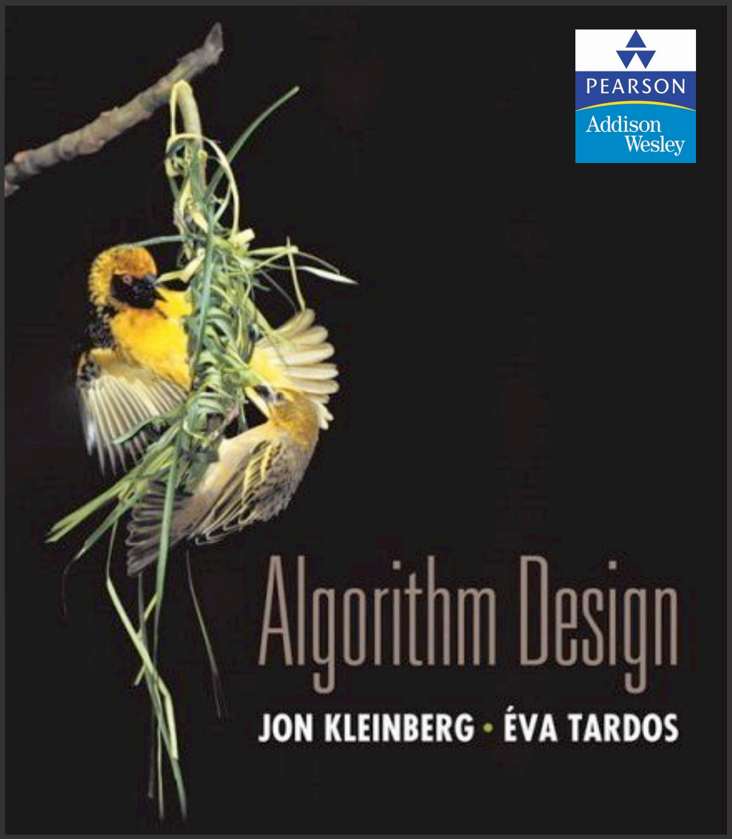


11. Approximation Algorithms

- *load balancing*
- *center selection*
- *pricing method: vertex cover*
- *LP rounding: vertex cover*
- *generalized load balancing*
- *knapsack problem*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Nov 8, 2014, 4:35 AM

Coping with NP-completeness

Q. Suppose I need to solve an **NP**-hard problem. What should I do?

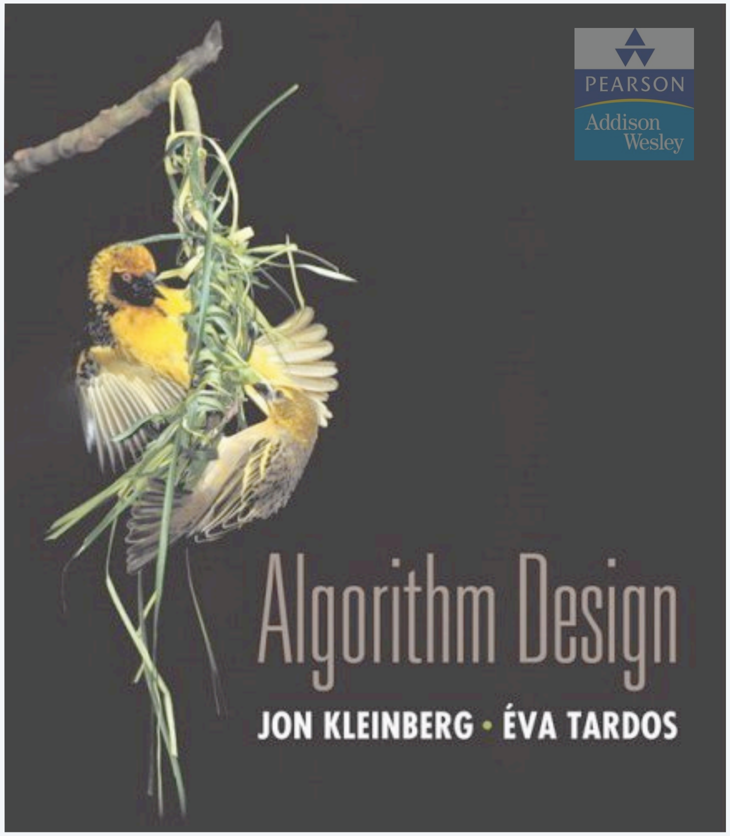
A. Sacrifice one of three desired features.

i. Solve arbitrary instances of the problem.
ii. Solve problem to optimality.
iii. Solve problem in polynomial time.

ρ-approximation algorithm.

- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

Challenge. Need to prove a solution's value is close to optimum, without even knowing what optimum value is



SECTION 11.1

11. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *pricing method: vertex cover*
- *LP rounding: vertex cover*
- *generalized load balancing*
- *knapsack problem*

Load balancing

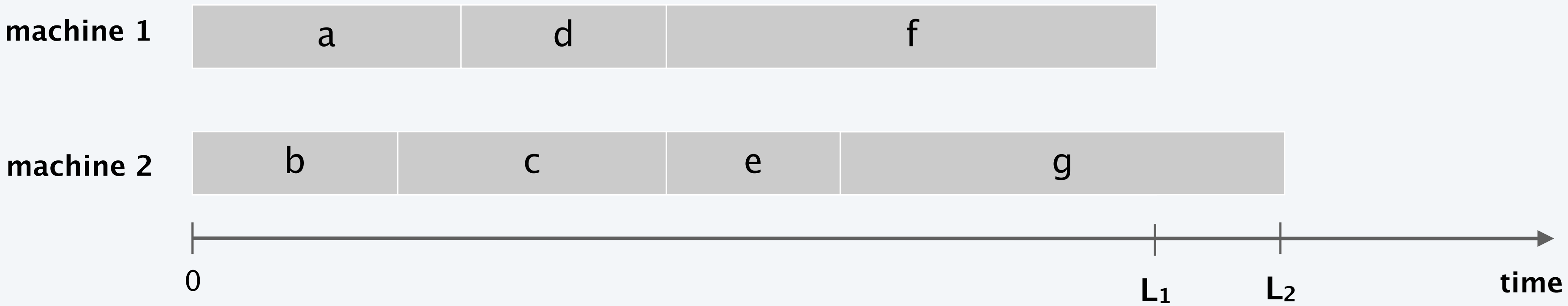
Input. m identical machines; n jobs, job j has processing time t_j.

- 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.

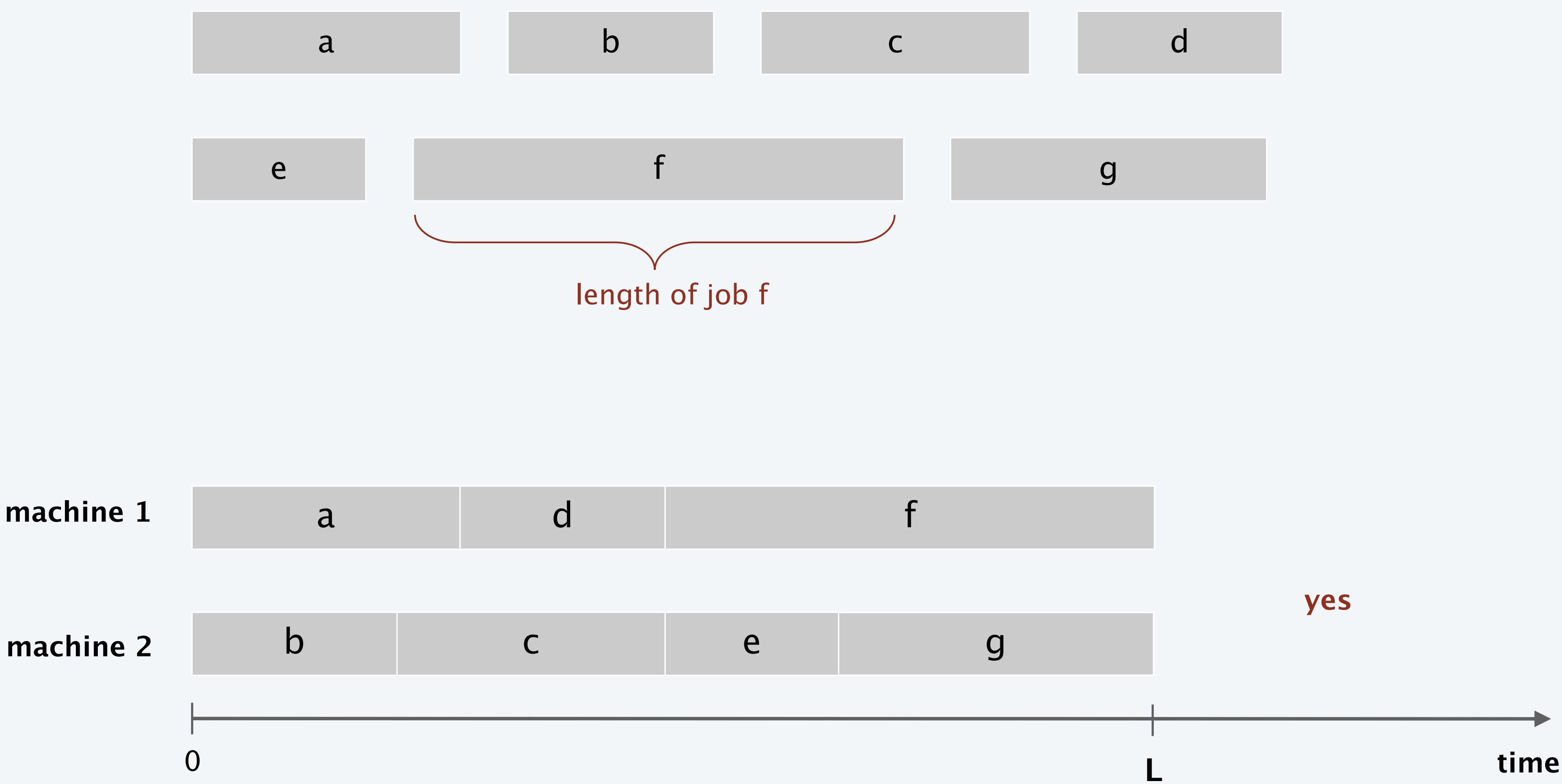


Load balancing on 2 machines is NP-hard

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

Pf. NUMBER-PARTITIONING $\le_P$ LOAD-BALANCE.

NP-complete by Exercise 8.26



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) ← ∅               ← 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 $L^* \geq \max_j t_j$.

