CSE 6140/ CX 4140:

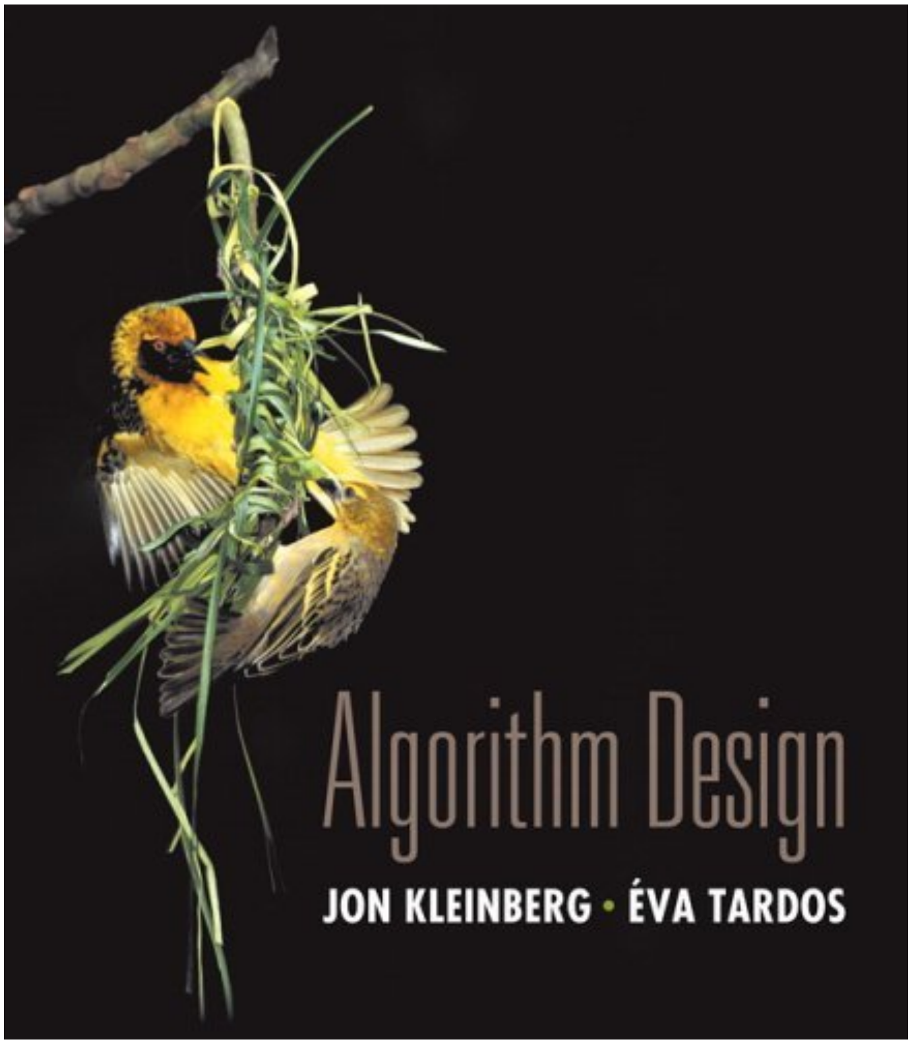
Computational Science and Engineering ALGORITHMS

Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina

KT 11.1 Load Balancing



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 J(i) be the subset of jobs assigned to machine i. The **load** of machine i is $L_i = \Sigma_{j \in J(i)} t_j$.

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: List Scheduling

List-scheduling algorithm.

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

```
List-Scheduling(m, n, t1,t2,…,tn) {
   for i = 1 to m {
      Li ← 0           ← load on machine i
      J(i) ← φ         ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink Lk         ← machine i has smallest load
      J(i) ← J(i) ∪ {j}      ← assign job j to machine i
      Li ← Li + tj           ← update load of machine i
   }
   return J(1), …, J(m)
}
```

Implementation. $O(n \log m)$ using a priority queue.

Load Balancing: List Scheduling Analysis

Theorem. [Graham, 1966] 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.

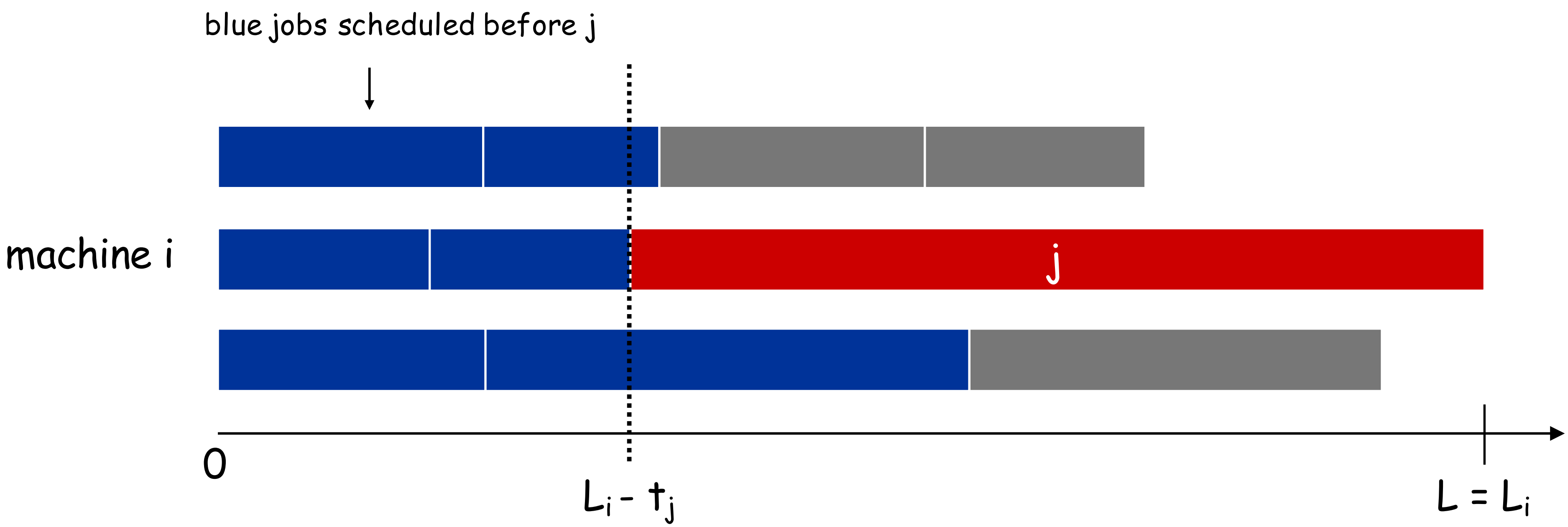
- The total processing time is $\Sigma_j t_j$.
- One of m machines must do at least a $1/m$ fraction of total work. ▪

