CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2019

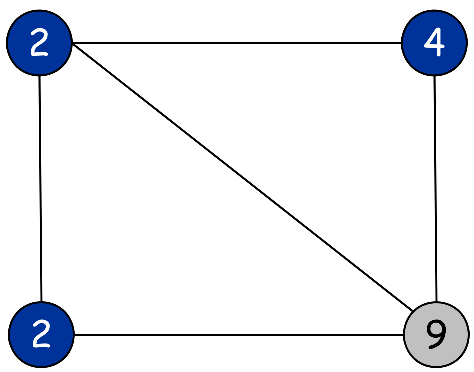
Homework 6 Released Tonight: Due April 23 at 11:59 PM on Gradescope

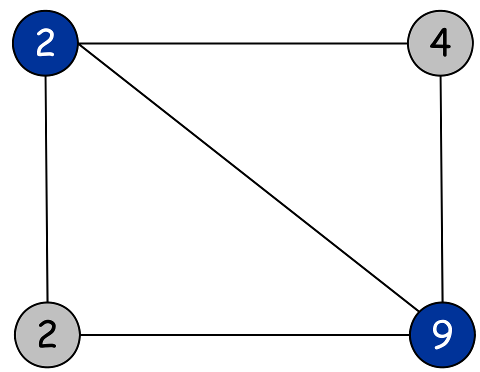
11.6 LP Rounding: Vertex Cover

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.



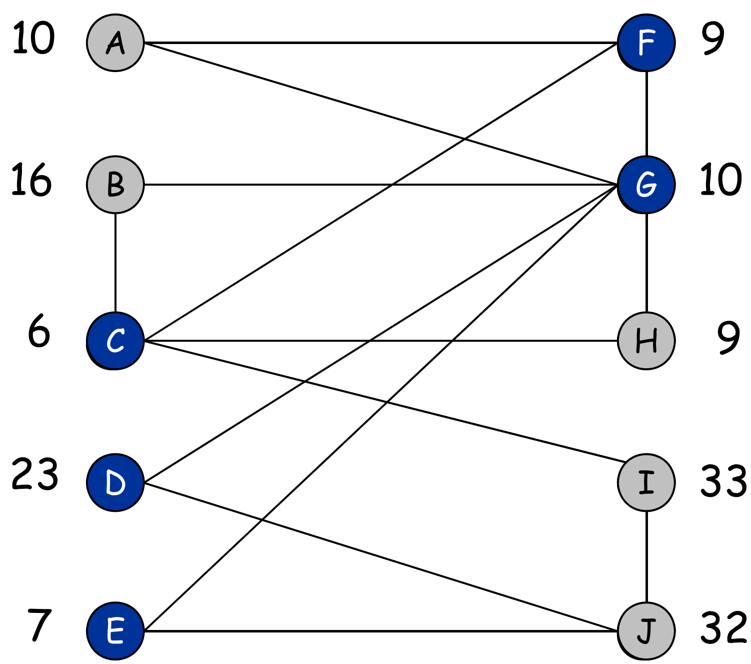


weight = 2 + 2 + 4

weight = 11

Weighted Vertex Cover

Weighted vertex cover. Given an undirected graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S such that every edge is incident to at least one vertex in S.



total weight = 55

Weighted Vertex Cover: IP Formulation

Weighted vertex cover. Given an undirected graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a minimum weight subset of nodes S 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 i or j: $x_i + x_j \geq 1$.

Weighted Vertex Cover: IP Formulation

Weighted vertex cover. Integer programming formulation.

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

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

Integer Programming

INTEGER-PROGRAMMING. Given integers a_{ij} and b_i, find integers x_j that satisfy:

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

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

Observation. Vertex cover formulation proves that integer programming is NP-hard search problem.

even if all coefficients are 0/1 and
at most two variables per inequality

Linear Programming

Linear programming. Max/min linear objective function subject to linear inequalities.

- Input: integers c_j, b_i, a_{ij} .
- Output: real numbers x_j.