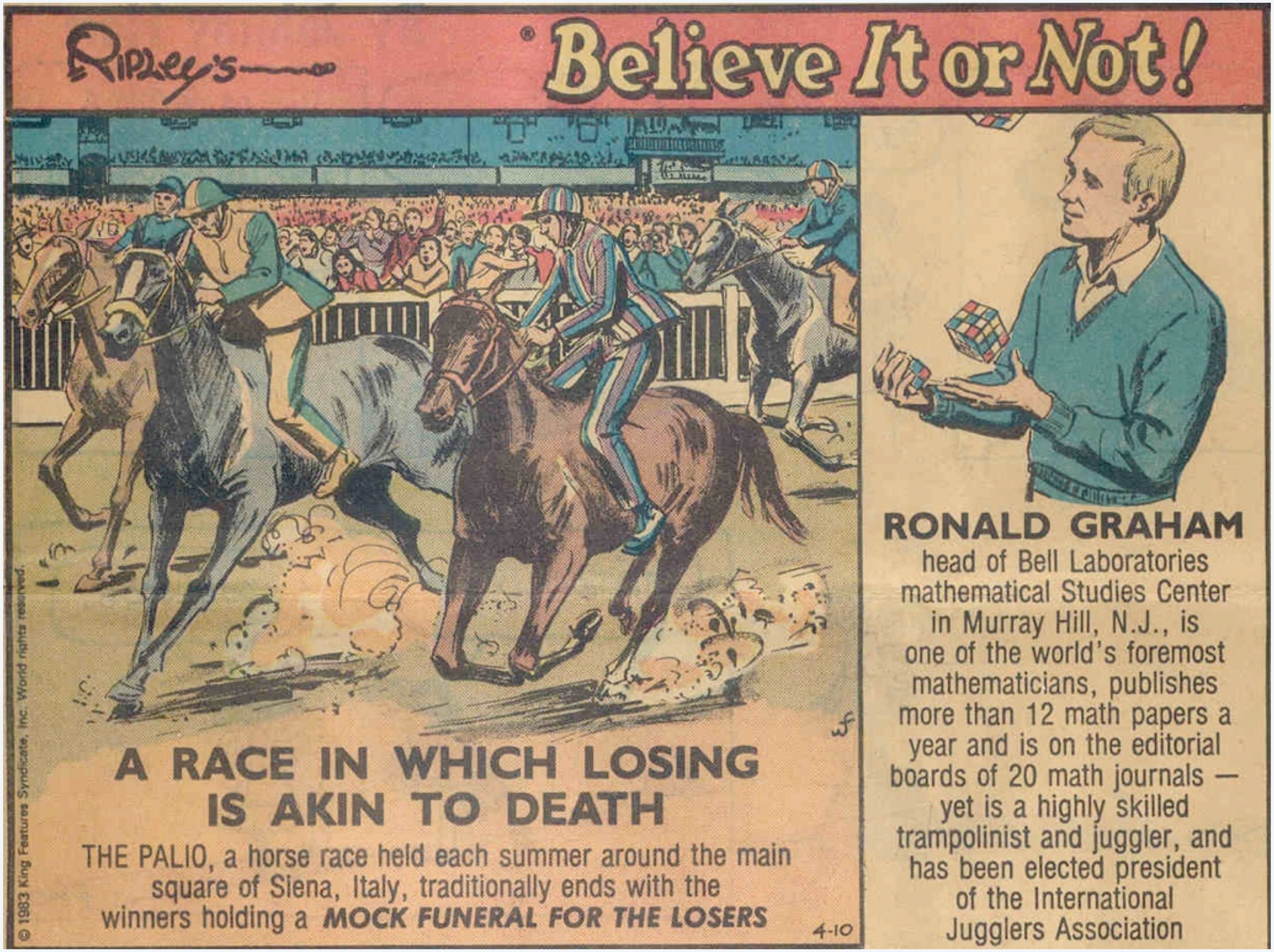
Pf. Some machine must process the most time-consuming job. ▪

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. ▪

Believe it or not

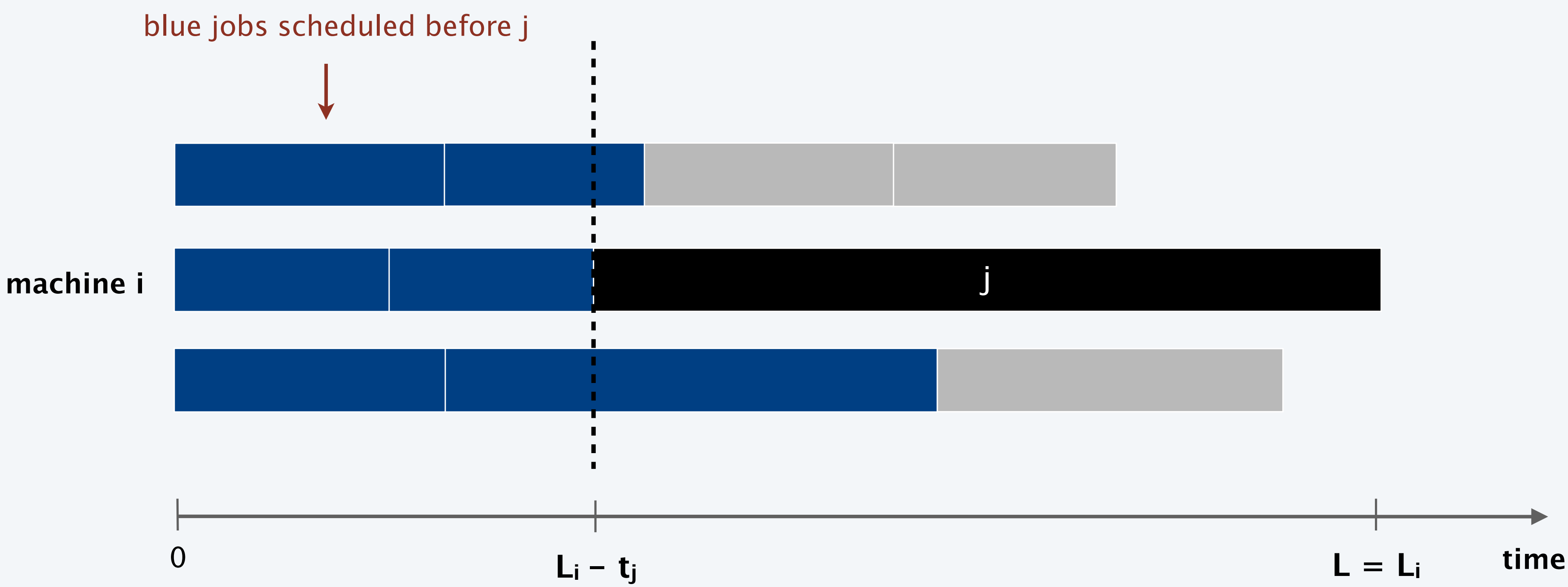


Load balancing: list scheduling analysis

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 $L_i - t_j \;\Rightarrow\; L_i - t_j \le L_k$ 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 had smallest load.
  Its load before assignment is $L_i - t_j \;\Rightarrow\; L_i - t_j \le L_k$ for all $1 \le k \le m$.
- Sum inequalities over all k and divide by m:

$$\begin{aligned} L_i - t_j &\le \tfrac{1}{m} \textstyle\sum_k L_k \\ &= \tfrac{1}{m} \textstyle\sum_k t_k \\ \text{Lemma 2} \longrightarrow \quad &\le L^* \end{aligned}$$

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

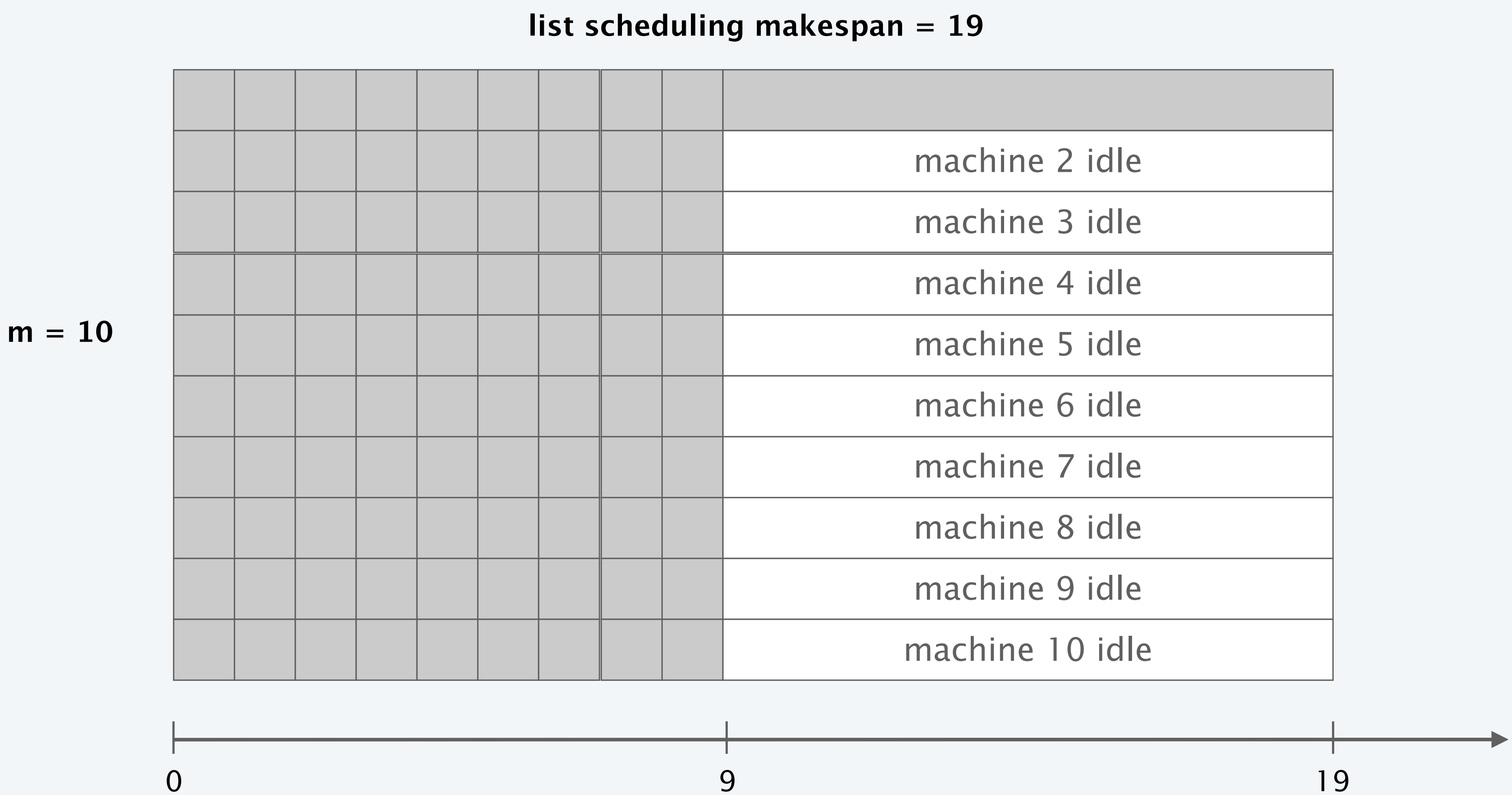
Lemma 1

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.

