Algorithm II

11. Approximation Algorithms

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

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.

- 1. Solve *arbitrary instances* of the problem.
- 2. Solve problem to *optimality*.
- 3. Solve problem in *polynomial time*.

ρ**-approximation algorithm**.

- Runs in polynomial time.
- Applies to arbitrary instances of the problem.
- Guaranteed to find a solution within ratio ρ of true optimum.

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

Load balancing

Load balancing

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

- A job 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.

6	a	a	d	f	f	f	
7	b	c	c	e	g	g	g

LOAD-BALANCE on 2 machines is NP-hard

Claim. Load balancing is hard even if $m = 2$ machines.
Pf. PARTITION $\leq_P$ LOAD-BALANCE.

Number Partitioning Problem. [Exercise 8.26] You are given positive integers $x_1, .., x_n$; you want to decide whether the numbers can be partitioned into two sets S_1 and S_2 with the same sum: $\sum_{x_i \in S_1} x_i = \sum_{x_j \in S_2} x_j$.

- Hint: SUBSET-SUM $\leq_P$ PARTITION

6	a	a	d	f	f	f
6	b	c	c	e	g	g

LOAD-BALANCE: 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 $(m, n, t_1, t_2, .., t_n)$

1. FOR $i = 1..m$:
   1. $L[i] = 0$;
   2. $S[i] = \emptyset$;
2. FOR $j = 1..n$:
   1. $i = \arg\min_k L[k]$;
   2. $S[i] = S[i] \cup \{j\}$;
   3. $L[i] = L[i] + t_j$;
3. RETURN $S[1], S[2], .., S[m]$;

Implementation. $O(n \log m)$ using a priority queue for loads $L[k]$.

Demo: list scheduling

Greedy for LOAD-BALANCE: 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^*.

Greedy for LOAD-BALANCE: 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. For all k: the optimal makespan $L^* \geq t_k$.

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

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

Pf.

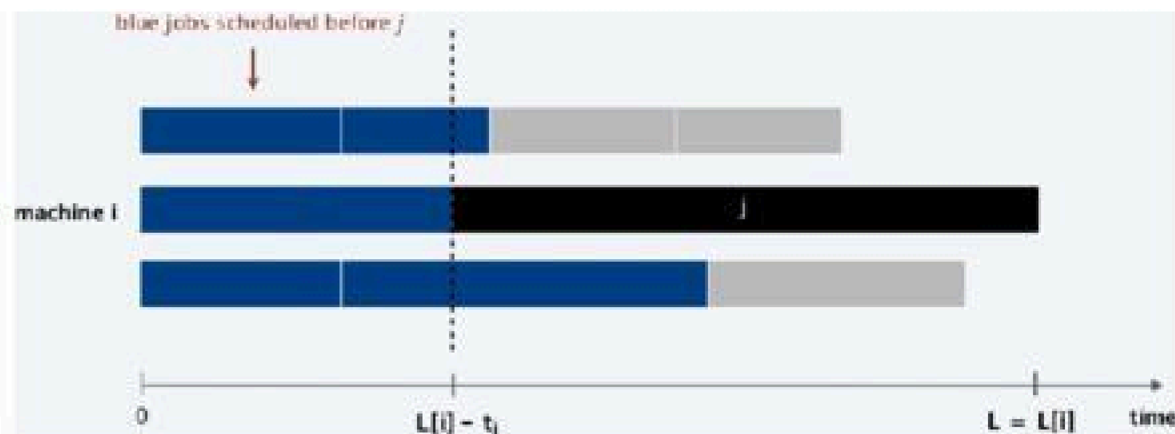
- The total processing time is $\sum_k t_k$.
- One of m machines must do at least a $1/m$ fraction of total work.

Greedy for LOAD-BALANCE: analysis

Bottleneck machine. Machine that has highest load after dispatching.

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 $L[i] - t_j$; hence $L[i] - t_j \leq L[k]$ for all $1 \leq k \leq m$.



Greedy for LOAD-BALANCE: 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 $L[i] - t_j$; hence $L[i] - t_j \le L[k]$ for all $1 \le k \le m$.
- Sum inequalities over all k and divide by m:

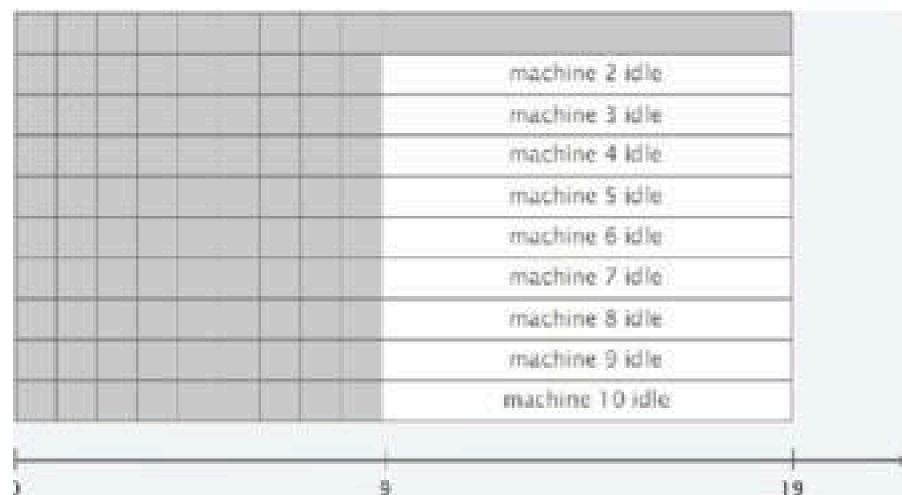
$$L[i] - t_j \le \frac{1}{m} \sum_k L[k] = \frac{1}{m} \sum_k t_k \le L^*$$

- Now, $L = L[i] = \underbrace{(L[i] - t_j)}_{\le L^*} + \underbrace{t_j}_{\le L^*} \le 2L^*$.

Greedy for LOAD-BALANCE: tightness

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

Ex: m machines, first $m(m{-}1)$ jobs have length 1, last job has length $$m.



- list scheduling makespan $= 19 = 2m - 1$
- optimal makespan $= 10 = m$

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 $(m, n, t_1, t_2, .., t_n)$

1. SORT jobs and renumber so that $t_1 \geq t_2 \geq .. \geq t_n$.
2. FOR $i = 1..m$:
   1. $L[i] = 0$;
   2. $S[i] = \emptyset$;
3. FOR $j = 1..n$:
   1. $i = \arg\min_k L[k]$;
   2. $S[i] = S[i] \cup \{j\}$;
   3. $L[i] = L[i] + t_j$;
4. RETURN $S[1], S[2], .., S[m]$;

LPT for Load balancing: analysis

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 $t_1 \geq t_2 \geq .. \geq t_{m+1}$.

- 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. [similar to proof for list scheduling]

- Consider load $L[i]$ of bottleneck machine i.
- Let j be last job scheduled on machine i.
  - assuming machine i has at least 2 jobs, we have $j \geq m+1$

≡

- Now, $L = L[i] = \underbrace{(L[i] - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2} L^*} \leq \frac{3}{2} L^*$.

LPT for Load balancing: analysis

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
- $2m$ jobs of length $m, m + 1, .., 2m{-}1$ and one more job of length m.
- Then, $L/L^* = ((m + (2m - 1)) + m)/(((3m - 1) * m + m)/m) = (4m - 1)/(3m)$

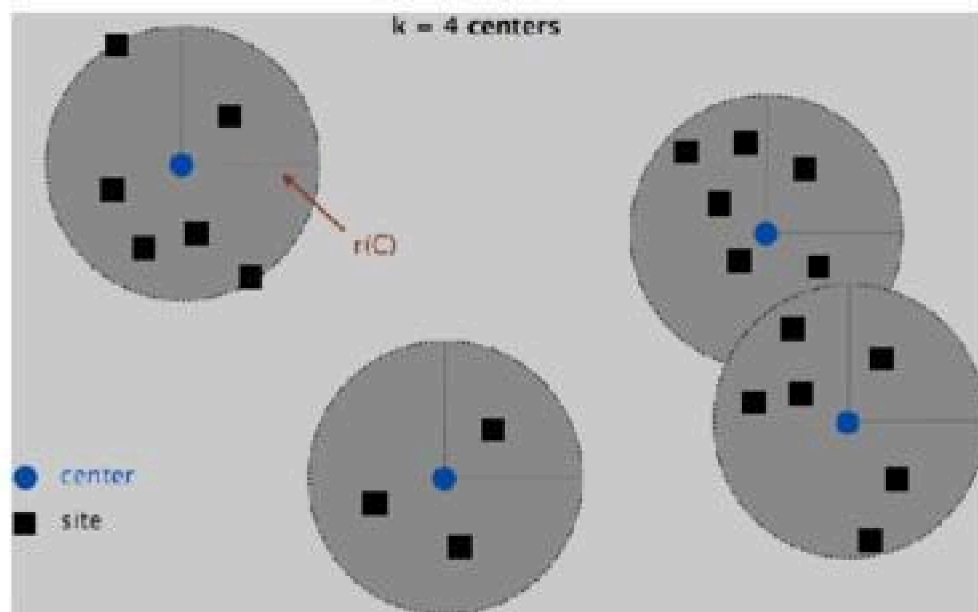
≡

Center selection

Center selection problem

Input. Set of n sites $s_1, .., s_n$ 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 $s_1, .., s_n$ 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 $|C| = k$.