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 had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.



Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i, i.e. $L_i = \max_k L_k$.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.
- Sum inequalities over all k and divide by m:

$$
\begin{aligned}
L_i - t_j &\leq \tfrac{1}{m} \sum_{k=1..m} L_k \\
&= \tfrac{1}{m} \sum_{s=1..n} t_s \\
\text{Lemma 1} \longrightarrow \quad &\leq L^*
\end{aligned}
$$

- Now $\quad L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq L^*} \leq 2L^*.$ ▪

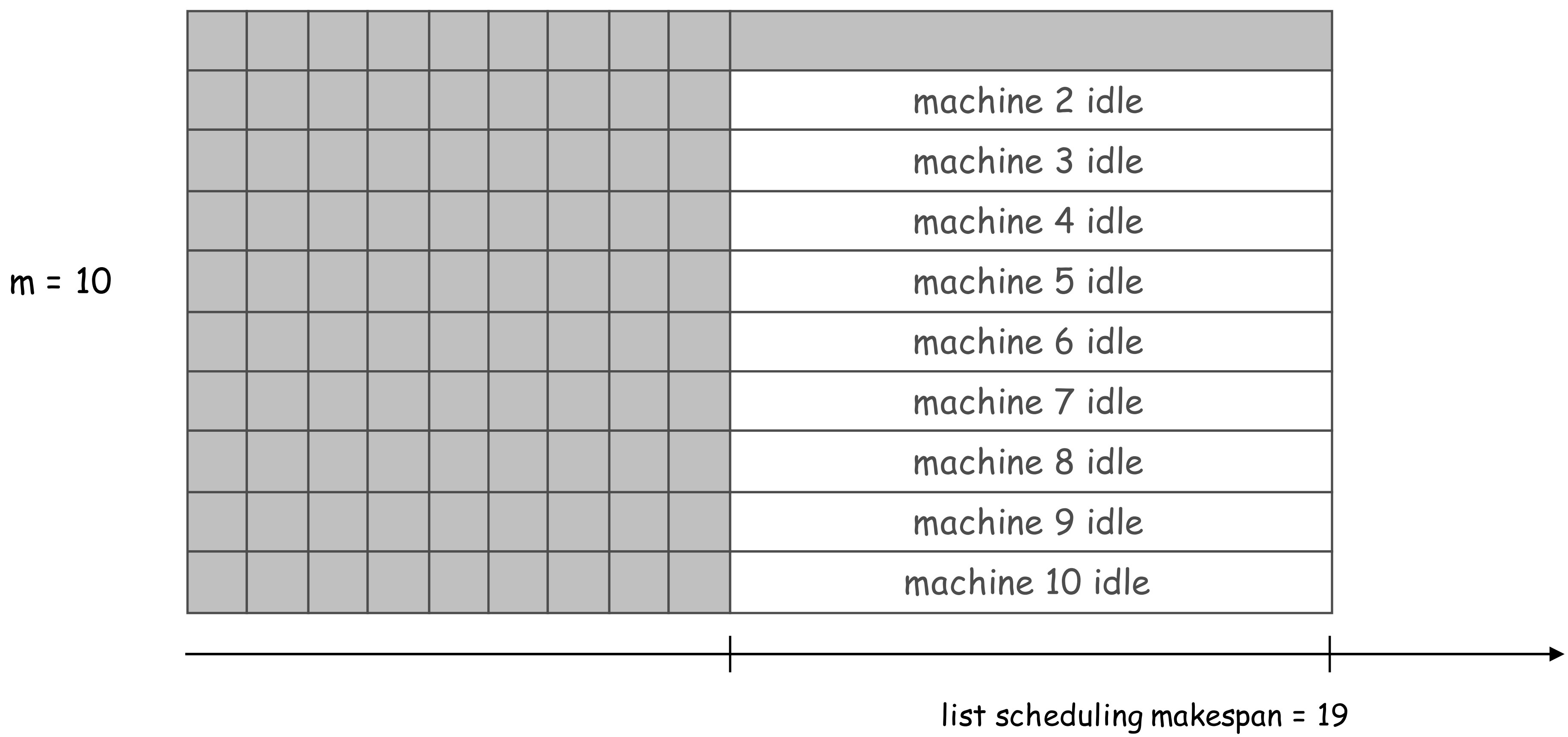
Lemma 2

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

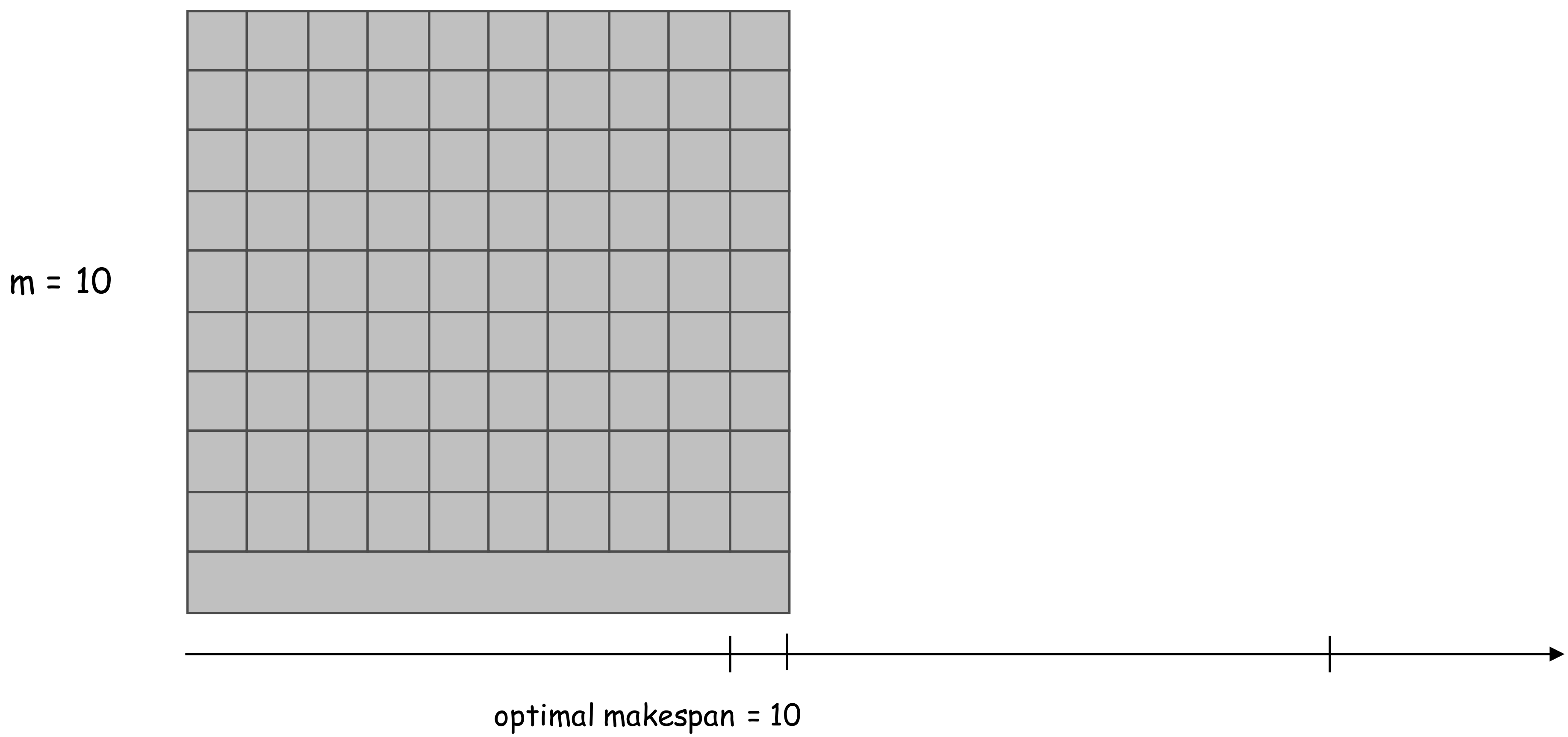
Ex: m machines, n=m(m-1)+1 jobs where m(m-1) jobs length 1 jobs, one job of length m



Load Balancing: List Scheduling Analysis

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

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



Load Balancing: LPT Rule (Offline Scheduling)

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

```
LPT-List-Scheduling(m, n, t1,t2,...,tn) {
   Sort jobs so that t1 ≥ t2 ≥ ... ≥ tn

   for i = 1 to m {
      Li ← 0                   ← load on machine i
      J(i) ← φ                 ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink Lk           ← machine i has smallest load
      J(i) ← J(i) ∪ {j}        ← assign job j to machine i
      Li ← Li + tj             ← update load of machine i
   }
   return J(1), …, J(m)
}
```

Load Balancing: LPT Rule

Observation. If at most m jobs, then list-scheduling is optimal.

Pf. Each job put on its own machine. ▪

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

Pf.