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

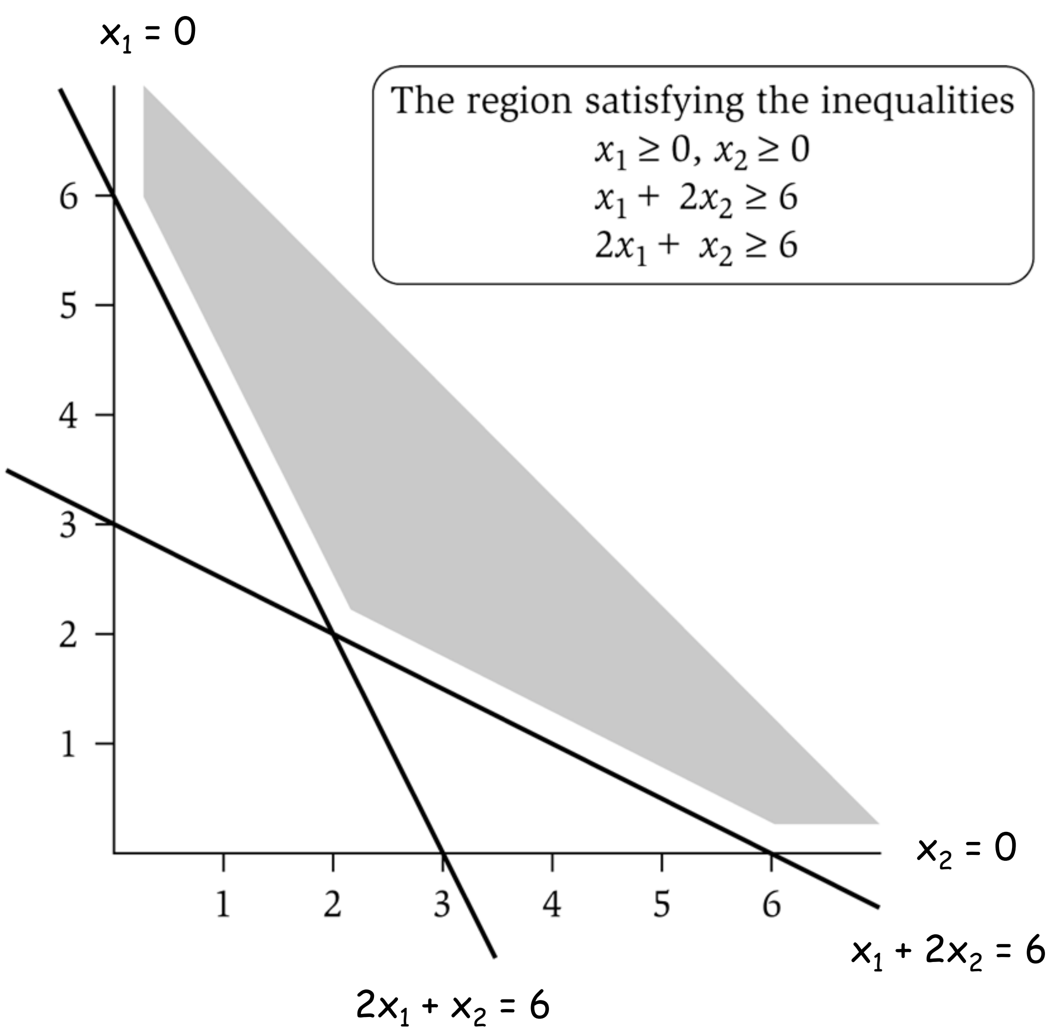
$$
\begin{array}{llrcll}
(P) & \max & \sum_{j=1}^{n} c_j x_j & & & \\
 & \text{s. t.} & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & 1 \leq i \leq m \\
 & & x_j & \geq & 0 & 1 \leq j \leq n
\end{array}
$$

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.



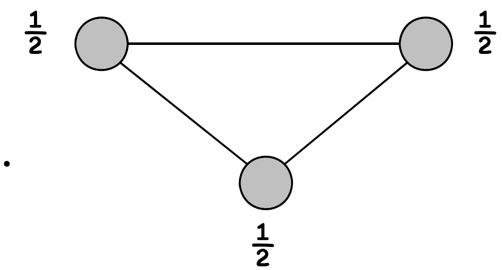
Weighted Vertex Cover: LP Relaxation

Weighted vertex cover. Linear programming formulation.

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

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

Theorem. If x^* is optimal solution to (LP), then $S = \{i \in V : x^*_i \geq \frac{1}{2}\}$ 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 $x^*_i \geq \frac{1}{2}$ or $x^*_j \geq \frac{1}{2}$ $\Rightarrow$ (i, j) covered.

Pf. [S has desired cost]

- Let S^* be optimal vertex cover. Then

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

Weighted Vertex Cover

Theorem. 2-approximation algorithm for weighted vertex cover.

Theorem. [Dinur-Safra 2001] If P $\neq$ NP, then no ρ-approximation for $\rho < 1.3607$, even with unit weights.

$10\sqrt{5} - 21$

Open research problem. Close the gap.

Theorem. [Khot-Regev 2003] No polynomial time ρ-approximation for any constant $\rho < 2$ under a stronger conjecture called the ``Unique Games Conjecture."

12.1 Landscape of an Optimization Problem

Gradient Descent: Vertex Cover

VERTEX-COVER. Given a graph $G = (V, E)$, find a subset of nodes S of minimal cardinality such that for each u-v in E, either u or v (or both) are in S.

Neighbor relation. $S \sim S'$ if S' can be obtained from S by adding or deleting a single node. Each vertex cover S has at most n neighbors.

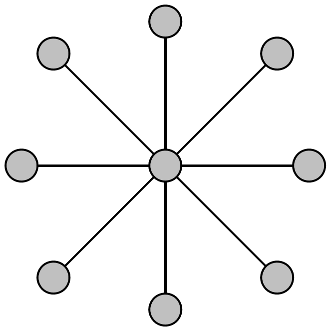
Gradient descent. Start with $S = V$. If there is a neighbor S' that is a vertex cover and has lower cardinality, replace S with S'.