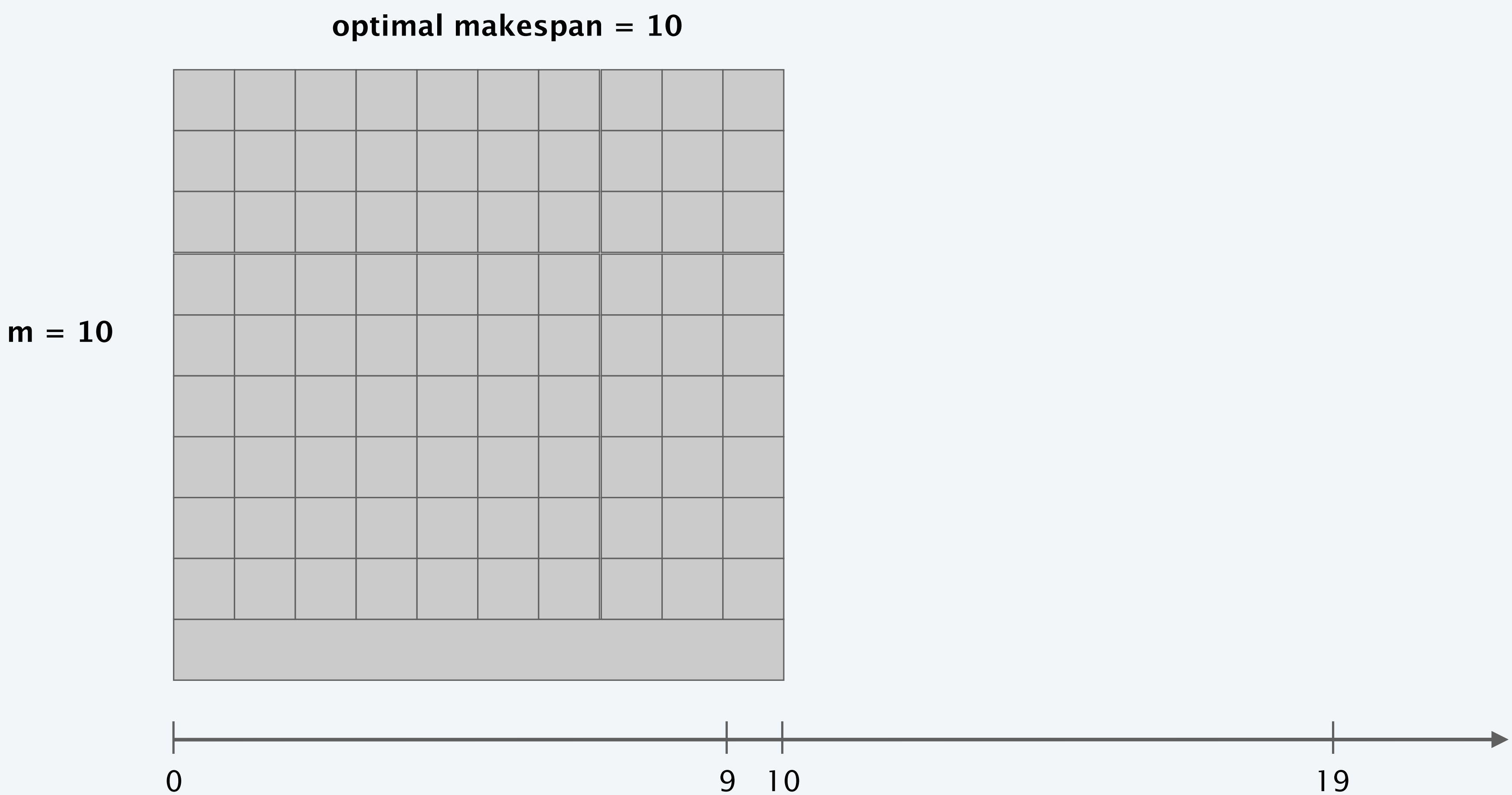


Load balancing: list scheduling analysis

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

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) ← ∅        ← 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, at least one machine gets two jobs. ▪

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

Pf. Same basic approach as for list scheduling.

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

Lemma 3

(by observation, can assume number of jobs > m)

Load Balancing: LPT rule

Q. Is our 3/2 analysis tight?

A. No.

Theorem. [Graham 1969] LPT rule is a 4/3-approximation.

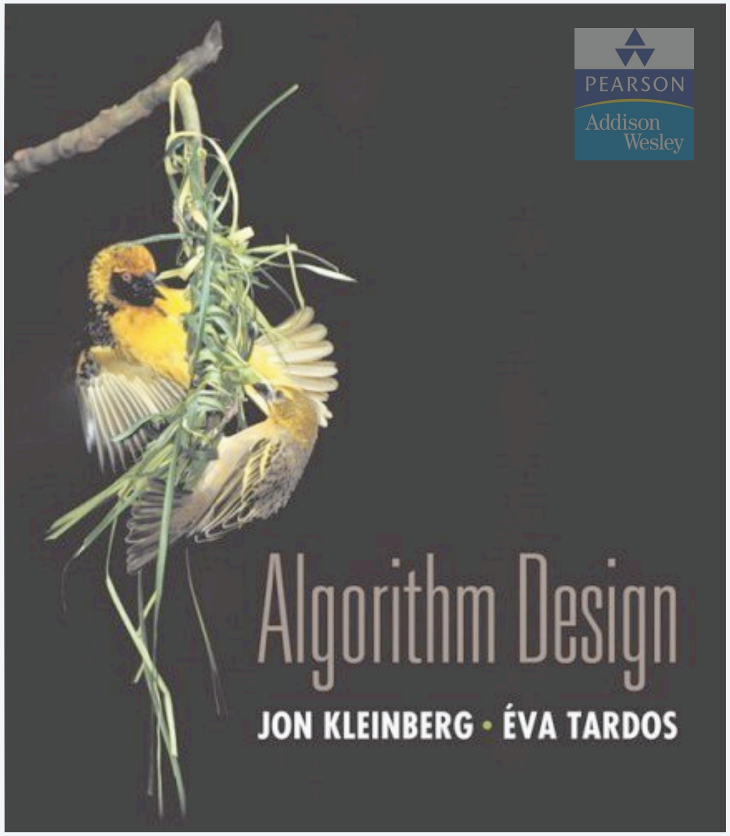
Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's 4/3 analysis tight?

A. Essentially yes.

Ex: m machines, $n = 2m + 1$ jobs, 2 jobs of length $m, m+1, \ldots, 2m-1$ and one more job of length m.

11. APPROXIMATION ALGORITHMS



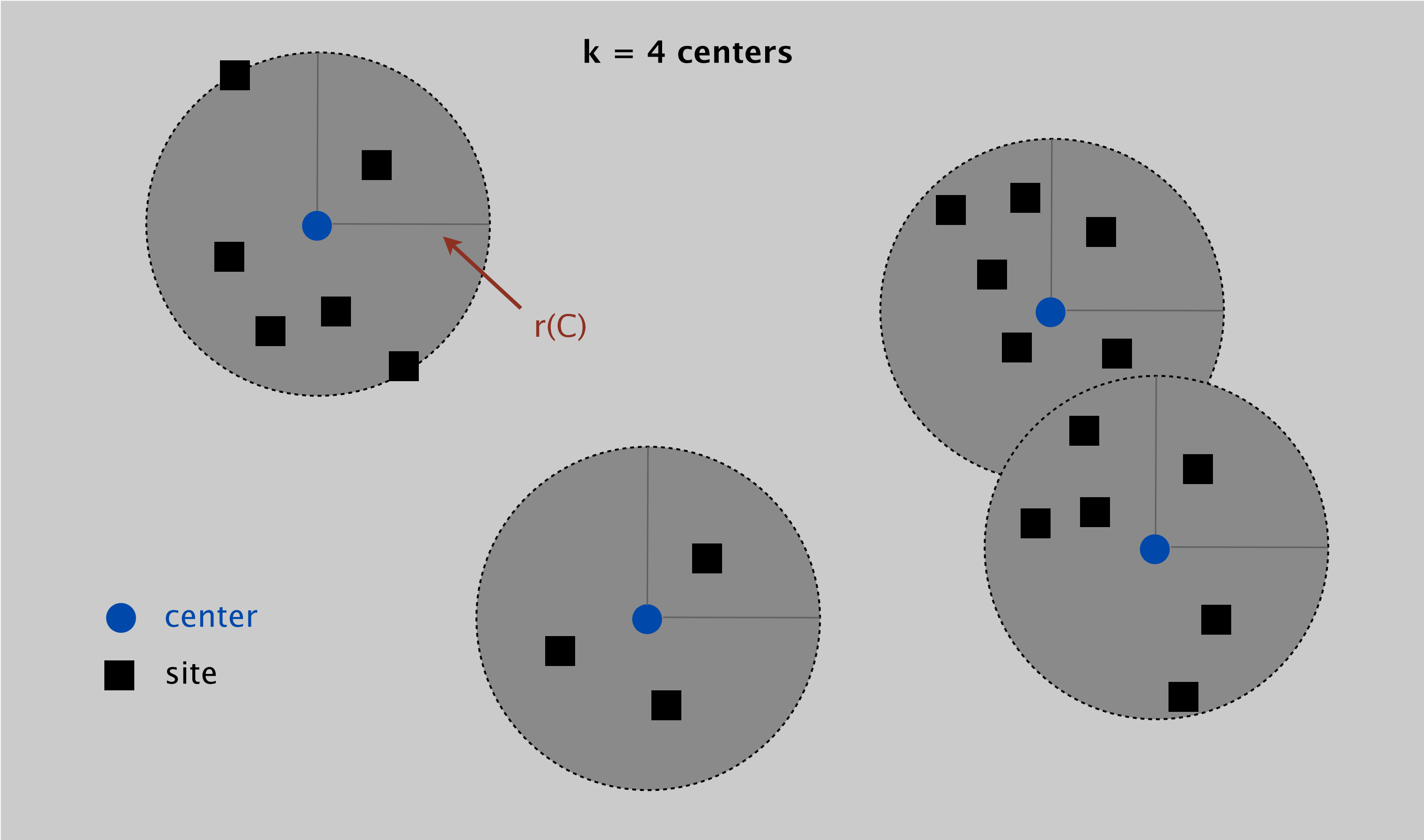
SECTION 11.2

- *load balancing*
- *center selection*
- *pricing method: vertex cover*
- *LP rounding: vertex cover*
- *generalized load balancing*
- *knapsack problem*

Center selection problem

Input. Set of n sites $s_1, \ldots, s_n$ 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 $s_1, \ldots, s_n$ 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.
- $dist(s_i, C) = \min_{c \in C} dist(s_i, c)$ = 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 $|C| = k$.

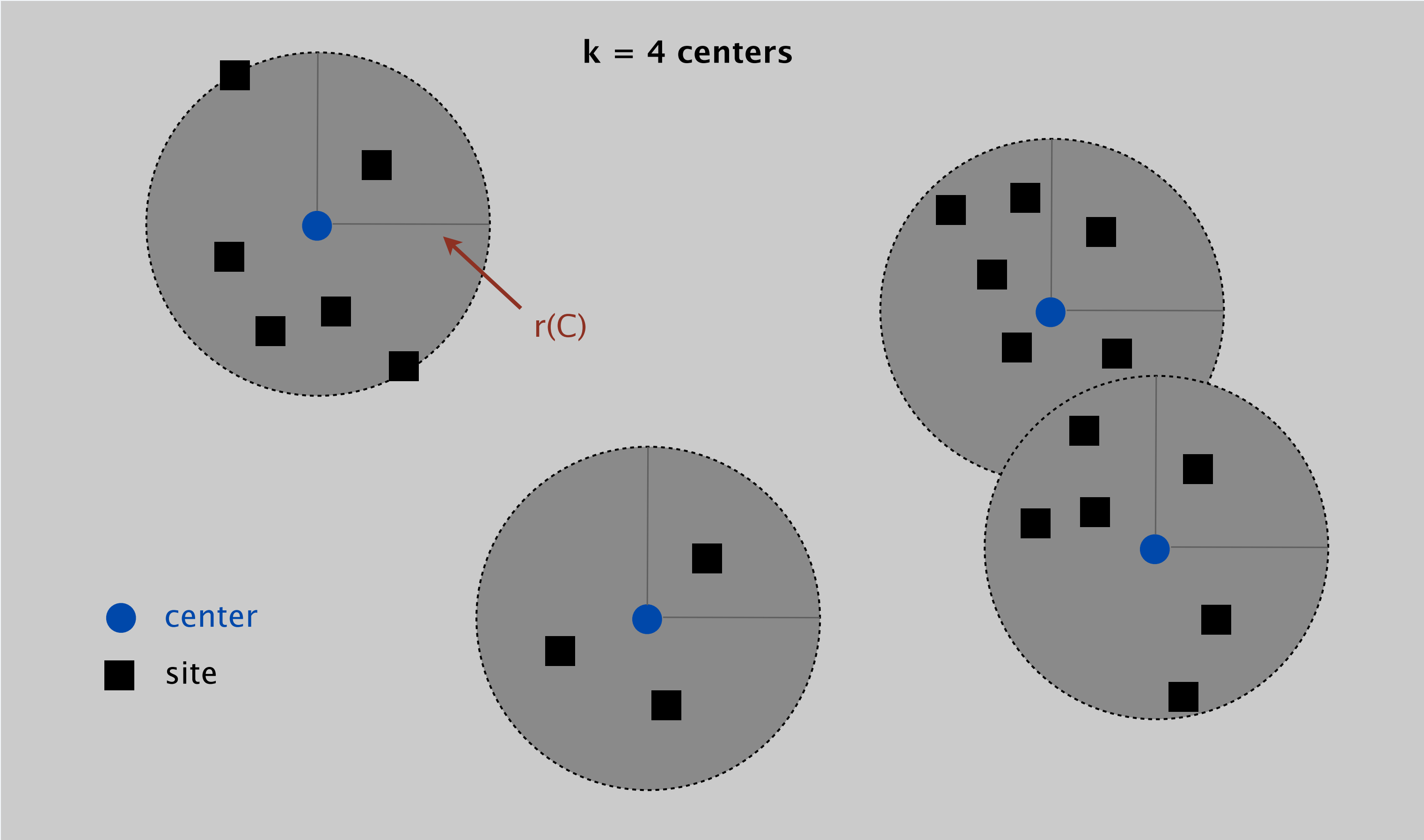
Distance function properties.