- Consider first m+1 jobs $t_1, \ldots, t_{m+1}$.
- Since the t_i's are in descending order, each takes at least t_{m+1} time.
- There are m+1 jobs and m machines, so by pigeonhole principle, in ANY solution at least one machine gets two jobs from the jobs $t_1, \ldots, t_{m+1}$. ▪

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

Pf. Same basic approach as for list scheduling.

$$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2} L^*} \leq \tfrac{3}{2} L^*. \quad \blacksquare$$

If machine i has 1 job, then job j <= m and hence j=1, $L_i = t_{max}$, and Greedy is optimal
Else in Greedy bottleneck machine i has at least 2 jobs, and hence j >= m+1 and $t_j \leq t_{m+1}$
Apply Lemma 3, $t_j \leq t_{m+1} \leq L^*/2$

Load Balancing: LPT Rule

Q. Is our 3/2 analysis tight?
A. No.

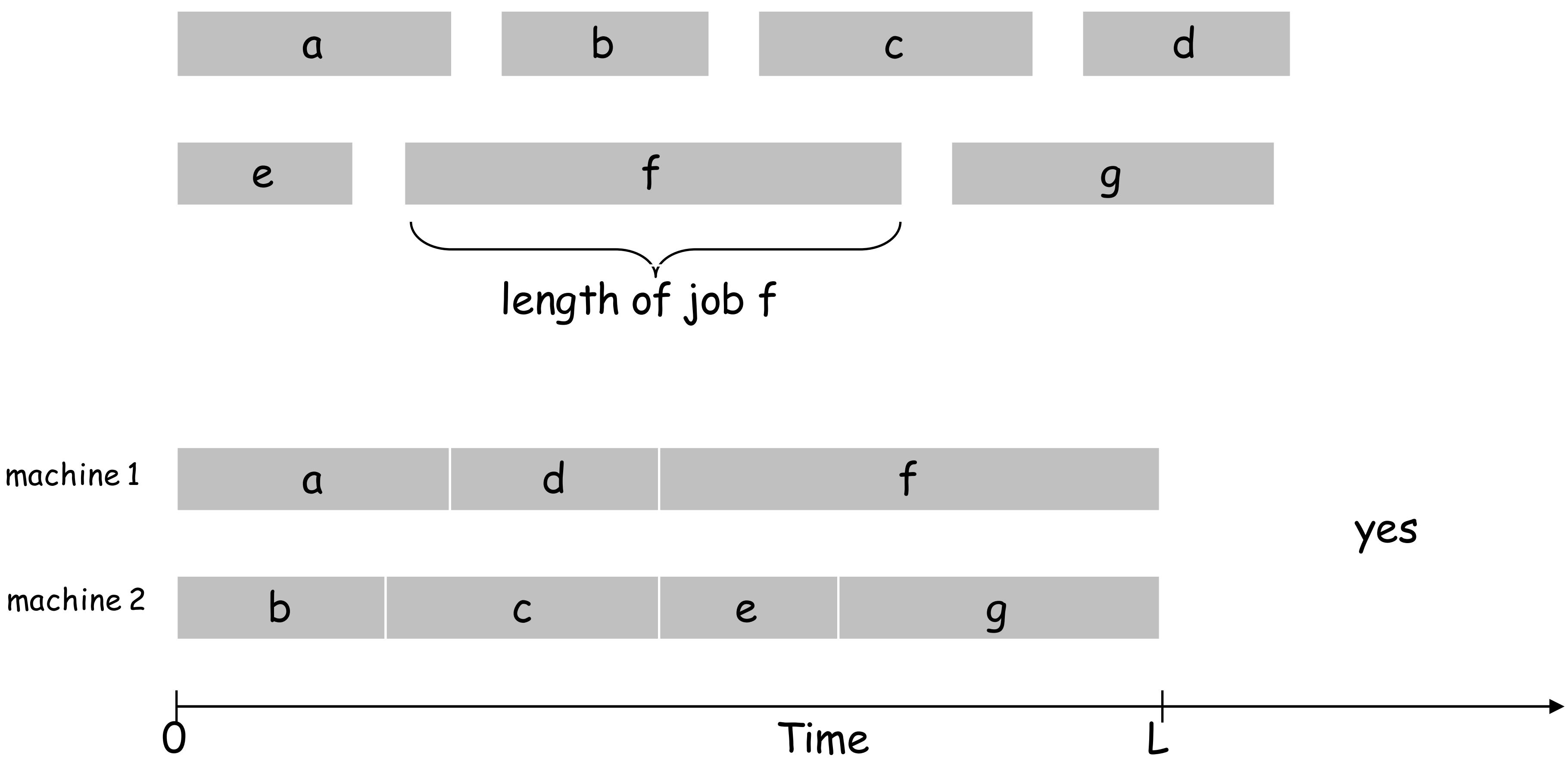
Theorem. [Graham, 1969] LPT rule is a 4/3-approximation.
Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's 4/3 analysis of LPT rule tight?
A. Essentially yes.