Alternative. Run 2-appx alg for Vertex-Cover $S=S_{apx}$ to obtain run Gradient Descent with to improve the solution.

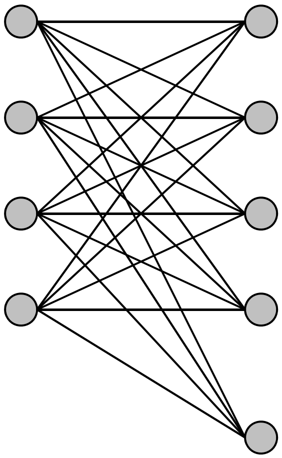
Remark. Algorithm terminates after at most n steps since each update decreases the size of the cover by one.

Gradient Descent: Vertex Cover

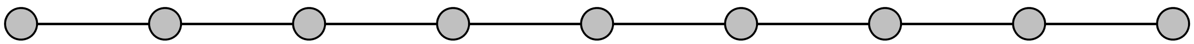
Local optimum. No neighbor is strictly better.



optimum = center node only
local optimum = all other nodes



optimum = all nodes on left side
local optimum = all nodes on right side



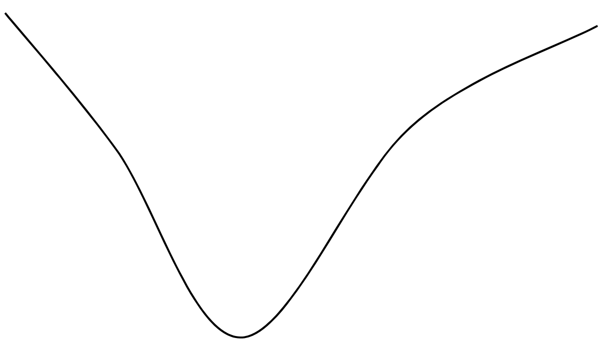
optimum = even nodes
local optimum = omit every third node

Local Search

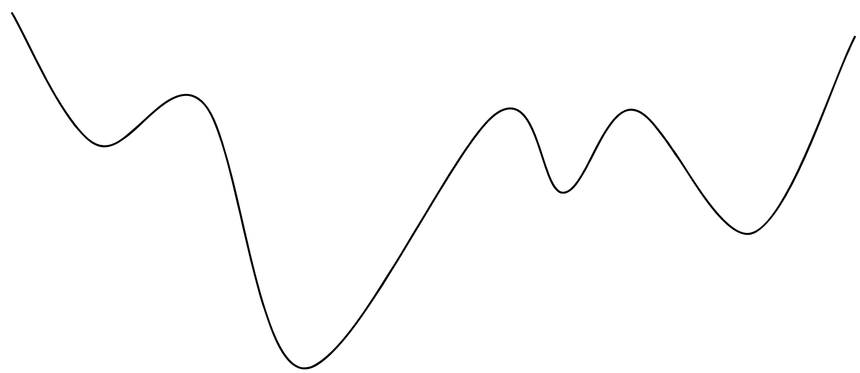
Local search. Algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a "nearby" one.

Neighbor relation. Let $S \sim S'$ be a neighbor relation for the problem.

Gradient descent. Let S denote current solution. If there is a neighbor S' of S with strictly lower cost, replace S with the neighbor whose cost is as small as possible. Otherwise, terminate the algorithm.



A funnel



A jagged funnel

11.8 Knapsack Problem

Polynomial Time Approximation Scheme

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

- Load balancing. [Hochbaum-Shmoys 1987]
- Euclidean TSP. [Arora 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'll assume $w_i \leq W$
- Knapsack can carry weight up to W.
- Goal: fill knapsack so as to maximize total value.

Ex: { 3, 4 } has value 40.

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack is NP-Complete

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

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

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

Claim. SUBSET-SUM $\leq_P$ KNAPSACK.

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

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

Knapsack Problem: Dynamic Programming 1

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

- Case 1: OPT does not select item i.
  - OPT selects best of 1, …, 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, …, i-1 using up to weight limit w – w_i

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

Running time. O(n W).

- W = weight limit.
- Not polynomial in input size!

Knapsack Problem: Dynamic Programming II

Def. OPT(i, v) = min weight subset of items 1, …, i that yields value exactly v.

- Case 1: OPT does not select item i.
  - OPT selects best of 1, …, i-1 that achieves exactly value v
- Case 2: OPT selects item i.
  - consumes weight w_i, new value needed = v – v_i
  - OPT selects best of 1, …, i-1 that achieves exactly value v

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

$V^* \le n\, v_{max}$

Running time. $O(n V^*) = O(n^2 v_{max})$.

- V^* = optimal value = maximum v such that OPT(n, v) $\le$ W.
- Not polynomial in input size!