- $dist(x, x) = 0$ [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.

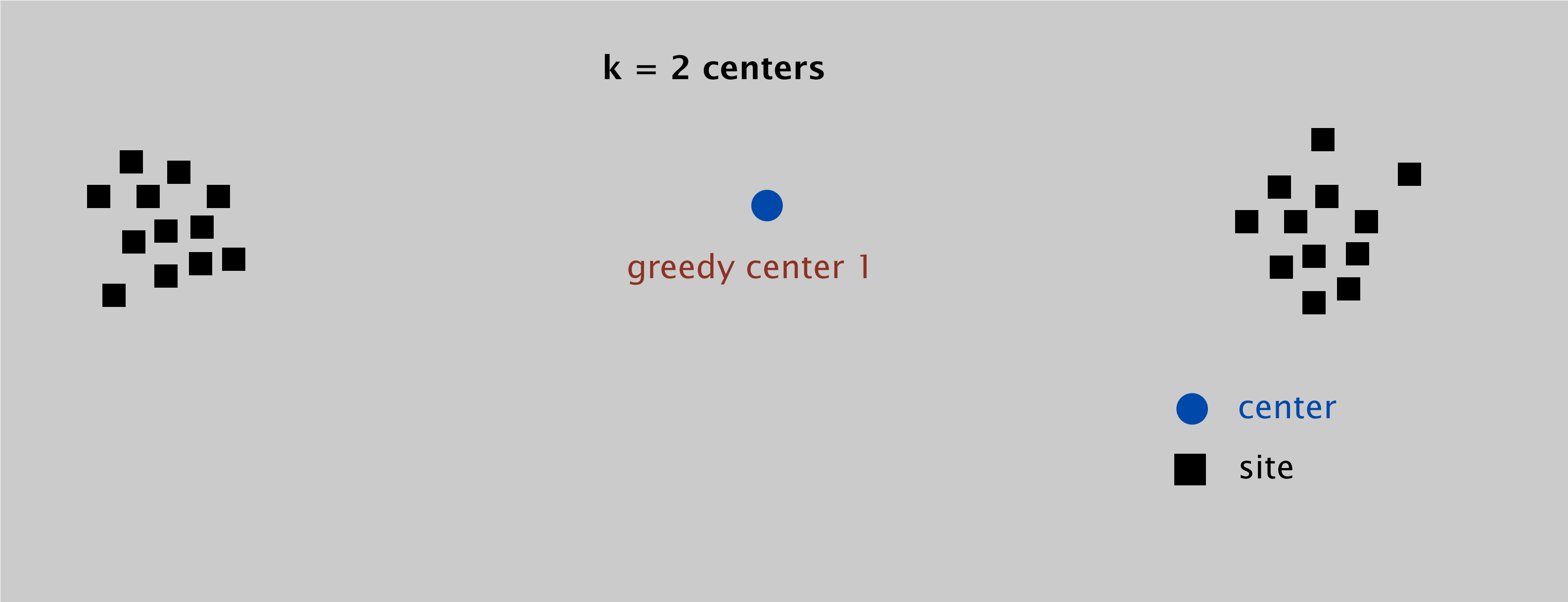
Remark: search can be infinite!



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 $(k, n, s_1, s_2, \ldots, s_n)$

$C \leftarrow \varnothing$.

REPEAT k times

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

    $C \leftarrow C \cup s_i$.

RETURN C.

site farthest from any center

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

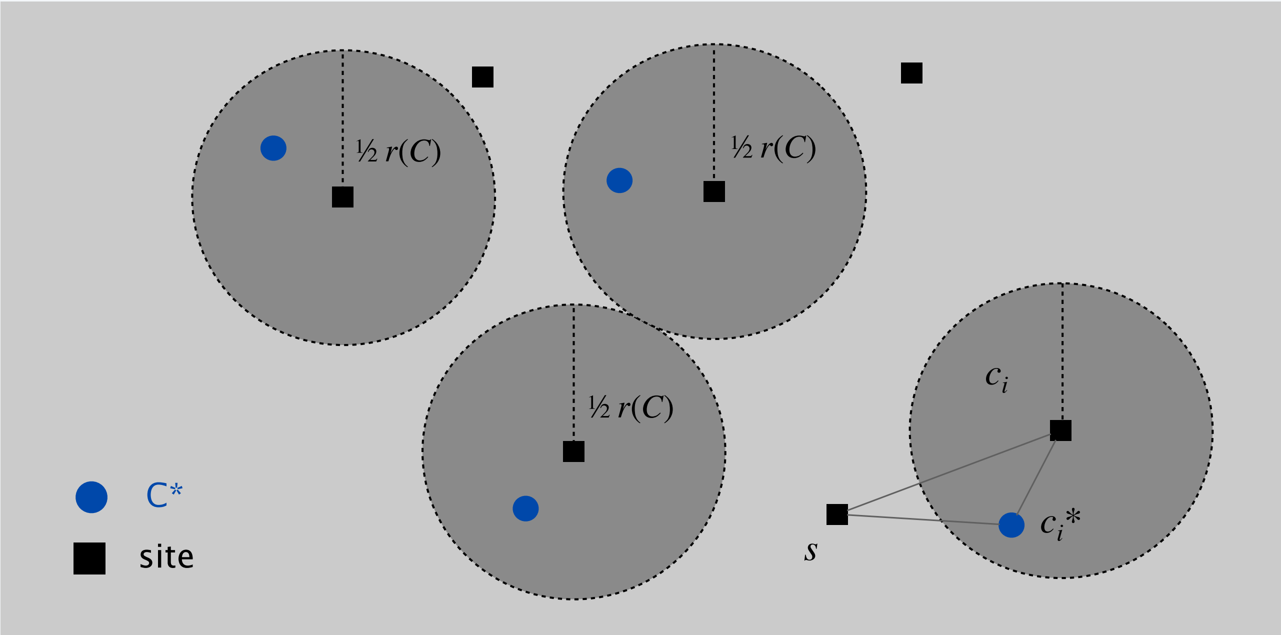
Pf. By construction of algorithm.

Center selection: analysis of greedy algorithm

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

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

- 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 $c_i^* \in C^*$.
- $dist(s, C) \le dist(s, c_i) \underbrace{\le}_{\Delta\text{-inequality}} \underbrace{dist(s, c_i^*) + dist(c_i^*, c_i)}_{\le\, r(C^*) \text{ since } c_i^* \text{ is closest center}} \le 2r(C^*)$.
- Thus, $r(C) \le 2r(C^*)$. ▪



Center selection

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

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.

e.g., points in the plane

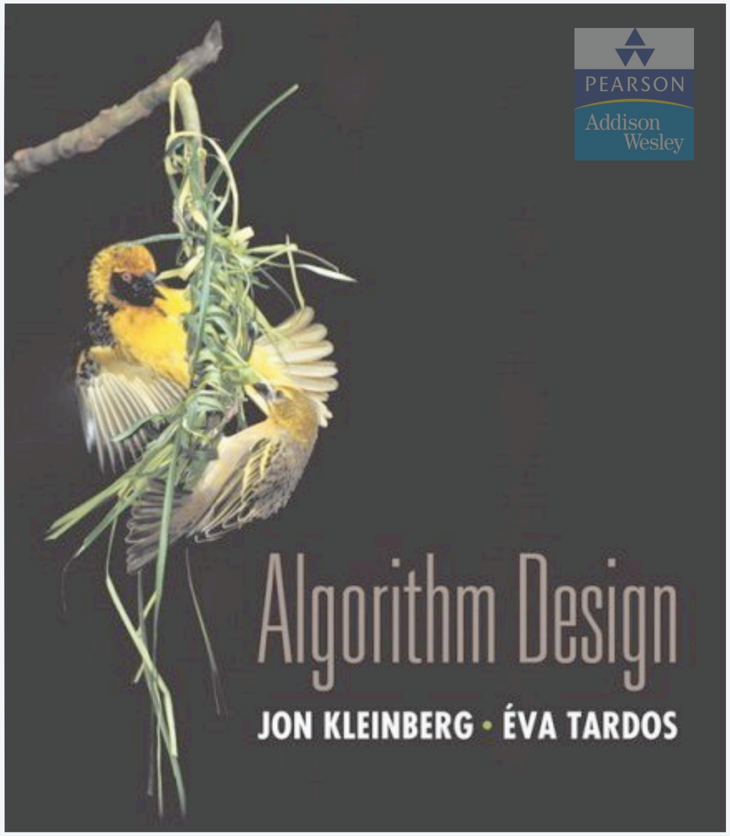
Question. Is there hope of a 3/2-approximation? 4/3?

Dominating set reduces to center selection

Theorem. Unless **P** = **NP**, there no ρ-approximation for center selection problem for any $\rho < 2$.

Pf. We show how we could use a $(2 - \varepsilon)$ approximation algorithm for CENTER-SELECTION selection to solve DOMINATING-SET in poly-time.

- Let $G = (V, E)$, k be an instance of DOMINATING-SET.
- Construct instance G' of CENTER-SELECTION with sites V and distances
  - $dist(u, v) = 1$ if $(u, v) \in E$
  - $dist(u, v) = 2$ if $(u, v) \notin E$
- Note that G' satisfies the triangle inequality.
- G has dominating set of size k iff there exists k centers C^* with $r(C^*) = 1$.
- Thus, if G has a dominating set of size k, a $(2 - \varepsilon)$-approximation algorithm for CENTER-SELECTION would find a solution C^* with $r(C^*) = 1$ since it cannot use any edge of distance 2. ▪



SECTION 11.4

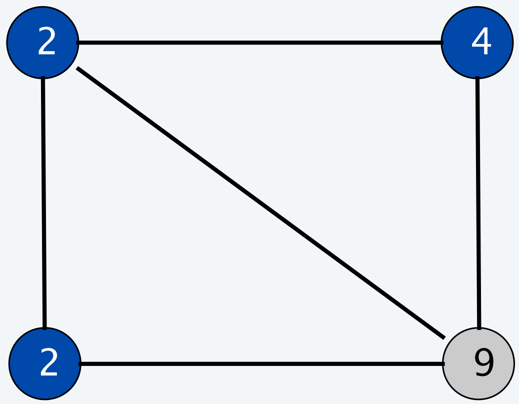
11. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *pricing method: vertex cover*
- *LP rounding: vertex cover*
- *generalized load balancing*
- *knapsack problem*

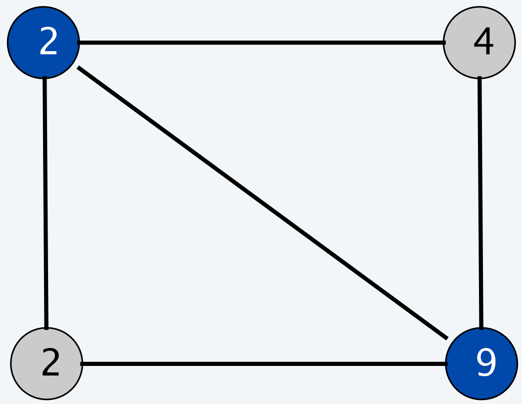
Weighted vertex cover

Definition. Given a graph $G = (V, E)$, a vertex cover is a set $S \subseteq V$ 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



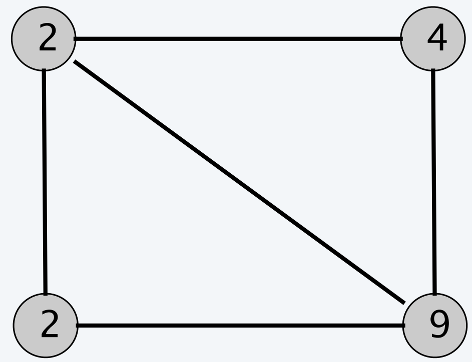
weight = 11

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.