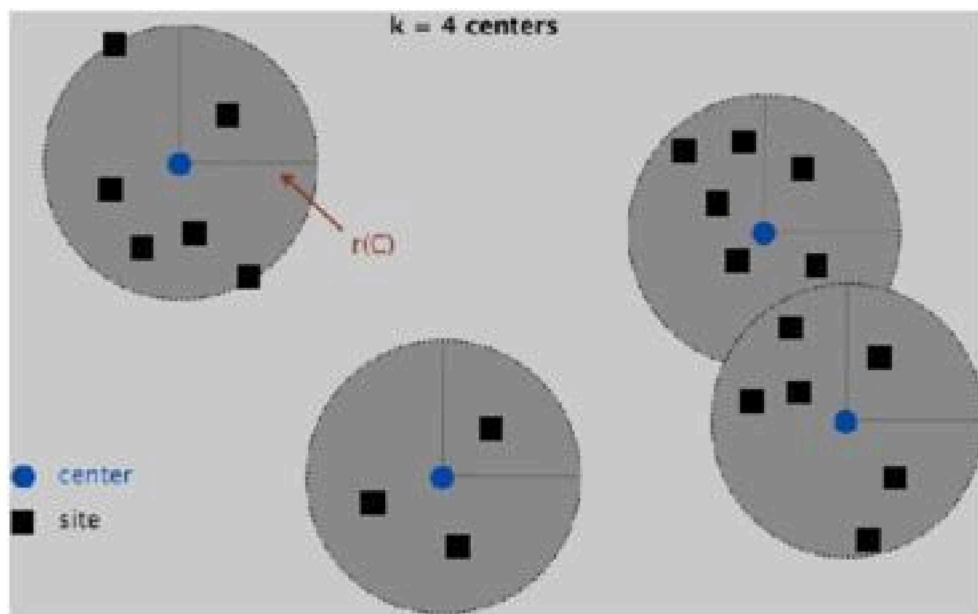
Distance function properties.

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

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!

- Ex. two seperated cluster of sites.

Center selection: greedy algorithm

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

GREEDY-CENTER-SELECTION $(k, n, s_1, s_2, .., s_n)$

1. C = ∅;
2. REPEAT k times
   1. Select a site s_i with maximum distance $dist(s_i, C)$;
   2. $C = C \cup s_i$;
3. RETURN C;

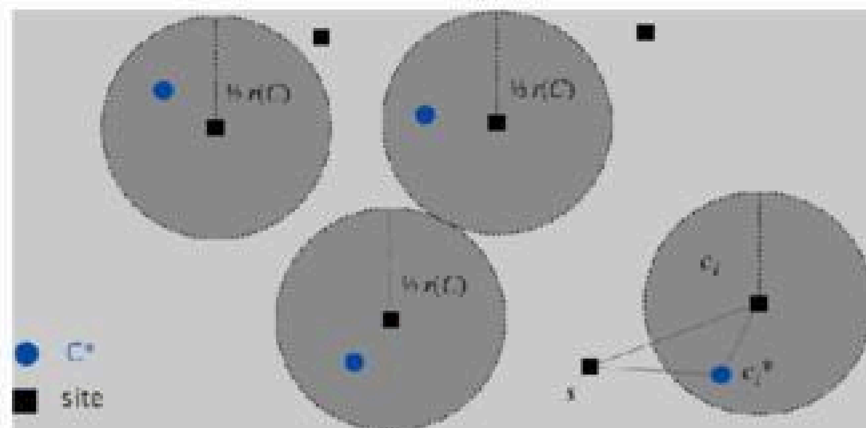
Property. Upon termination, all centers in C are pairwise at least $r(C)$ apart.
Pf. By construction, $r(C) = \max_i dist(s_i, C)$ = maximum distance $dist(s_i, C)$.

Greedy for center selection: analysis

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

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

- For each site $c_i \in C$, draw a ball of radius r around it.
- Consider a site s covered by $c_i \in C$, with $dist(s, c_i) > 2r$.
  - c_i covered by C^*: let c_i^* be the center paired with c_i.
  - If s covered by c_i^*, then $dist(s, c_i) > 2r \geq dist(s, c_i^*) + dist(c_i^*, c_i)$!
  - Otherwise, by farthest selection rule, $dist(s, c_j) > 2r, \forall c_j \in C$.
    - Need k center to cover $c_i \in C$, not possible to cover s.



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.

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

DOMINATING-SET $\leq_P$ CENTER-SELECTION

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

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

DOMINATING-SET. Each *vertex* is adjacent to at least one member of the DOMINATING-SET, as opposed to each *edge* being incident to at least one member of the VERTEX-COVER.

