex covers in 1-1 correspondence with 0/1 assignments: $S = \{i \in V\colon\ x_i = 1\}$.

- Objective function: minimize $\sum_i w_i x_i$.
- For every edge (i, j), take either vertex i or j (or both): $x_i + x_j \geq 1$.

Weighted Vertex Cover: ILP Formulation

Weighted vertex cover. Integer linear programming formulation.

$$
\begin{aligned}
&(ILP)\ \min \textstyle\sum_{i \in V} w_i x_i \\
&s.t.\ \ x_i + x_j \geq 1 \qquad (i,j) \in E \\
&x_i \in \{0,1\} \qquad\qquad\ \ i \in V
\end{aligned}
$$

Observation. If x^* is optimal solution on ILP, then $S = \{i \in V\colon\ x_i^* = 1\}$ is a min-weight vertex cover.

Integer Linear Programming

Given integers a_{ij}, b_i, and c_j, find integers x_j that satisfy:

$$
\begin{aligned}
&\min c^T x \\
&s.t.\ \ Ax \geq b \\
&x \geq 0 \\
&x \ \ integral
\end{aligned}
$$

$$
\begin{aligned}
&\min \textstyle\sum_{j=1}^{n} c_j x_j \\
&s.t.\ \ \textstyle\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad 1 \leq i \leq m \\
&x_j \geq 0 \qquad\qquad 1 \leq j \leq n \\
&x_j \ \ integral \quad 1 \leq j \leq n
\end{aligned}
$$

Linear Programming

Given integers a_{ij}, b_i, and c_j, find real numbers x_j that satisfy:

$$\begin{aligned} &\min c^T x \\ &s.t.\ \ Ax \geq b \\ &x \geq 0 \end{aligned}$$

$$\begin{aligned} &\min \textstyle\sum_{j=1}^{n} c_j x_j \\ &s.t.\ \ \textstyle\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad && 1 \leq i \leq m \\ &x_j \geq 0 && 1 \leq j \leq n \end{aligned}$$

Linear. No x^2, xy, $\arccos(x)$, $x(1-x)$, etc.

Simplex algorithm. Can solve LP in practice.

Weighted Vertex Cover: LP Relaxation

Linear programming relaxation.

$$
\begin{aligned}
&(LP)\ \min \textstyle\sum_{i\in V} w_i x_i \\
&s.t.\ \ x_i + x_j \geq 1 \qquad (i,j) \in E \\
&x_i \geq 0 \qquad\qquad\qquad\ \ i \in V
\end{aligned}
$$

Note. LP is not equivalent to weighted vertex cover.

Q. How can solving LP help us find a low-weight vertex cover?
A. Solve LP and round fractional values.

Weighted Vertex Cover: LP Rounding Algorithm

Lemma. If x^* is optimal solution to LP, then $S = \{i \in V\colon\ x_i^* \geq 1/2\}$ is a vertex cover whose weight is at most twice the min possible weight.

Pf. [S is a vertex cover]

- Consider an edge $(i, j) \in E$.
- Since $x_i^* + x_j^* \geq 1$, either $x_i^* \geq 1/2$ or $x_j^* \geq 1/2$ (or both) $\Longrightarrow$ (i, j) covered.

Pf. [S has desired cost]

- Let $S^\#$ be optimal vertex cover. Then

$$\sum_{i \in S^\#} w_i \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i$$

Theorem. The rounding algorithm is a 2-apprimation algorithm.