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



Fairness lemma. For any vertex cover S and any fair prices p_e: $\sum_e p_e \leq w(S)$.

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

each edge e covered by at least one node in S

sum fairness inequalities for each node in S

Pricing method

Set prices and find vertex cover simultaneously.

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

WEIGHTED-VERTEX-COVER (G, w)

$S \leftarrow \varnothing$.

FOREACH $e \in E$

  $p_e \leftarrow 0$.

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

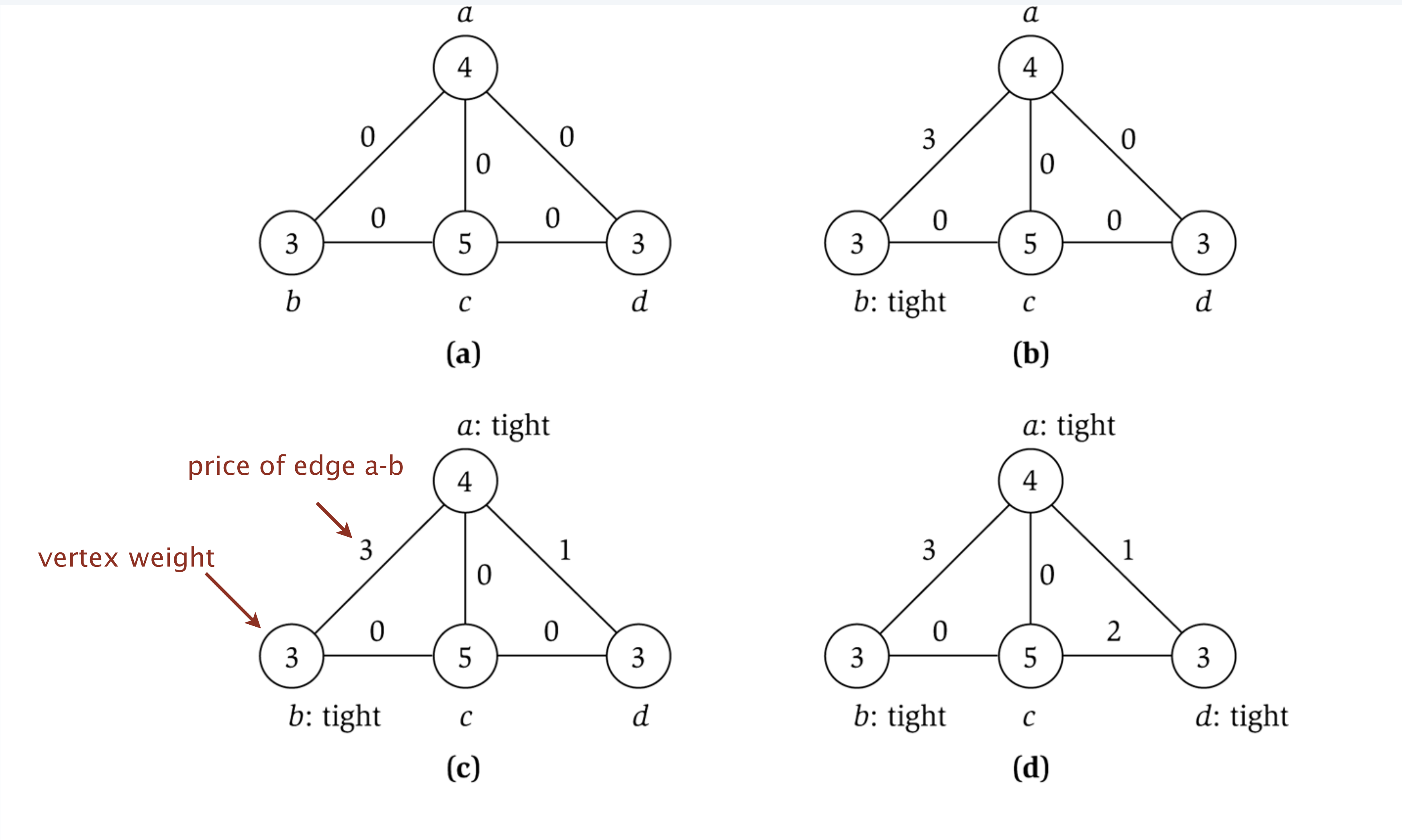
  Select such an edge $e = (i, j)$.

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

S $\leftarrow$ set of all tight nodes.

RETURN S.

Pricing method example



Pricing method: analysis

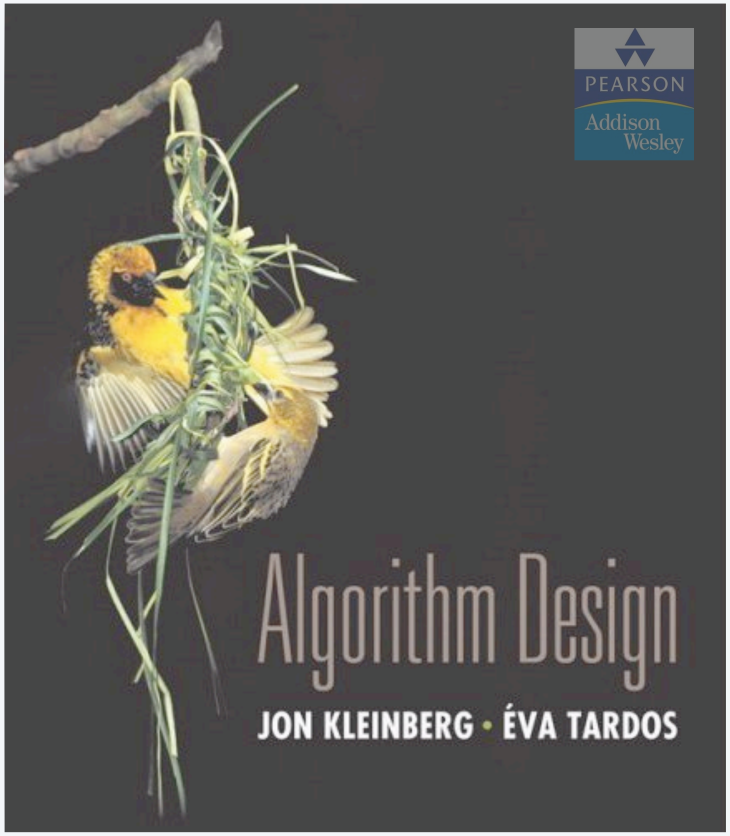
Theorem. Pricing method is a 2-approximation for WEIGHTED-VERTEX-COVER.
Pf.

- 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 nor j is tight. But then while loop would not terminate.

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

$$w(S) = \sum_{i \in S} w_i \underset{\substack{\uparrow \\ \text{all nodes in S are tight}}}{=} \sum_{i \in S} \sum_{e=(i,j)} p_e \underset{\substack{\uparrow \\ S \subseteq V, \\ \text{prices} \geq 0}}{\leq} \sum_{i \in V} \sum_{e=(i,j)} p_e \underset{\substack{\uparrow \\ \text{each edge counted twice}}}{=} 2 \sum_{e \in E} p_e \underset{\substack{\uparrow \\ \text{fairness lemma}}}{\leq} 2w(S^*). \quad \blacksquare$$



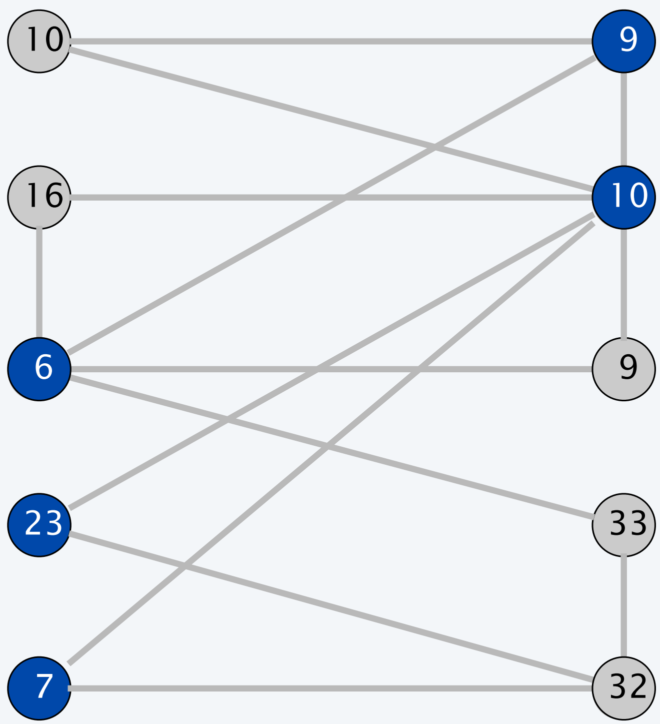
SECTION 11.6

11. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *pricing method: vertex cover*
- *LP rounding: vertex cover*
- *generalized load balancing*
- *knapsack problem*

Weighted vertex cover

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.



total weight = 6 + 23 + 7 + 9 + 10 = 55

Weighted vertex cover: IP 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 programming formulation.

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

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

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

- Objective function: minimize $\Sigma_i w_i x_i$.

- Must take either vertex i or j (or both): $x_i + x_j \geq 1$.

Weighted vertex cover: IP formulation

Weighted vertex cover. Integer programming formulation.

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

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

Integer programming

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

$$\begin{array}{lrcl} \max & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & & \text{integral} \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\ x_j & \geq & 0 & 1 \leq j \leq n \\ x_j & & \text{integral} & 1 \leq j \leq n \end{array}$$

Observation. Vertex cover formulation proves that INTEGER-PROGRAMMING is an **NP**-hard search problem.

Linear programming

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

$$
\begin{array}{llcl}
(P) & \max & c^t x & \\
& \text{s.t.} & Ax & \geq & b \\
& & x & \geq & 0
\end{array}
$$