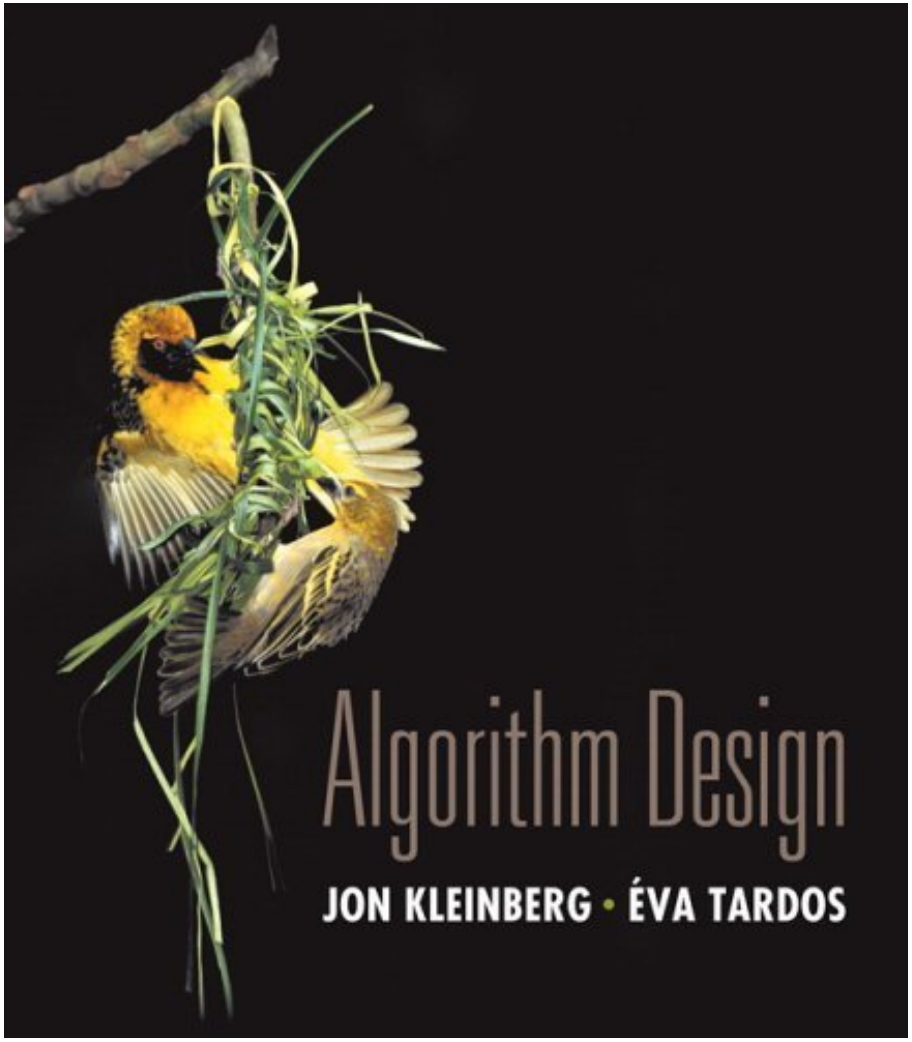
Load Balancing on 2 Machines

Claim. Load balancing is hard even if only 2 machines.

Pf. 2-PARTITION $\le_P$ LOAD-BALANCE.