DOMINATING-SET $\leq_P$ CENTER-SELECTION

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

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

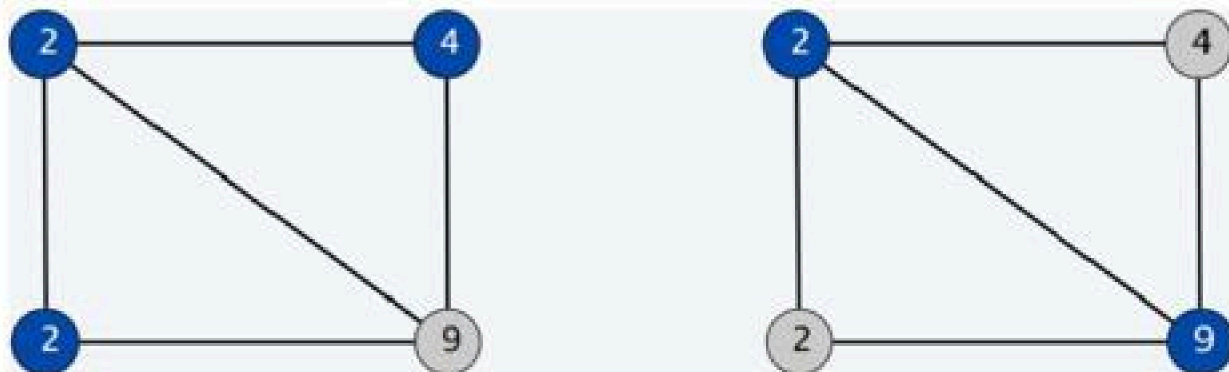
- Let $G = (V, E)$, k be an instance of DOMINATING-SET.
- Construct instance G' of CENTER-SELECTION with sites V and distances
  - $dist(u, v) = 1$ if $(u, v) \in E$
  - $dist(u, v) = 2$ if $(u, v) \notin E$
- Note that G' satisfies the triangle inequality.
- G has dominating set of size k iff there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

Pricing method: weighted 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 $G = (V, E)$ with vertex weights $w_i \geq 0$, find a vertex cover of minimum weight.



How to define "progress" in this setting?

- small weight w_i.
- cover lots of elements.

Greedy method

How to define "progress" in this setting?

- small weight w_i.
- cover lots of elements.

Option 1. $w_i/|S_i|$: "cost per element covered".

Option 2. $w_i/|S_i \cap R|$: we are only concerned with elements still left uncovered.

Greedy method

How to define "progress" in this setting?

- small weight w_i.
- cover lots of elements.

Option 1. $w_i/|S_i|$: "cost per element covered".

Option 2. $w_i/|S_i \cap R|$: we are only concerned with elements still left uncovered.

Greedy algorithm. Assignment.

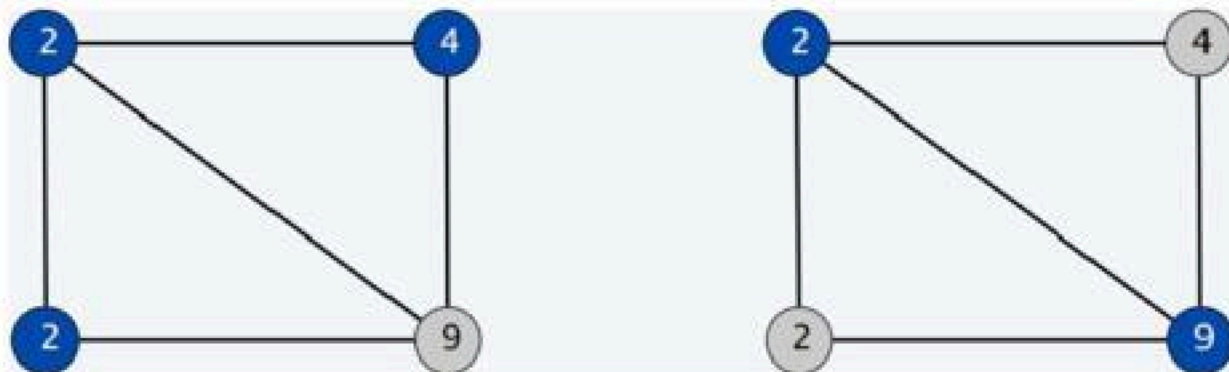
Greedy analysis. $O(\log d^*)$-approximation, $d^* = \max_i |S_i|$. Assignment.

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.

- ie. $\sum_{e=(i,j)} p_e \leq w_i$



Fairness lemma. For any vertex cover S and any fair prices p_e : $\sum_e p_e \leq w(S)$.
Pf. $\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i \leq w(S)$.