Knapsack: FPTAS

Intuition for approximation algorithm.

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

Item	Value	Weight
1	934,221	1
2	5,956,342	2
3	17,810,013	5
4	21,217,800	6
5	27,343,199	7

W = 11

original instance

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

rounded instance

Knapsack: FPTAS

Knapsack FPTAS. Round up all values: $\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta \quad \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$

- v_{max} = largest value in original instance
- ε = precision parameter
- θ = scaling factor = $\varepsilon\, v_{max} / n$

Observation. Optimal solution to problems with $\bar{v}$ or $\hat{v}$ are equivalent.

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

Running time. $O(n^3/\varepsilon)$

- Dynamic program II running time is $O(n^2 \hat{v}_{max})$, where

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

Knapsack: FPTAS

Knapsack FPTAS. Round up all values: $\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta$

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

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

$$
\begin{aligned}
\sum_{i \in S^*} v_i &\leq \sum_{i \in S^*} \bar{v}_i && \text{always round up} \\
&\leq \sum_{i \in S} \bar{v}_i && \text{solve rounded instance optimally} \\
&\leq \sum_{i \in S} (v_i + \theta) && \text{never round up by more than } \theta \\
&\leq \sum_{i \in S} v_i + n\theta && |S| \leq n \\
&\leq (1+\varepsilon) \sum_{i \in S} v_i && n\theta = \varepsilon v_{max},\ \underset{\text{DP alg can take } v_{max}}{v_{max} \leq \textstyle\sum_{i \in S} v_i}
\end{aligned}
$$

* 11.7 Load Balancing Reloaded

Generalized Load Balancing

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

- Job j must run contiguously on an authorized machine in $M_j \subseteq M$.
- Job 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

Def. 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.

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

LP relaxation.

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

Generalized Load Balancing: Lower Bounds

Lemma 1. Let L be the optimal value to the LP. Then, the optimal makespan $L^* \geq L$.

Pf. LP has fewer constraints than IP formulation.

Lemma 2. The optimal makespan $L^* \geq \max_j t_j$.

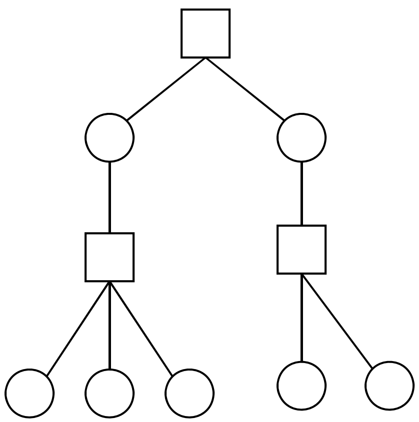
Pf. Some machine must process the most time-consuming job. ▪

Generalized Load Balancing: Structure of LP Solution

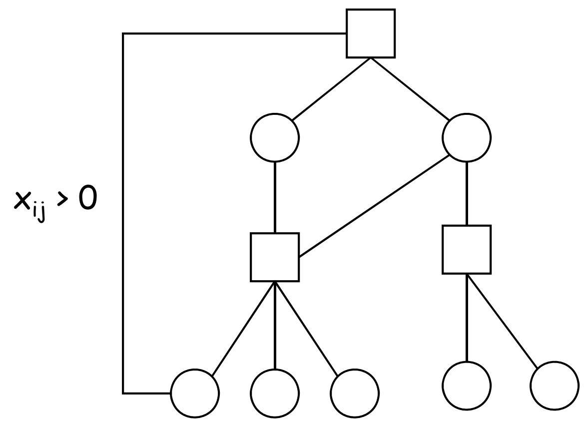
Lemma 3. Let x be solution to LP. Let G(x) be the graph with an edge from machine i to job j if $x_{ij} > 0$. Then G(x) is acyclic.

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

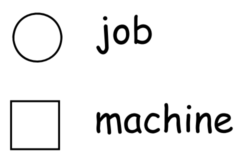
Pf. (deferred)