KT 11.2 Clustering

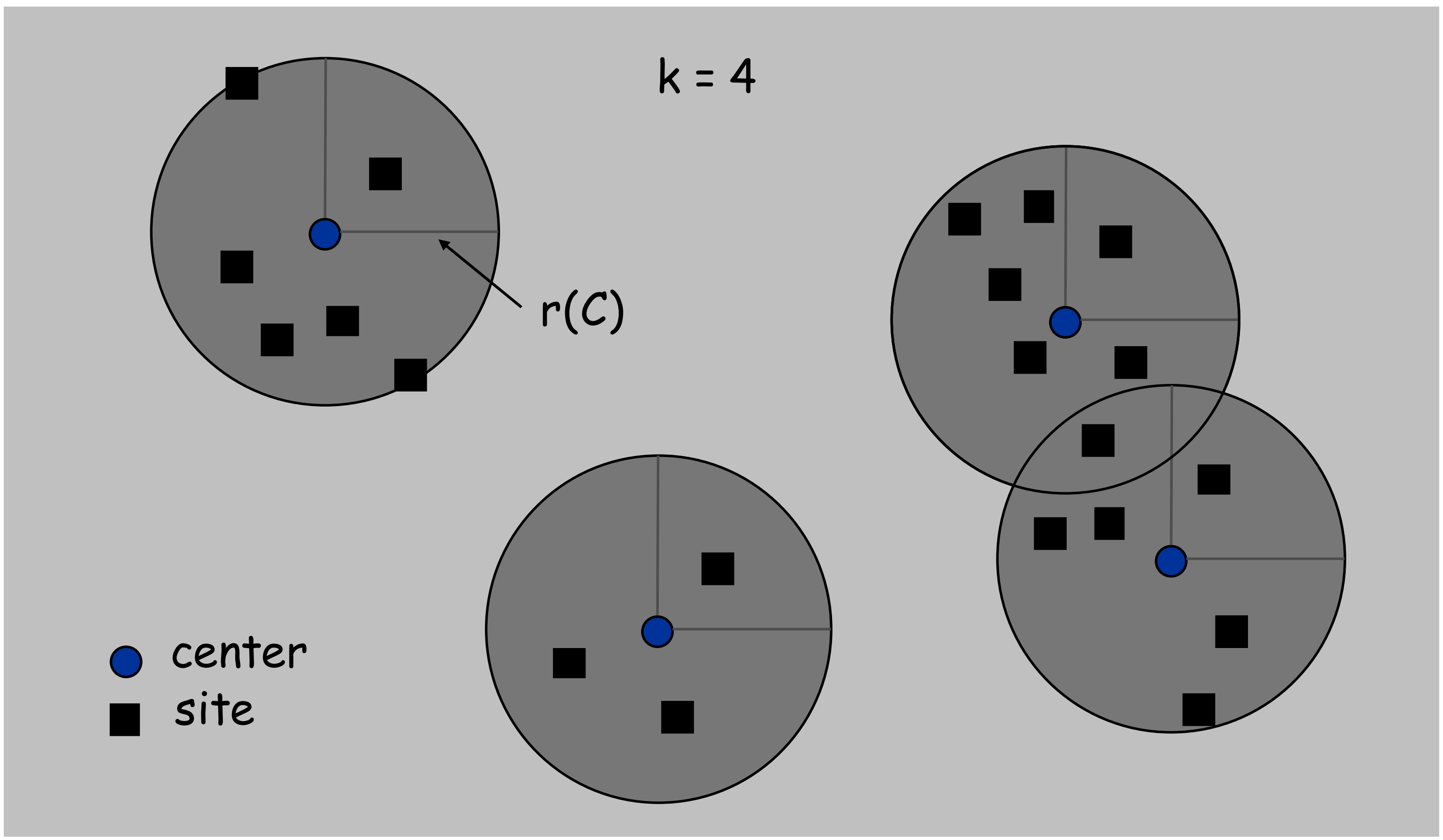


Center Selection Problem

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

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.

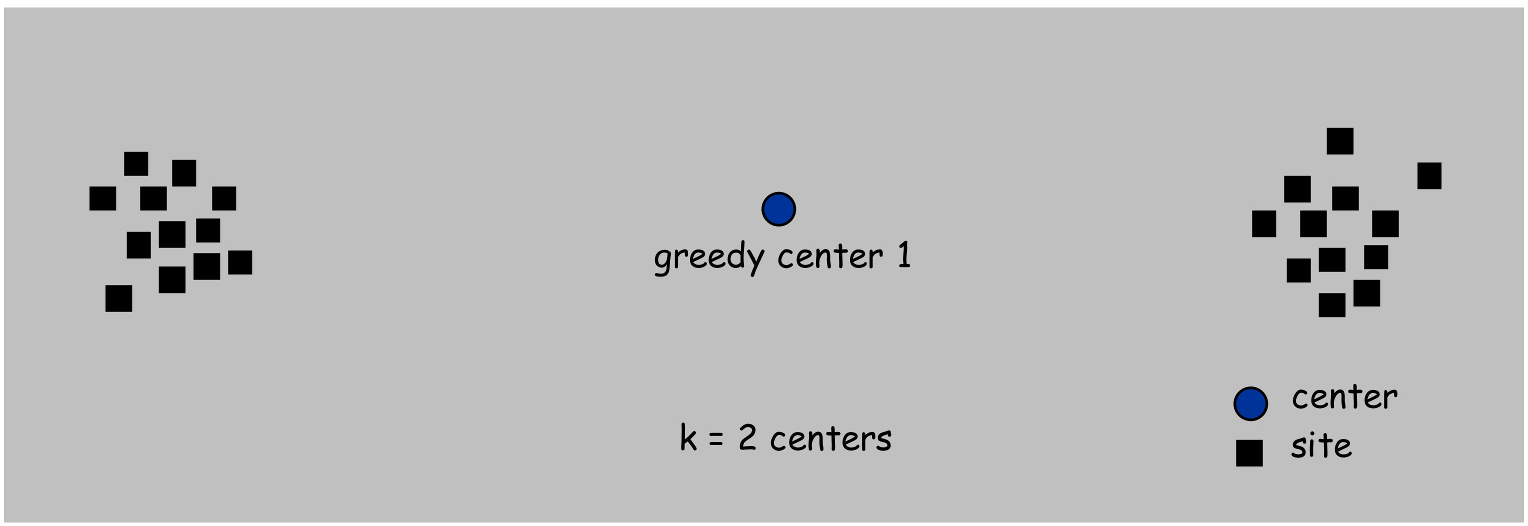
Application: where to put the branch offices w.r.t. clients?



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 Problem

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

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.

Notation.

- $\text{dist}(x, y)$ = distance between x and y.
- $\text{dist}(s_i, C) = \min_{c \in C} \text{dist}(s_i, c)$ = distance from s_i to closest center.
- $r(C) = \max_i \text{dist}(s_i, C)$ = smallest covering radius.

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

Distance function properties.

- $\text{dist}(x, x) = 0$ (identity)
- $\text{dist}(x, y) = \text{dist}(y, x)$ (symmetry)
- $\text{dist}(x, y) \le \text{dist}(x, z) + \text{dist}(z, y)$ (triangle inequality)