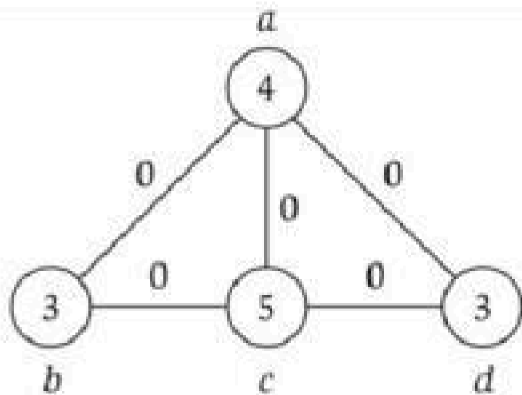
Pricing algorithm

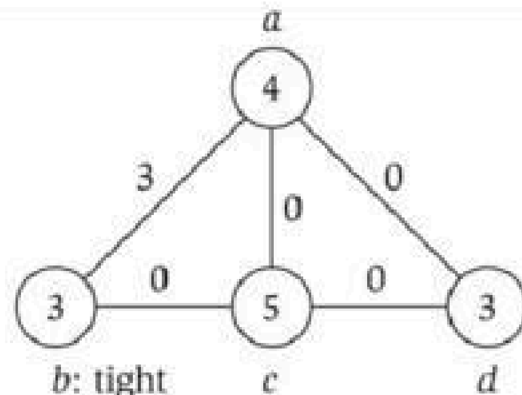
WEIGHTED-VERTEX-COVER (G, w)

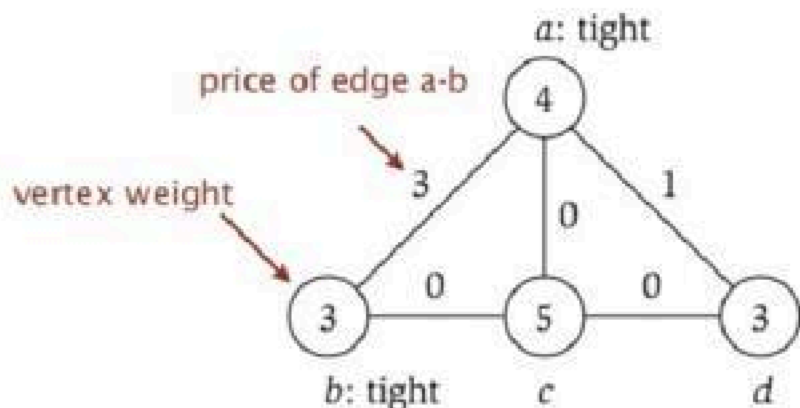
1. $S = \emptyset$;
2. FOREACH $e \in E$: $p_e = 0$;
3. WHILE (there exists an edge (i, j) such that neither i nor j is *tight*)
   1. Select such an edge $e = (i, j)$;
   2. Increase p_e as much as possible until i or j tight;
4. S = set of all tight nodes;
5. RETURN S;

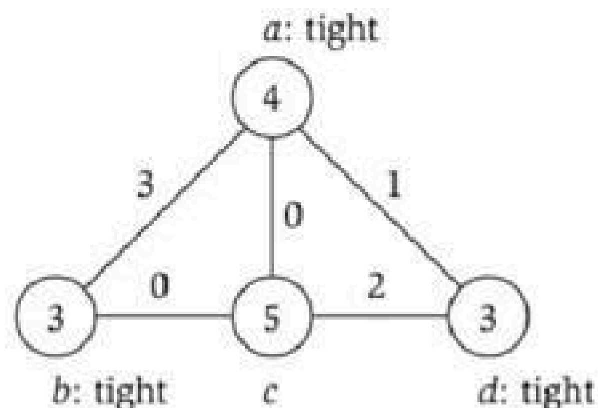
tight. $\sum_{e=(i,j)} p_e = w_i$

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 $w(S) \leq 2w(S^*)$.