G(x) acyclic



G(x) cyclic



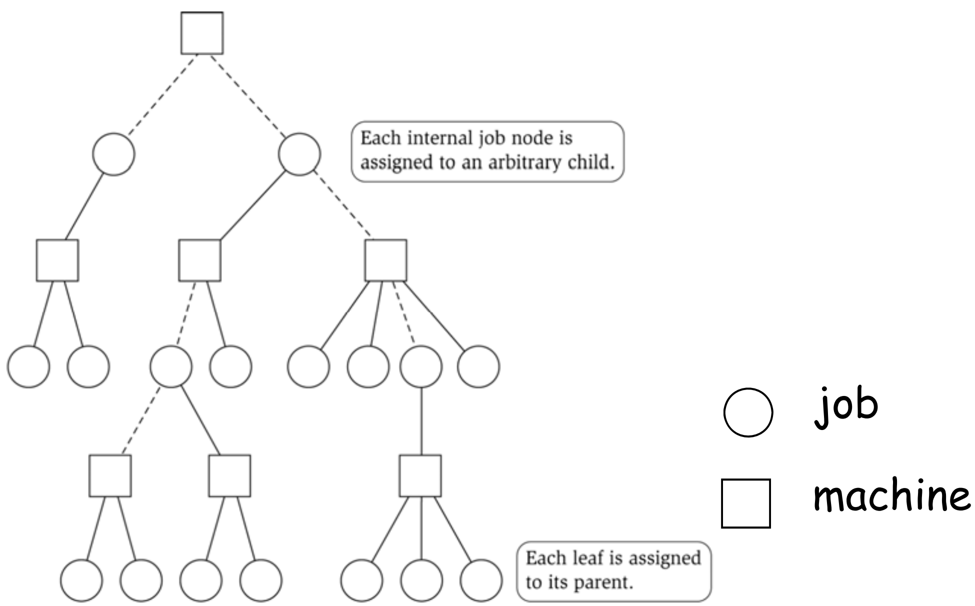
Generalized Load Balancing: Rounding

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 one of its children.

Lemma 4. Rounded solution only assigns jobs to authorized machines.

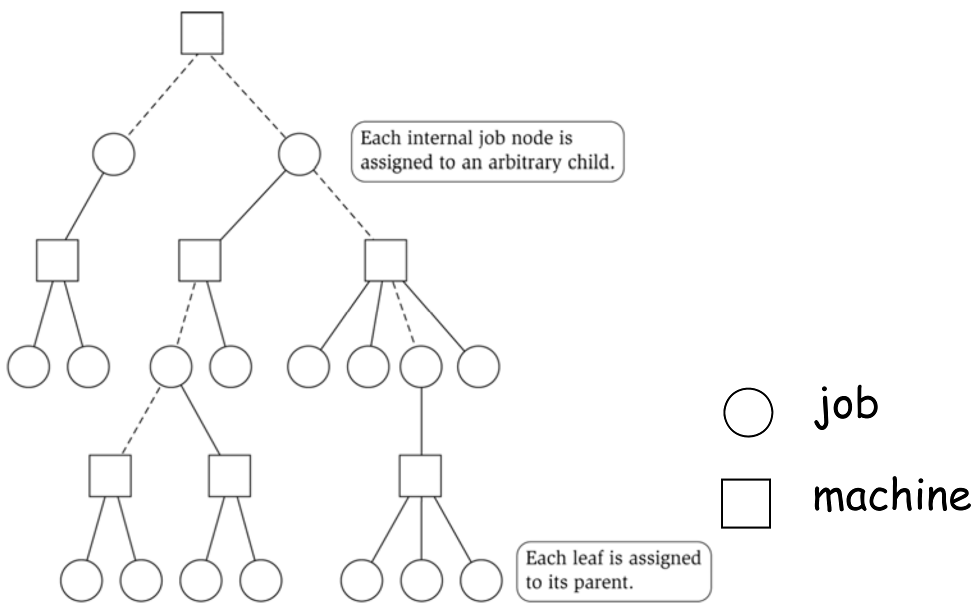
Pf. If job j is assigned to machine i, then $x_{ij} > 0$. LP solution can only assign positive value to authorized machines. ▪



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

$$\sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} t_j \overset{\text{Lemma 5}}{=} \sum_{\substack{j \in J(i) \\ j \text{ is a leaf}}} x_{ij} \le \sum_{j \in J} x_{ij} \overset{\text{LP}}{\le} \underset{\text{optimal value of LP}}{L} \overset{\text{Lemma 1 (LP is a relaxation)}}{\le} L^*$$

  - parent(i)

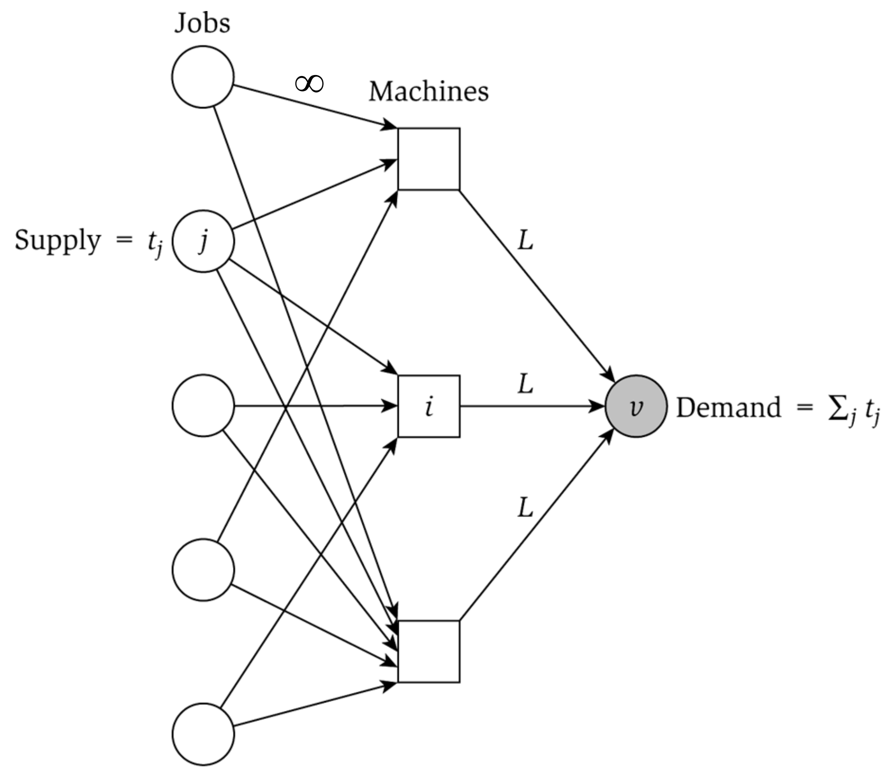
$$t_{\text{parent}(i)} \overset{\text{Lemma 2}}{\le} L^*$$