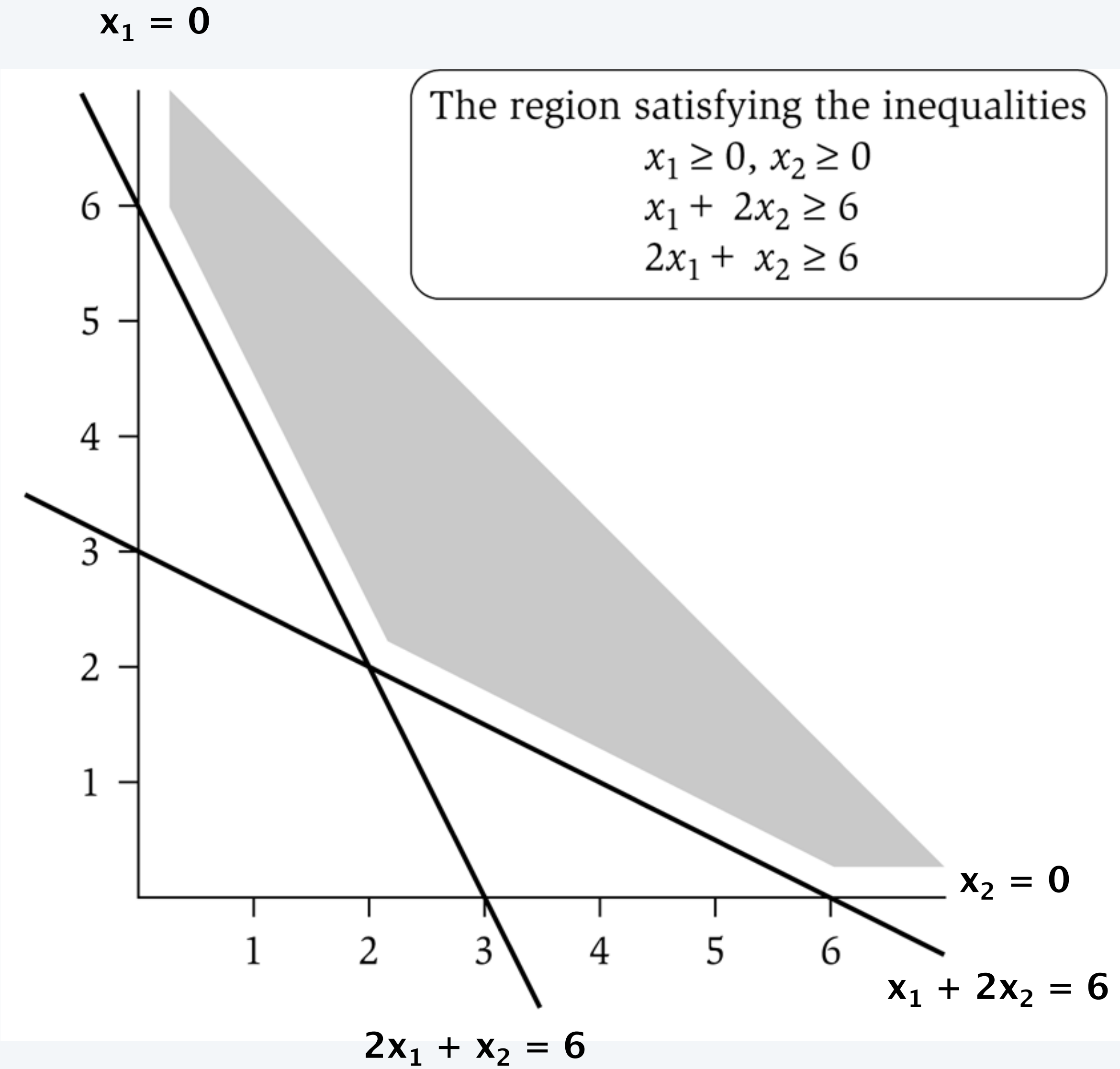
$$
\begin{array}{lllcll}
(P) & \max & \sum_{j=1}^{n} c_j x_j & & & \\
& \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\
& & x_j & \geq & 0 & 1 \leq j \leq n
\end{array}
$$

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

Simplex algorithm. [Dantzig 1947] Can solve LP in practice.
Ellipsoid algorithm. [Khachian 1979] Can solve LP in poly-time.

LP feasible region

LP geometry in 2D.



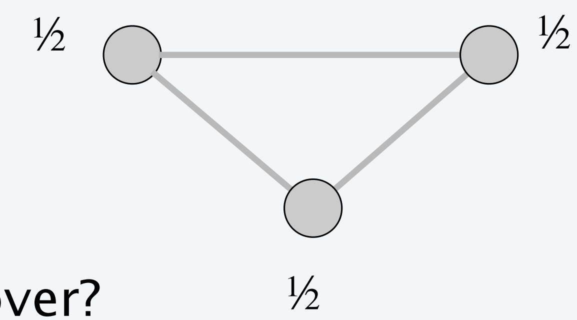
Weighted vertex cover: LP relaxation

Linear programming relaxation.

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

Observation. Optimal value of (LP) is $\leq$ optimal value of (ILP).
Pf. LP has fewer constraints.

Note. LP is not equivalent to vertex cover.



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

Weighted vertex cover: LP rounding algorithm

Lemma. If x^* is optimal solution to (LP), then $S = \{ i \in V : x_i^* \geq ½\}$ 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 $x_i^* \geq ½$ or $x_j^* \geq ½$ $\Rightarrow$ (i, j) covered.

Pf. [S has desired cost]

- Let S^* be optimal vertex cover. Then

$$\underset{\text{LP is a relaxation}}{\sum_{i \in S^*} w_i \;\; \geq \;\; \sum_{i \in S} w_i \, x_i^*} \;\; \underset{x_i^* \,\geq\, ½}{\geq} \;\; \frac{1}{2} \sum_{i \in S} w_i$$

Theorem. The rounding algorithm is a 2-approximation algorithm.
Pf. Lemma + fact that LP can be solved in poly-time.

Weighted vertex cover inapproximability

Theorem. [Dinur-Safra 2004] If **P** ≠ **NP**, then no ρ-approximation for WEIGHTED-VERTEX-COVER for any $\rho < 1.3606$ (even if all weights are 1).

On the Hardness of Approximating Minimum Vertex Cover

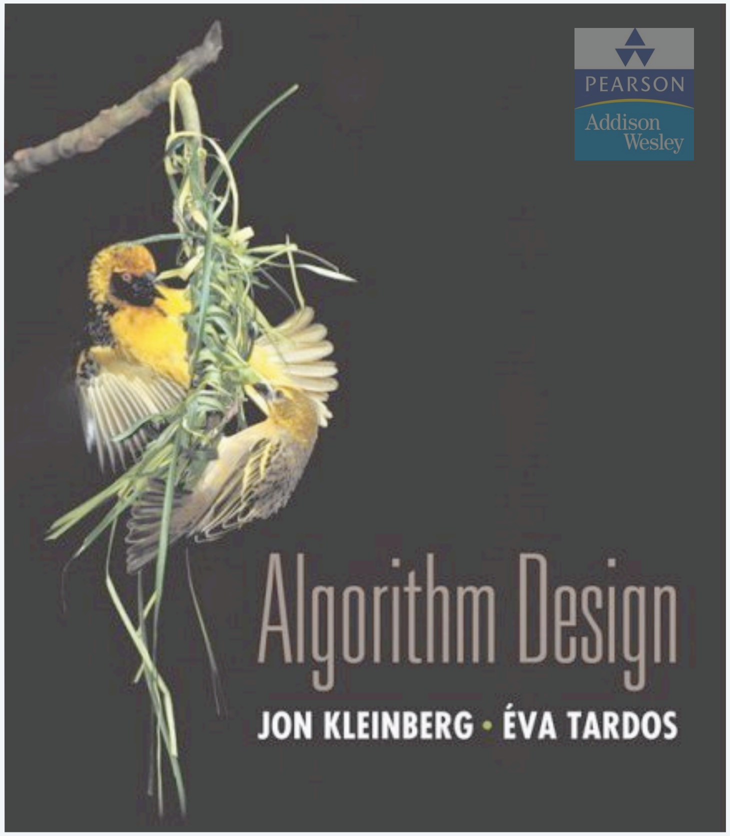
Irit Dinur* Samuel Safra†

May 26, 2004

Abstract

We prove the Minimum Vertex Cover problem to be NP-hard to approximate to within a factor of 1.3606, extending on previous PCP and hardness of approximation technique. To that end, one needs to develop a new proof framework, and borrow and extend ideas from several fields.

Open research problem. Close the gap.



SECTION 11.7

11. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *pricing method: vertex cover*
- *LP rounding: vertex cover*
- *generalized load balancing*
- *knapsack problem*

Generalized load balancing

Input. Set of m machines M; set of n jobs J.

- Job $j \in J$ must run contiguously on an authorized machine in $M_j \subseteq M$.
- Job $j \in J$ has processing time t_j.
- Each 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 $= \max_i L_i$.

Generalized load balancing. Assign each job to an authorized machine to minimize makespan.

Generalized load balancing: integer linear program and relaxation

ILP formulation. x_{ij} = time machine i spends processing job j.

$$
\begin{array}{llll}
(IP) \ \min & L & & \\
\text{s.t.} & \sum_i x_{ij} & = & t_j & \text{for all } j \in J \\
& \sum_j x_{ij} & \leq & L & \text{for all } i \in M \\
& x_{ij} & \in & \{0, t_j\} & \text{for all } j \in J \text{ and } i \in M_j \\
& x_{ij} & = & 0 & \text{for all } j \in J \text{ and } i \notin M_j
\end{array}
$$

LP relaxation.

$$
\begin{array}{llll}
(LP) \ \min & L & & \\
\text{s.t.} & \sum_i x_{ij} & = & t_j & \text{for all } j \in J \\
& \sum_j x_{ij} & \leq & L & \text{for all } i \in M \\
& x_{ij} & \geq & 0 & \text{for all } j \in J \text{ and } i \in M_j \\
& x_{ij} & = & 0 & \text{for all } j \in J \text{ and } i \notin M_j
\end{array}
$$

Generalized load balancing: lower bounds

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

Pf. Some machine must process the most time-consuming job. ▪

Lemma 2. Let L be optimal value to the LP. Then, optimal makespan $L^* \geq L$.

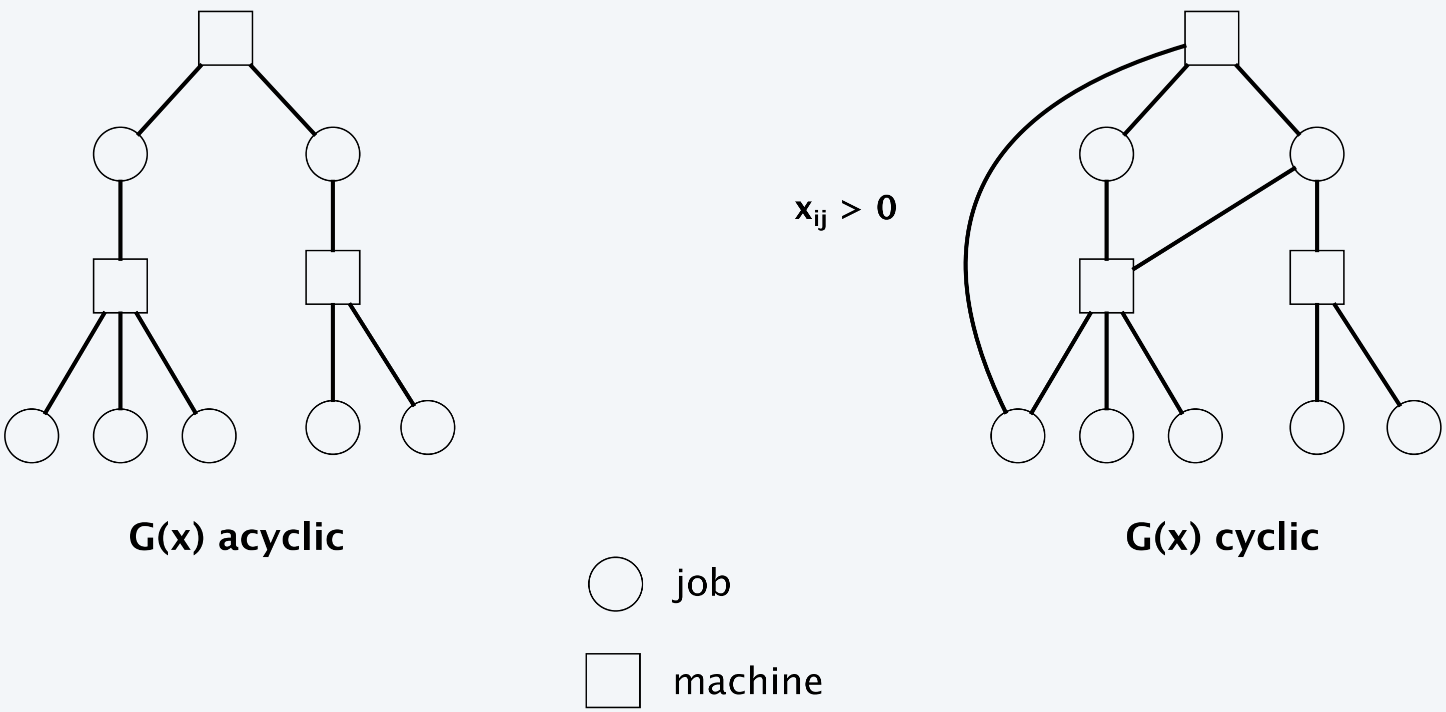
Pf. LP has fewer constraints than IP formulation. ▪