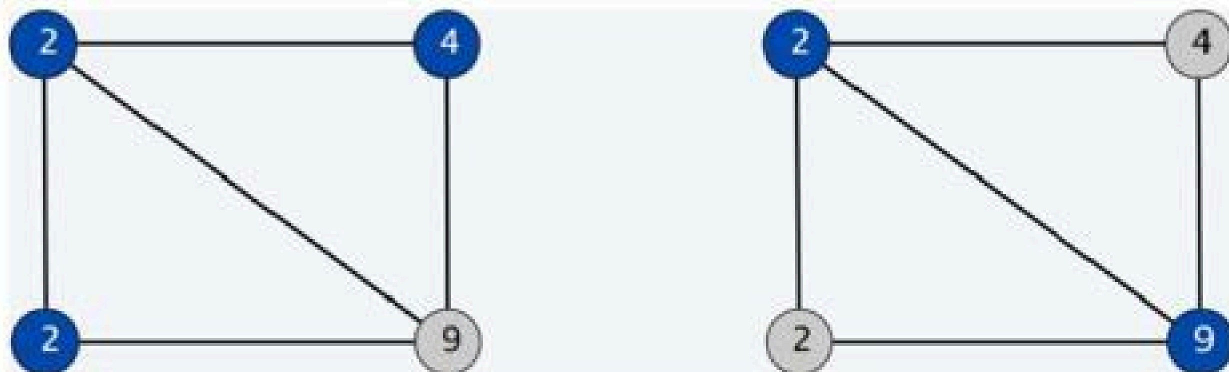
$$
\begin{aligned}
w(S) &= \sum_{i \in S} w_i && \text{all nodes tight} \\
&= \sum_{i \in S} \sum_{e=(i,j)} p_e && S \subseteq V \\
&\leq \sum_{i \in V} \sum_{e=(i,j)} p_e && \text{edge counted twice} \\
&= 2 \sum_{e \in E} pe \leq 2w(S^*) && \text{fairness lemma}
\end{aligned}
$$

LP rounding: weighted 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 $G = (V, E)$ with vertex weights $w_i \geq 0$, find a vertex cover of minimum weight.



Weighted vertex cover: ILP formulation

Weighted vertex cover. Given a graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a vertex cover of minimum weight.

Integer linear programming formulation.

- Model inclusion of each vertex i using a $0/1$ variable x_i.
  - Vertex covers in 1–1 correspondence with $0/1$ assignments: $S = \{i \in V : x_i = 1\}$.
- Objective function: minimize $\sum_i w_i x_i$.
- For every edge (i, j), must take either vertex i or j (or both): $x_i + x_j \geq 1$.

ILP formulation in math language

Weighted vertex cover. Integer linear programming formulation.

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

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

Integer linear programming

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

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

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

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

≡

linear programming

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

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

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

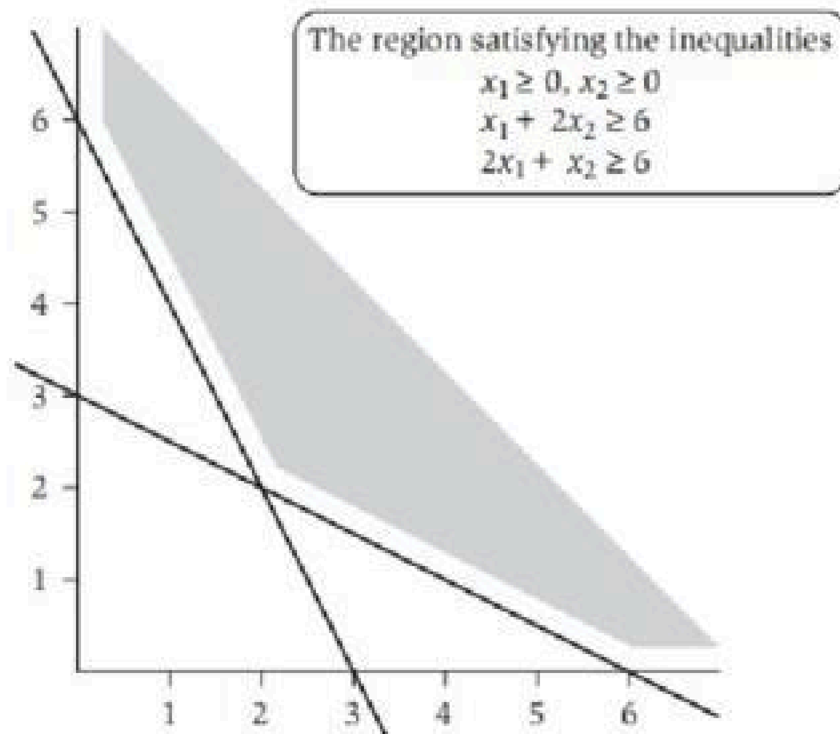
Linear. No $x^2, xy, \arccos(x), x(1-x)$, etc.

Simplex algorithm. [Dantzig 1947] Can solve LP in practice.

Ellipsoid algorithm. [Khachiyan 1979] Can solve LP in poly-time.

LP feasible region

LP geometry in 2D.



Weighted vertex cover: LP relaxation

Linear programming relaxation.