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

Generalized Load Balancing: Flow Formulation

Flow formulation of LP.

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



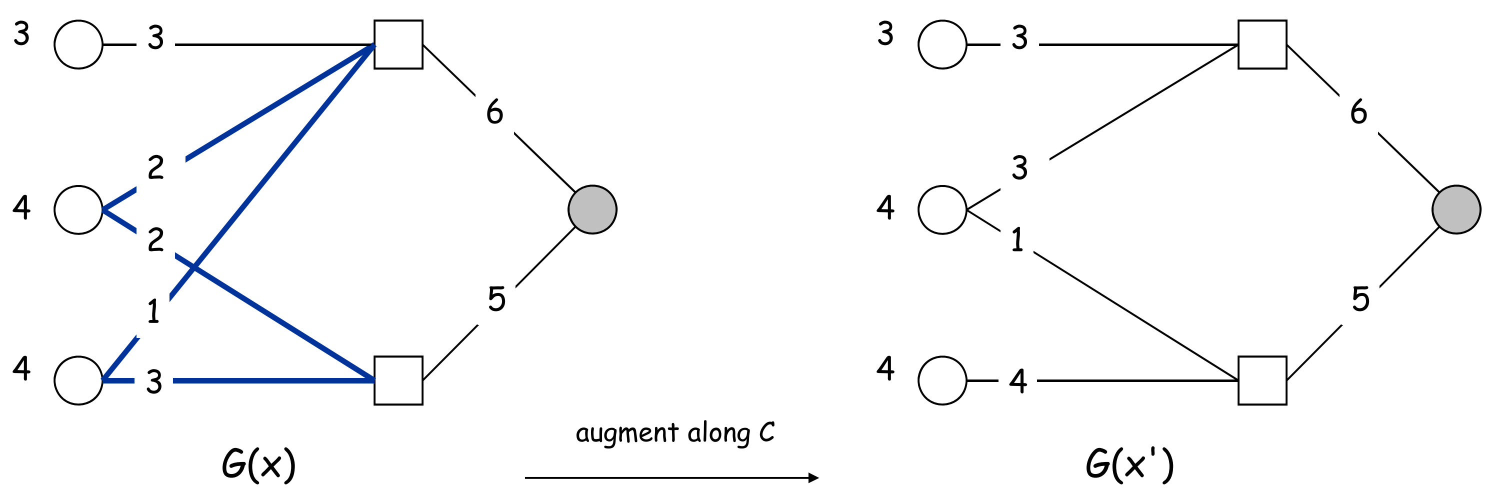
Observation. Solution to feasible flow problem with value L are in one-to-one correspondence with LP solutions of value L.

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]

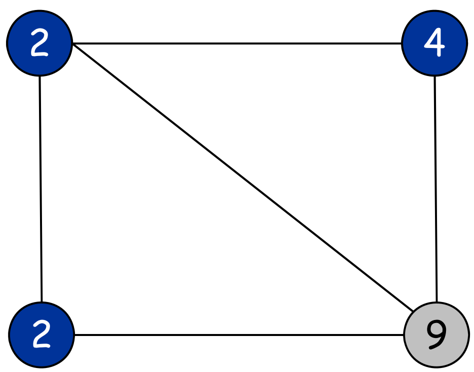
- Job j takes t_{ij} time if processed on machine i.
- 2-approximation algorithm via LP rounding.
- No 3/2-approximation algorithm unless P = NP.