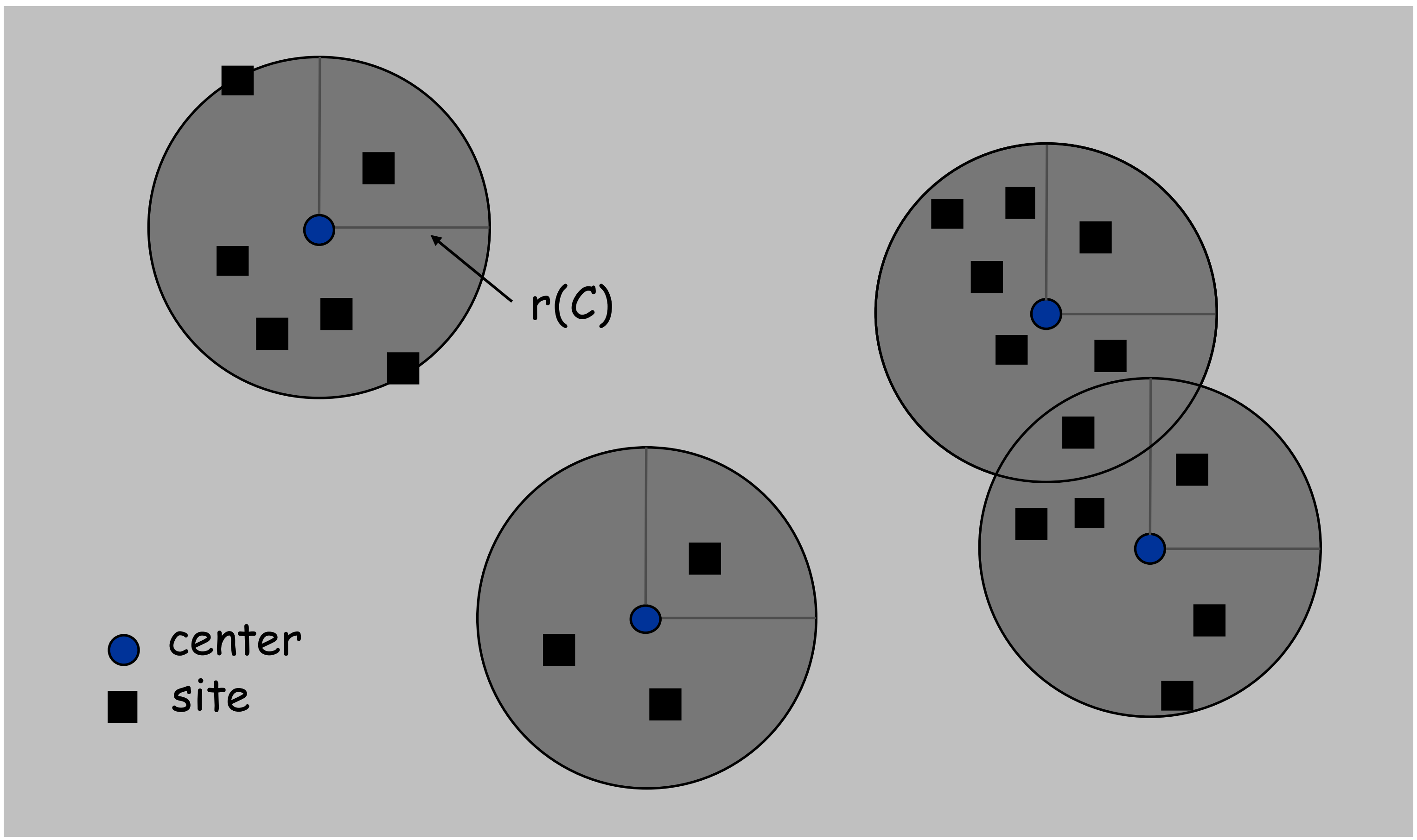
Also known as Metric Facility Location problem

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.

Ex: similarly in multi-dimensional space, where each site is a feature vector

Remark: search can be infinite!



Center Selection: Greedy Algorithm

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

```
Greedy-Center-Selection(k, n, s1,s2,…,sn) {

   C = φ
   repeat k times {
      Select a site si with maximum dist(si, C)
      Add si to C
   }                          site farthest from any center
   return C
}
```

Center Selection: Greedy Algorithm

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

Observation. Upon termination, all centers in C are pairwise at least $r(C)$ apart.

Pf.

Remember that $r(C) = \max_i \text{dist}(s_i, C)$

Let us call the point that achieves this maximum radius s'

- clearly s' is not one of the chosen centers, $k < n$
- s' is at least $r(C)$ away from any chosen center

Assume there are two centers c_i and c_j at distance $< r(C)$ (let $i < j$)

- when we chose j, its distance to the current C was $< r(C)$ due to c_i
- s' was an option to choose as center and it was at least $r(C)$ away from all centers in current C => s' is further than j

By construction of algorithm, we always choose the furthest point from the current C -> Contradiction

Center Selection: Analysis of Greedy Algorithm

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

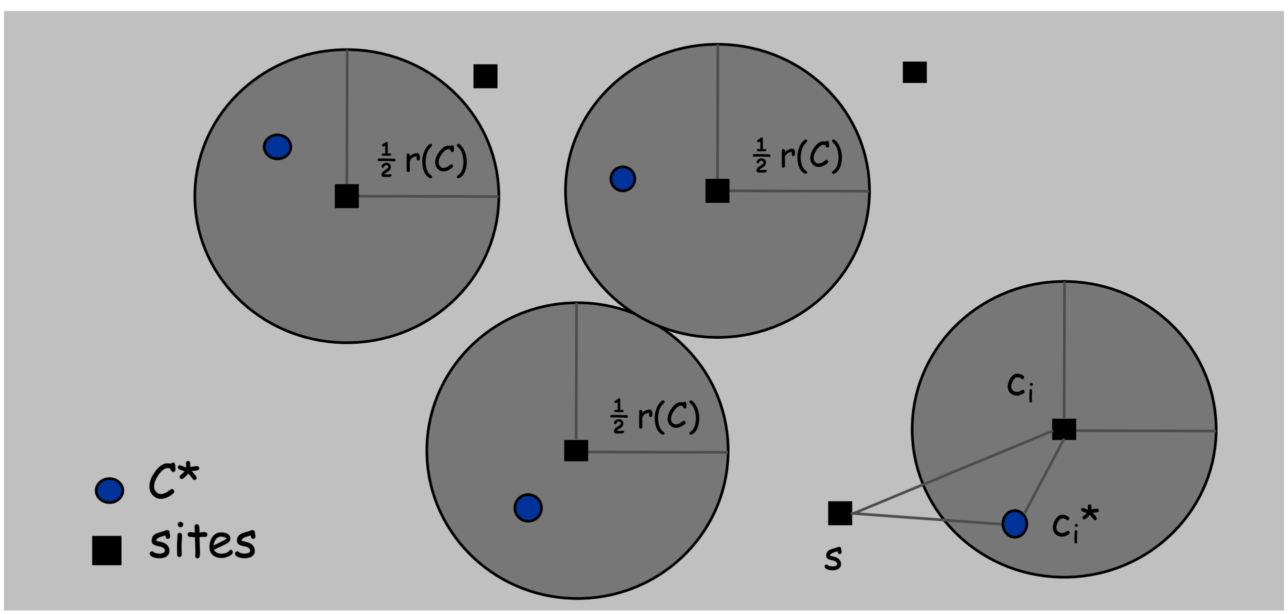
Pf. (by contradiction)