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

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

Weighted vertex cover: LP relaxation

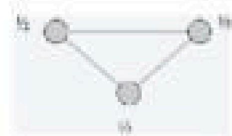
Linear programming relaxation.

$$\text{(ILP)} \ \min \ \sum_{i \in V} w_i x_i \qquad \text{s.t.} \quad x_i + x_j \geq 1 \quad (i,j) \in \qquad x_i \in \{0,1\} \quad i \in$$

$$\text{(LP)} \ \min \ \sum_{i \in V} w_i x_i \qquad \text{s.t.} \quad x_i + x_j \geq 1 \quad (i,j) \in E \qquad x_i \geq 0 \quad i \in V$$

Observation. Optimal value of LP is $\leq$ optimal value of ILP, ie. better.
Pf. LP has fewer constraints.

Note. LP solution x^* may not correspond to a vertex cover. (even if all weights are 1)



Weighted vertex cover: LP relaxation

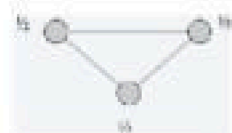
Linear programming relaxation.

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

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

Observation. Optimal value of LP is $\leq$ optimal value of ILP, ie. better.
Pf. LP has fewer constraints.

Note. LP solution x^* may not correspond to a vertex cover. (even if all weights are 1)



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

LP rounding algorithm

Lemma. 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 $x_i^* \geq \frac{1}{2} or x_j^* \geq \frac{1}{2}$ (or both) $\Rightarrow (i, j)$ covered.

Pf. [S has desired weight]

- Let S^* be optimal vertex cover. Then
  - $\sum_{i \in S^*} w_i \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i$

LP rounding algorithm

Lemma. 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 $x_i^* \geq \frac{1}{2} or x_j^* \geq \frac{1}{2}$ (or both) $\Rightarrow (i, j)$ covered.

Pf. [S has desired weight]

- 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-approximation algorithm.

Pf. Lemma + fact that LP can be solved in poly-time.

Weighted vertex cover inapproximability

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

Open research problem. Close the gap.

Theorem. [Kohot–Regev 2008] If Unique Games Conjecture is true, then no $(2 - \epsilon)$-approximation algorithm for WEIGHTED-VERTEX-COVER for any $\epsilon > 0$.

Open research problem. Prove the Unique Games Conjecture.

Generalized load balancing

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 $L_i = \sum_{j \in J_i} t_j$.

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.

Integer linear program and relaxation

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

$$
\begin{array}{llll}
\text{(ILP)} \ \min & L & \\
\text{s.t.} & \displaystyle\sum_i x_{ij} = t_j & & \forall j \in J \\
& \displaystyle\sum_j x_{ij} \le L & & \forall i \in M \\
& x_{ij} \in \{0, t_j\} & \forall j \in J, i \in M_j \\
& x_{ij} = 0 & \forall j \in J, i \notin M_j
\end{array}
$$

LP relaxation.

≡

$$
\begin{aligned}
\text{(LP) min} \quad & L \\
\text{s.t.} \quad & \sum_i x_{ij} = t_j && \forall j \in J \\
& \sum_j x_{ij} \leq L && \forall i \in M \\
& x_{ij} \geq 0 && \forall j \in J, i \in M_j \\
& x_{ij} = 0 && \forall j \in J, i \notin M_j
\end{aligned}
$$

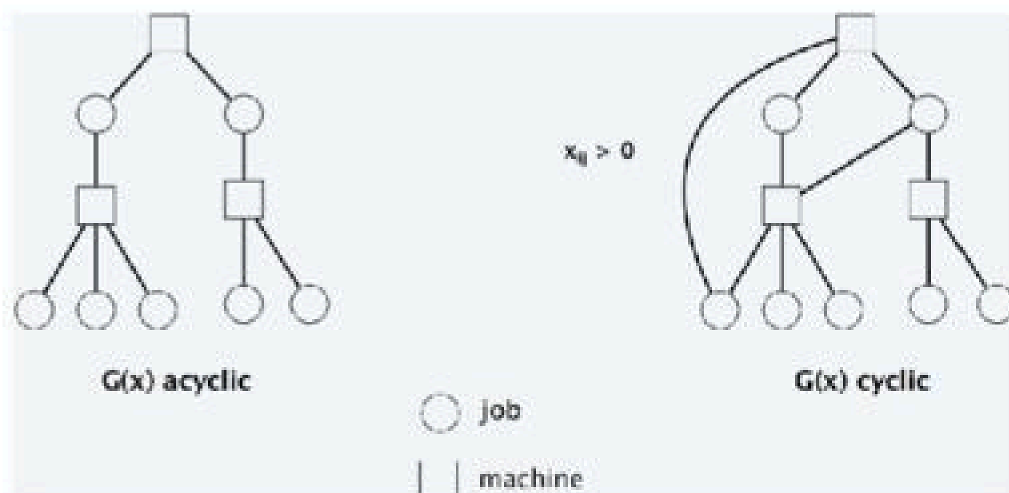
Lower bounds

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 ILP formulation.