Generalized load balancing: structure of LP solution

Lemma 3. Let x be solution to LP. Let $G(x)$ be the graph with an edge between machine i and job j if $x_{ij} > 0$. Then $G(x)$ is acyclic.

can transform x into another LP solution where G(x) is acyclic if LP solver doesn't return such an x

Pf. (deferred)

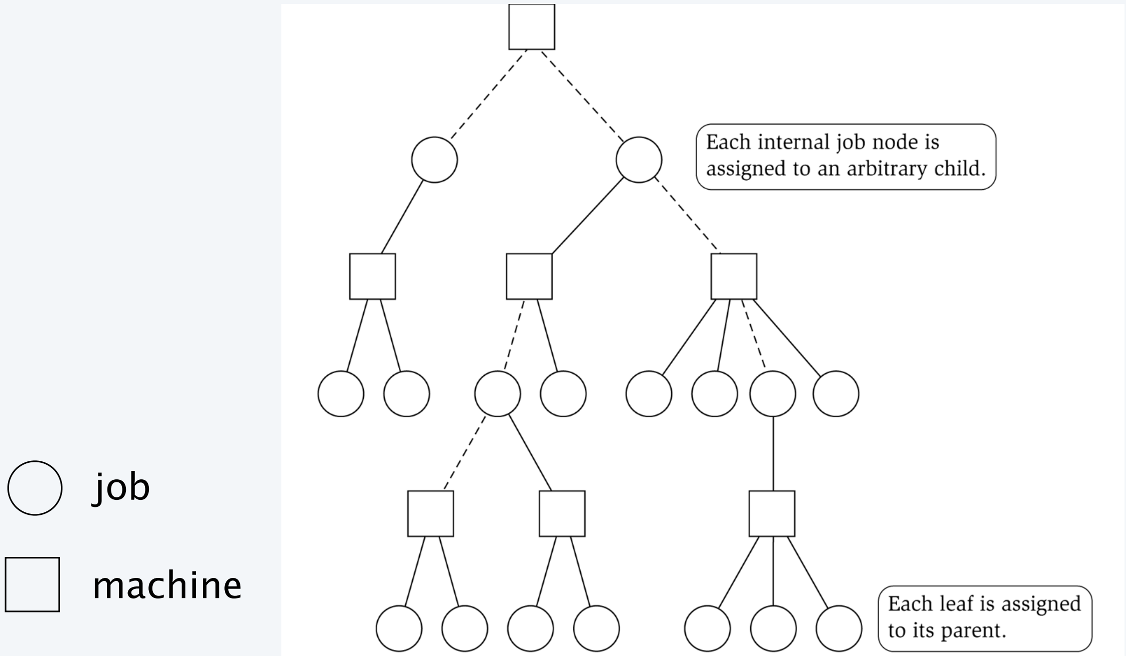


Generalized load balancing: rounding

Rounded solution. Find LP solution x where $G(x)$ is a forest. Root forest $G(x)$ at some arbitrary machine node r.

- If job j is a leaf node, assign j to its parent machine i.
- If job j is not a leaf node, assign j to any one of its children.

Lemma 4. Rounded solution only assigns jobs to authorized machines.
Pf. If job j is assigned to machine i, then $x_{ij} > 0$. LP solution can only assign positive value to authorized machines. ▪



Generalized load balancing: analysis

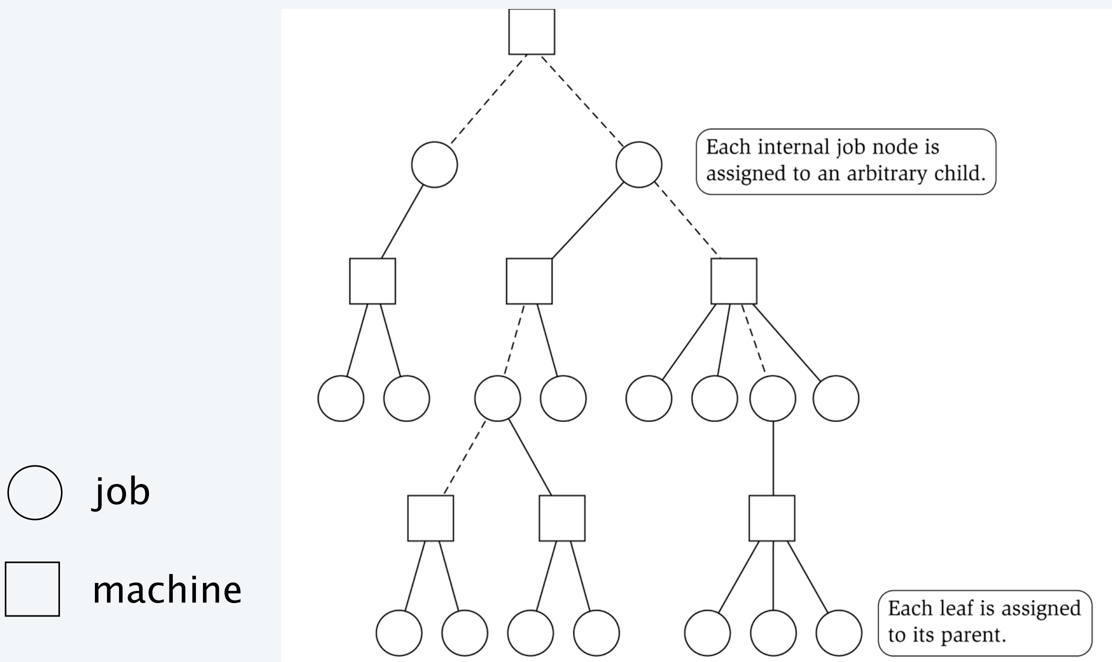
Lemma 5. If job j is a leaf node and machine $i = parent(j)$, then $x_{ij} = t_j$.

Pf.

- Since i is a leaf, $x_{ij} = 0$ for all $j \neq parent(i)$.
- LP constraint guarantees $\Sigma_i\, x_{ij} = t_j$. ▪

Lemma 6. At most one non-leaf job is assigned to a machine.

Pf. The only possible non-leaf job assigned to machine i is $parent(i)$. ▪



Generalized load balancing: analysis

Theorem. Rounded solution is a 2-approximation.

Pf.

- Let $J(i)$ be the jobs assigned to machine i.
- By LEMMA 6, the load L_i on machine i has two components:
  - leaf nodes:

$$\sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} t_j \;\overset{\text{Lemma 5}}{=}\; \sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} x_{ij} \;\le\; \sum_{j \in J} x_{ij} \;\overset{\text{LP}}{\le}\; \underset{\text{optimal value of LP}}{L} \;\overset{\text{Lemma 2 (LP is a relaxation)}}{\le}\; L^*$$

  - parent:

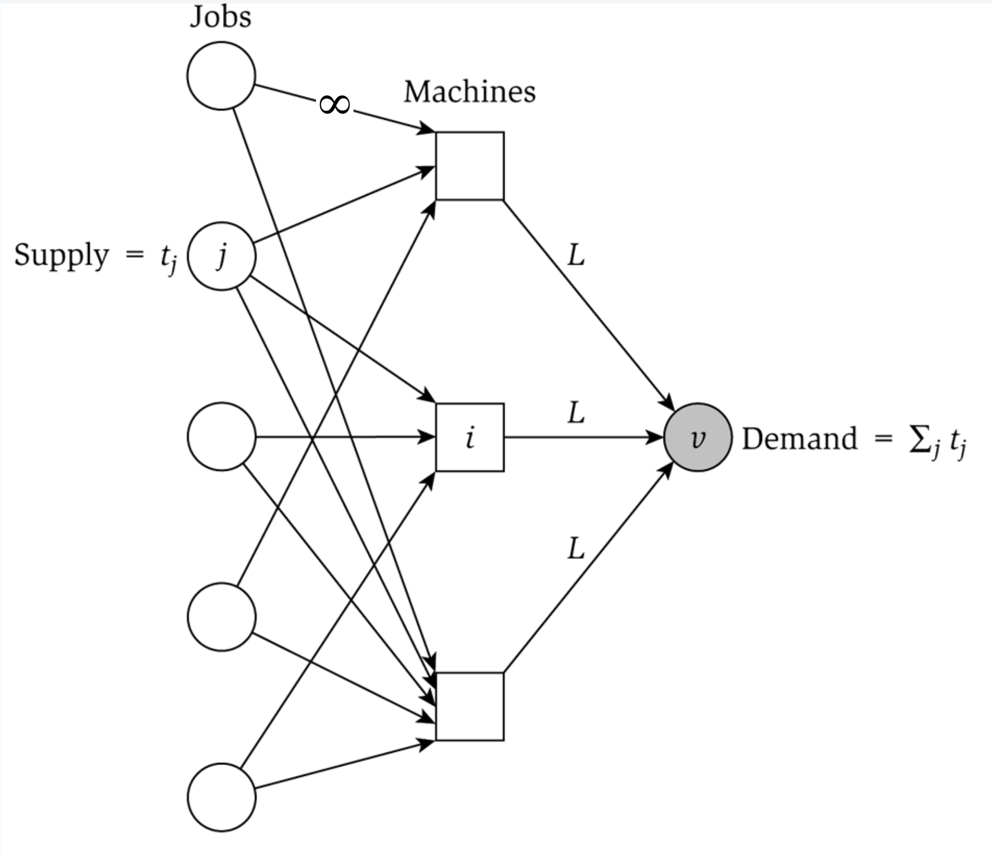
$$t_{\text{parent}(i)} \;\overset{\text{Lemma 1}}{\le}\; L^*$$

- Thus, the overall load $L_i \le 2L^*$. ▪

Generalized load balancing: flow formulation

Flow formulation of LP.

$$
\begin{array}{lcll}
\sum_i x_{ij} & = & t_j & \text{for all } j \in J \\
\sum_j x_{ij} & \leq & L & \text{for all } i \in M \\
x_{ij} & \geq & 0 & \text{for all } j \in J \text{ and } i \in M_j \\
x_{ij} & = & 0 & \text{for all } j \in J \text{ and } i \notin M_j
\end{array}
$$



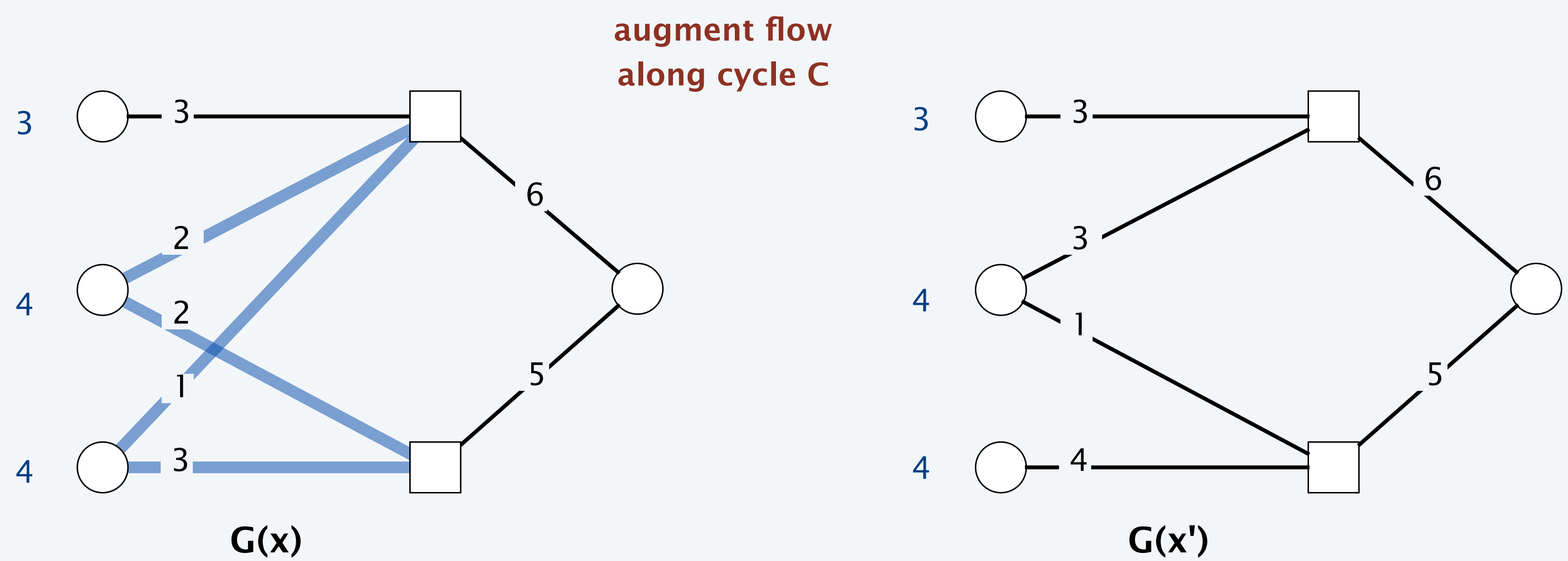
Observation. Solution to feasible flow problem with value L are in 1-to-1 correspondence with LP solutions of value L.