11.4 The Pricing Method: Vertex Cover

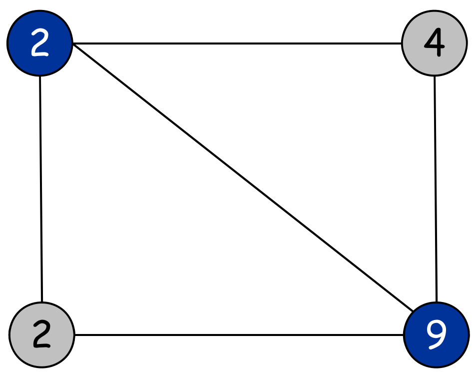
Weighted Vertex Cover

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



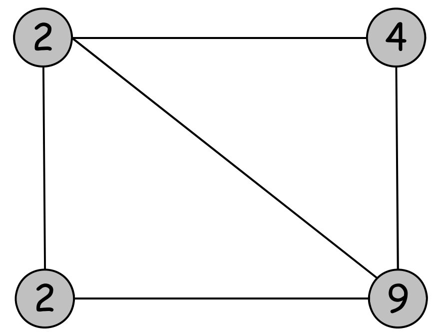
weight = 11

Pricing Method

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

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

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



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

Pf. ▪

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

each edge e covered by at least one node in S

sum fairness inequalities for each node in S

Pricing Method

Pricing method. Set prices and find vertex cover simultaneously.

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

```
Weighted-Vertex-Cover-Approx(G, w) {
   foreach e in E
      pe = 0

   while (∃ edge i-j such that neither i nor j are tight)
      select such an edge e
      increase pe as much as possible until i or j tight
   }

   S ← set of all tight nodes
   return S
}
```

Pricing Method

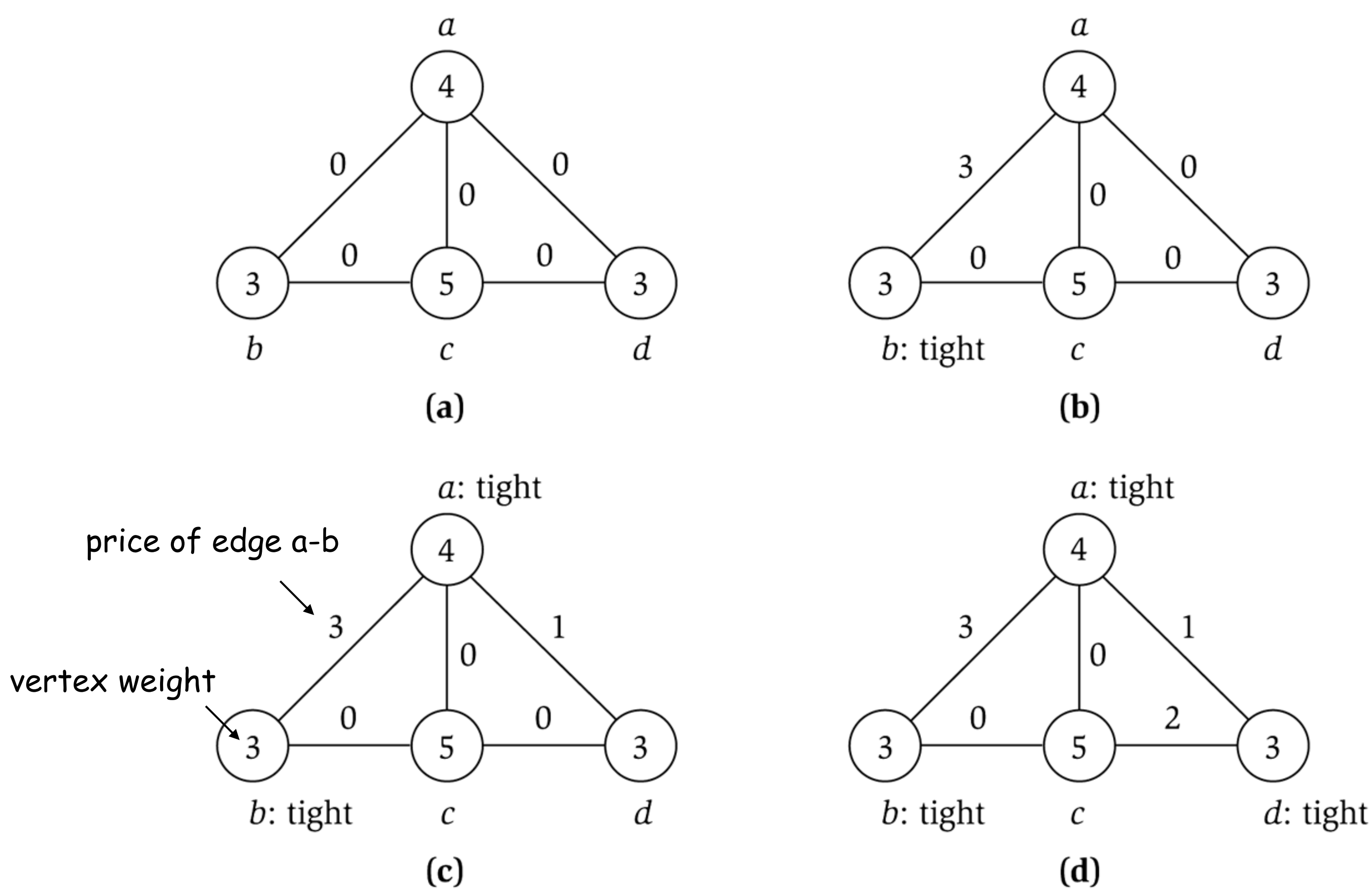


Figure 11.8

Pricing Method: Analysis

Theorem. Pricing method is a 2-approximation.
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) $\leq$ 2w(S*).