Assume $r(C) > 2r(C^*)$, i.e. $r(C^*) < \frac{1}{2}\, r(C)$.

- For each center c_i in C, consider ball of radius $\frac{1}{2}\, r(C)$ around it.
- $\text{dist}(c_i, C^*) \le r(C^*) < \frac{1}{2}\, r(C)$, so <u>at least</u> one c_i^* in each ball in C

By definition of radius

By our assumption

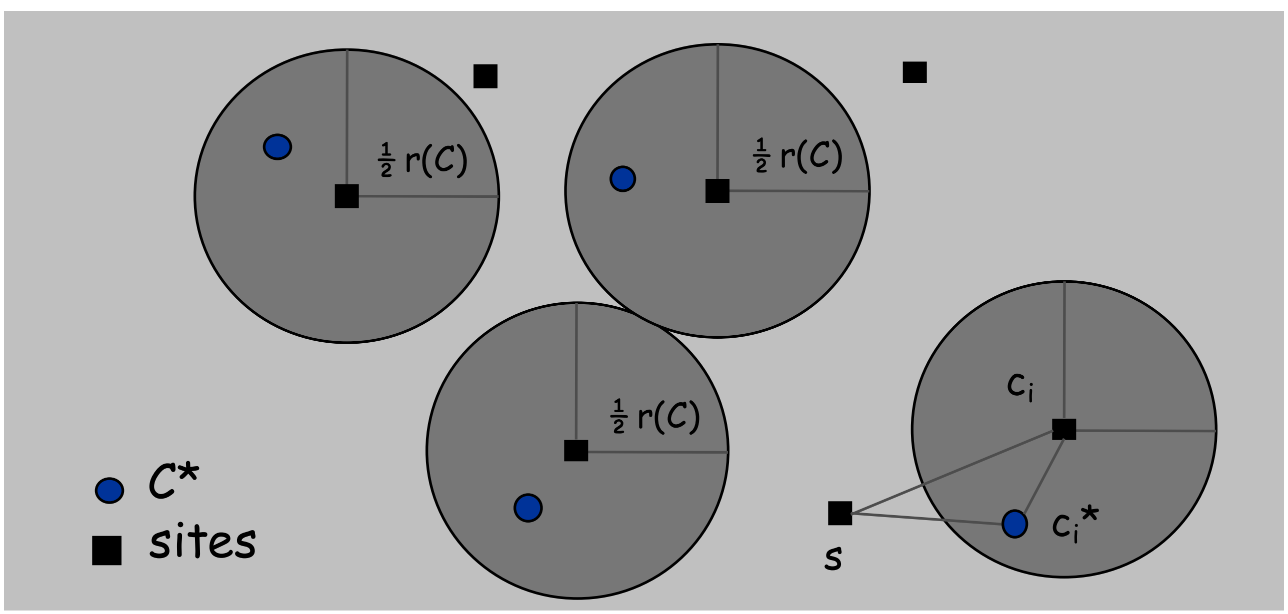


Center Selection: Analysis of Greedy Algorithm

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

Pf. (by contradiction) Assume $r(C) > 2r(C^*)$, i.e. $r(C^*) < \frac{1}{2} r(C)$.

- <u>at least</u> one c_i^* in each ball in C
- Every pair of c_i's in C are at least $r(C)$ apart (by alg.), so each ball around a c_i in C does not intersect with any other ball
- Each ball has at least one c_i^* and $|C|=|C^*|=k$, so <u>at most</u> one c_i^* in each ball
- Therefore <u>exactly one c_i^* in each ball</u>



Center Selection: Analysis of Greedy Algorithm

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

Pf. (by contradiction) Assume $r(C) > 2r(C^*)$, i.e. $r(C^*) < \frac{1}{2} r(C)$.

- exactly one c_i^* in each ball in C ; let c_i be the site paired with c_i^*
- Consider **<u>any</u>** site s and its closest center c_i^* in C^*.
- $\text{dist}(s, C) \le \text{dist}(s, c_i) \le \text{dist}(s, c_i^*) + \text{dist}(c_i^*, c_i) \le 2r(C^*)$.

  min across all c_i Δ-inequality $\le r(C^*)$ since c_i^* is closest center to both s and c_i

- true for any site s including the one that has $\text{dist}(s,C)=r(C)$
- Thus $r(C) \le 2r(C^*)$, this is a contradiction with our assumption ▪