Generalized load balancing: structure of solution

Lemma 3. Let (x, L) be solution to LP. Let $G(x)$ be the graph with an edge from machine i to job j if $x_{ij} > 0$. We can find another solution (x', L) such that $G(x')$ is acyclic.

Pf. Let C be a cycle in $G(x)$.

- Augment flow along the cycle C. ← flow conservation maintained
- At least one edge from C is removed (and none are added).
- Repeat until $G(x')$ is acyclic. ▪



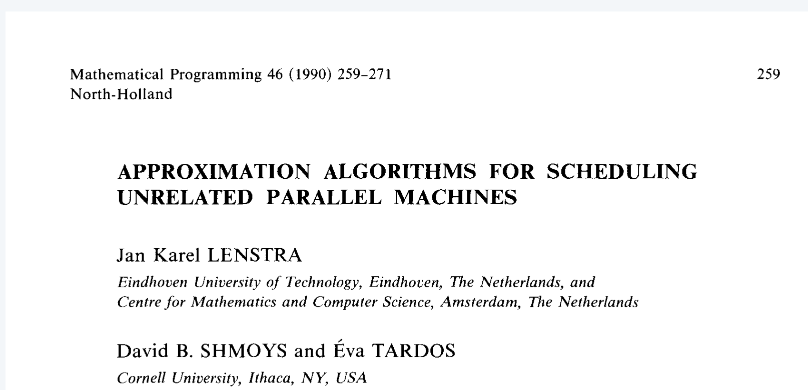
Conclusions

Running time. The bottleneck operation in our 2-approximation is solving one LP with $mn + 1$ variables.

Remark. Can solve LP using flow techniques on a graph with $m + n + 1$ nodes: given L, find feasible flow if it exists. Binary search to find L^*.

Extensions: unrelated parallel machines. [Lenstra-Shmoys-Tardos 1990]

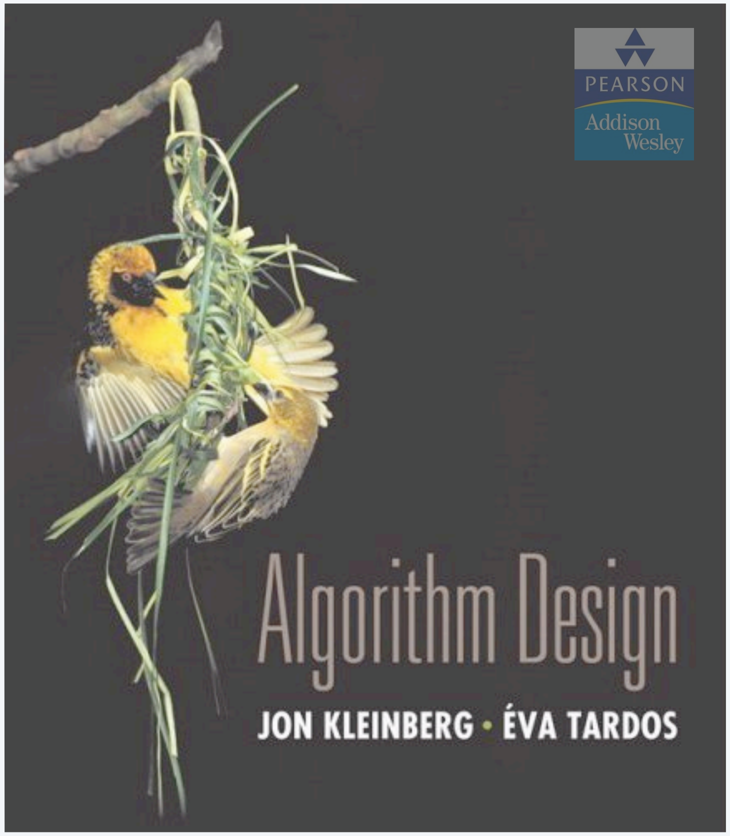
- Job j takes t_{ij} time if processed on machine i.
- 2-approximation algorithm via LP rounding.
- If **P ≠ NP**, then no no ρ-approximation exists for any $\rho < 3/2$.


Mathematical Programming 46 (1990) 259–271
North-Holland
259

APPROXIMATION ALGORITHMS FOR SCHEDULING UNRELATED PARALLEL MACHINES

Jan Karel LENSTRA
Eindhoven University of Technology, Eindhoven, The Netherlands, and Centre for Mathematics and Computer Science, Amsterdam, The Netherlands

David B. SHMOYS and Éva TARDOS
Cornell University, Ithaca, NY, USA



SECTION 11.8

11. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *pricing method: vertex cover*
- *LP rounding: vertex cover*
- *generalized load balancing*
- *knapsack problem*

Polynomial-time approximation scheme

PTAS. $(1 + \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.

- Load balancing. [Hochbaum-Shmoys 1987]
- Euclidean TSP. [Arora, Mitchell 1996]

Consequence. PTAS produces arbitrarily high quality solution, but trades off accuracy for time.

This section. PTAS for knapsack problem via rounding and scaling.

Knapsack problem

Knapsack problem.

- Given n objects and a knapsack.
- Item i has value $v_i > 0$ and weighs $w_i > 0$. ← we assume $w_i \le W$ for each i
- Knapsack has weight limit W.
- Goal: fill knapsack so as to maximize total value.

Ex: $\{ 3, 4 \}$ has value 40.

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

original instance (W = 11)

Knapsack is NP-complete

KNAPSACK. Given a set X, weights $w_i \geq 0$, values $v_i \geq 0$, a weight limit W, and a target value V, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} w_i \leq W$$
$$\sum_{i \in S} v_i \geq V$$

SUBSET-SUM. Given a set X, values $u_i \geq 0$, and an integer U, is there a subset $S \subseteq X$ whose elements sum to exactly U?

Theorem. SUBSET-SUM $\leq_P$ KNAPSACK.

Pf. Given instance $(u_1, \ldots, u_n, U)$ of SUBSET-SUM, create KNAPSACK instance:

$$v_i = w_i = u_i \qquad \sum_{i \in S} u_i \leq U$$
$$V = W = U \qquad \sum_{i \in S} u_i \geq U$$

Knapsack problem: dynamic programming I

Def. $OPT(i, w)$ = max value subset of items $1, \ldots, i$ with weight limit w.

Case 1. OPT does not select item i.

- OPT selects best of $1, \ldots, i-1$ using up to weight limit w.

Case 2. OPT selects item i.

- New weight limit $= w - w_i$.
- OPT selects best of $1, \ldots, i-1$ using up to weight limit $w - w_i$.

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{\, OPT(i-1, w),\ \ v_i + OPT(i-1, w-w_i) \,\} & \text{otherwise} \end{cases}$$

Theorem. Computes the optimal value in $O(n\,W)$ time.

- Not polynomial in input size.
- Polynomial in input size if weights are small integers.

Knapsack problem: dynamic programming II

Def. $OPT(i, v)$ = min weight of a knapsack for which we can obtain a solution of value ≥ v using a subset of items $1,\dots, i$.

Note. Optimal value is the largest value v such that $OPT(i, v) \leq W$.

Case 1. OPT does not select item i.

- OPT selects best of $1, \dots, i-1$ that achieves value v.

Case 2. OPT selects item i.

- Consumes weight w_i, need to achieve value $v - v_i$.
- OPT selects best of $1, \dots, i-1$ that achieves value $v - v_i$.

$$OPT(i,v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } i = 0 \text{ and } \ v > 0 \\ \min \{OPT(i-1, v), \ w_i + OPT(i-1, v - v_i)\} & \text{otherwise} \end{cases}$$

Knapsack problem: dynamic programming II

Theorem. Dynamic programming algorithm II computes the optimal value in $O(n^2\, v_{max})$ time, where v_{max} is the maximum of any value.

Pf.

- The optimal value $V^* \leq n\, v_{max}$.
- There is one subproblem for each item and for each value $v \leq V^*$.
- It takes $O(1)$ time per subproblem. ▪

Remark 1. Not polynomial in input size!

Remark 2. Polynomial time if values are small integers.

Knapsack problem: polynomial-time approximation scheme

Intuition for approximation algorithm.

- Round all values up to lie in smaller range.
- Run dynamic programming algorithm II on rounded/scaled instance.
- Return optimal items in rounded instance.

item	value	weight
1	934221	1
2	5956342	2
3	17810013	5
4	21217800	6
5	27343199	7

original instance (W = 11)

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

rounded instance (W = 11)

Knapsack problem: polynomial-time approximation scheme

Round up all values:

- $0 < \varepsilon \leq 1$ = precision parameter.
- v_{max} = largest value in original instance.
- θ = scaling factor = $\varepsilon\, v_{max} / 2n$.

$$\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta \,, \qquad \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$$

Observation. Optimal solutions to problem with $\bar{v}$ are equivalent to optimal solutions to problem with $\hat{v}$.

Intuition. $\bar{v}$ close to v so optimal solution using $\bar{v}$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm II is fast.

Knapsack problem: polynomial-time approximation scheme

Theorem. If S is solution found by rounding algorithm and S^* is any other feasible solution, then $$(1+\epsilon)\sum_{i \in S} v_i \;\geq\; \sum_{i \in S^*} v_i$$

Pf. Let S^* be any feasible solution satisfying weight constraint.

$$\sum_{i \in S^*} v_i \;\leq\; \sum_{i \in S^*} \bar{v}_i$$ always round up

$$\leq\; \sum_{i \in S} \bar{v}_i$$ solve rounded instance optimally

$$\leq\; \sum_{i \in S} (v_i + \theta)$$ never round up by more than θ

$$\leq\; \sum_{i \in S} v_i \;+\; n\theta$$ $|S| \leq n$

$$=\; \sum_{i \in S} v_i \;+\; \tfrac{1}{2}\,\epsilon\, v_{max}$$ $\theta = \varepsilon\, v_{max} / 2n$

$$=\; (1+\epsilon)\sum_{i \in S} v_i$$ $v_{max} \leq 2\,\Sigma_{i \in S}\, v_i$

choosing $S^* = \{ \, max \, \}$

$$v_{max} \;\leq\; \sum_{i \in S} v_i \;+\; \tfrac{1}{2}\,\epsilon\, v_{max}$$
$$\leq\; \sum_{i \in S} v_i \;+\; \tfrac{1}{2}\, v_{max}$$

thus

$$v_{max} \;\leq\; 2\sum_{i \in S} v_i$$

Knapsack problem: polynomial-time approximation scheme

Theorem. For any $\varepsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1 + \varepsilon)$ factor of the optimum in $O(n^3 / \varepsilon)$ time.

Pf.

- We have already proved the accuracy bound.
- Dynamic program II running time is $O(n^2 \, \hat{v}_{\max})$, where

$$\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$