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

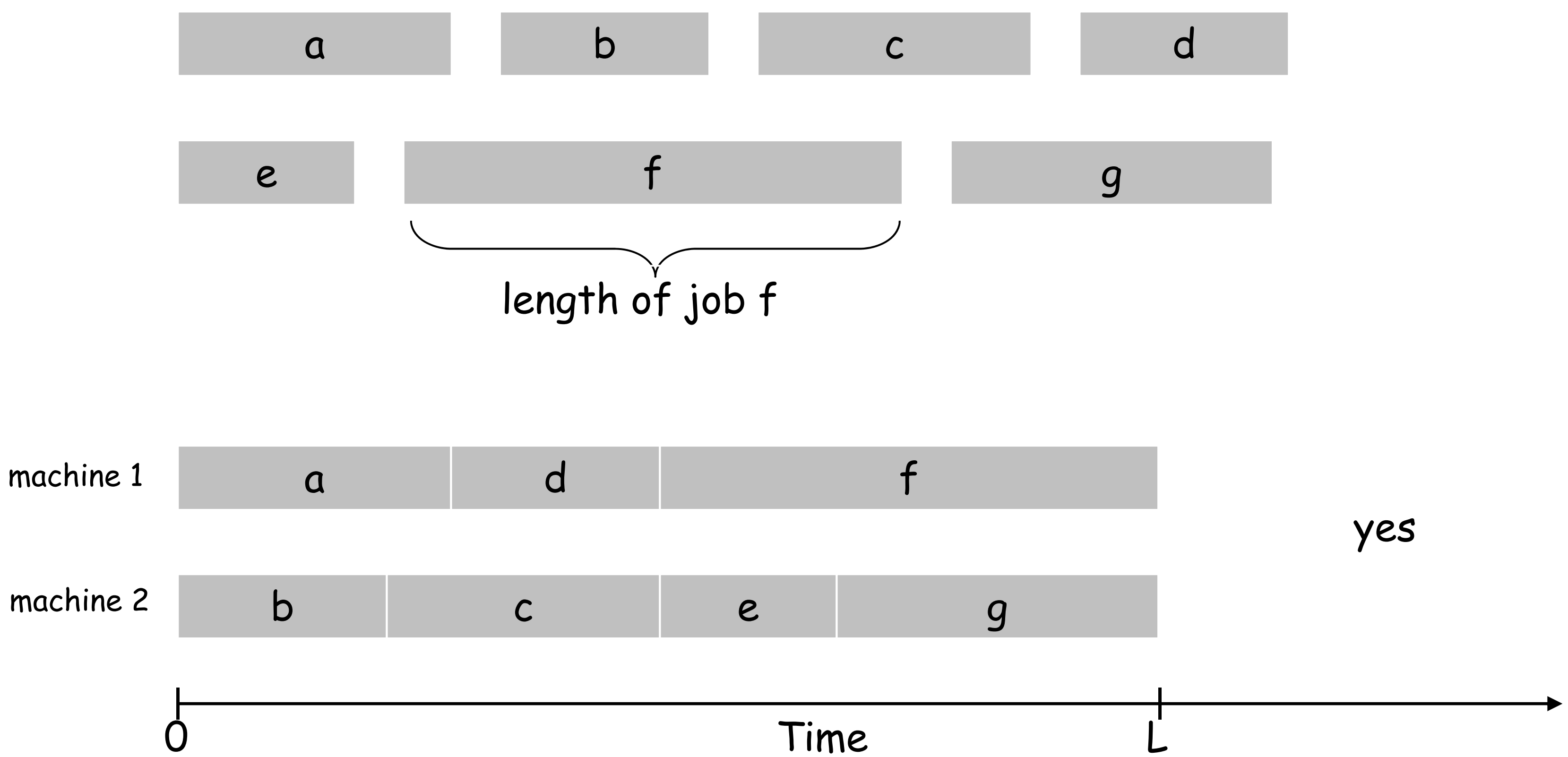
Extra Slides

Load Balancing on 2 Machines

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

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

NP-complete by Exercise 8.26



Center Selection: Hardness of Approximation

Theorem. Unless P = NP, there is no ρ-approximation algorithm for metric k-center problem for any $\rho < 2$.

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

- Let $G = (V, E)$, k be an instance of DOMINATING-SET. ← see Exercise 8.29
- Construct instance G' of k-center with sites V and distances
  - $d(u, v) = 2$ if $(u, v) \in E$
  - $d(u, v) = 1$ if $(u, v) \notin E$
- Note that G' satisfies the triangle inequality.
- Claim: 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 on G' must find a solution C^* with $r(C^*) = 1$ since it cannot use any edge of distance 2.