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*.
Pf. (deferred)



Why a job can connect to multiple machines?

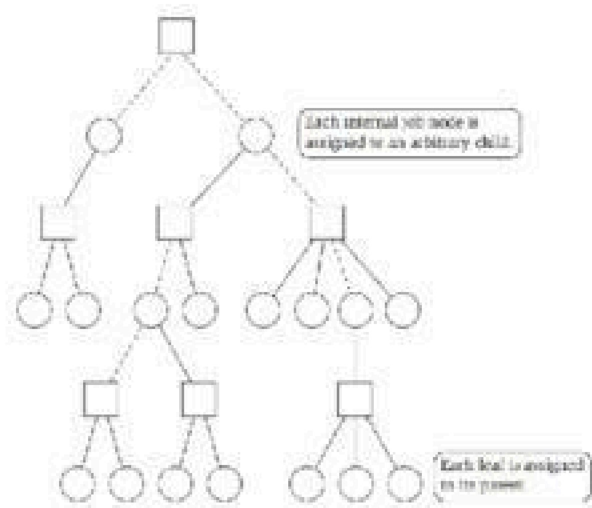
- LP solution may break the job into small fractions.

Generalized LB: 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.

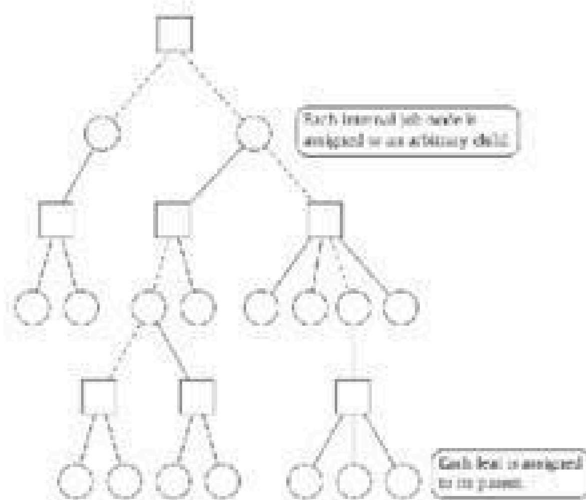

Each internal job node is assigned to an arbitrary child.
Each leaf is assigned to its parent.

Generalized LB: analysis

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

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

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 LB: 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:
  - parent: $t_{parent(i)} \leq L^*$ (LEMMA 1)
  - leaf nodes:

$$
\begin{aligned}
\sum_{j \in J(i)} t_j &= \sum_{j \in J(i)} x_{ij} && \text{LEMMA 5} \\
&\leq \sum_{j \in J} x_{ij} \leq L && \text{LP} \\
&\leq L^* && \text{LEMMA 2}
\end{aligned}
$$

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

Generalized LB: flow formulation

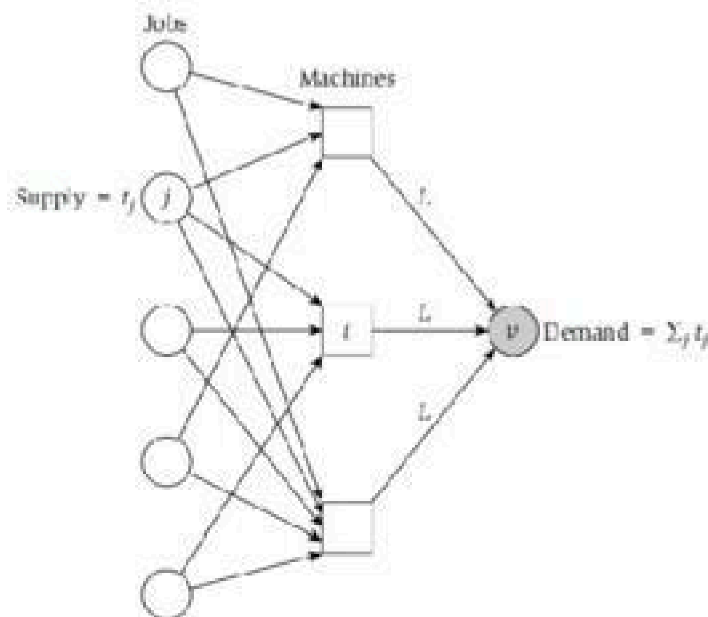
Flow formulation of LP.

$$\sum_i x_{ij} = t_j \qquad \forall j \in J$$

$$\sum_j x_{ij} \leq L \qquad \forall i \in M$$

$$x_{ij} \geq 0 \quad \forall j \in J, i \in M_j$$

$$x_{ij} = 0 \quad \forall j \in J, i \notin M_j$$

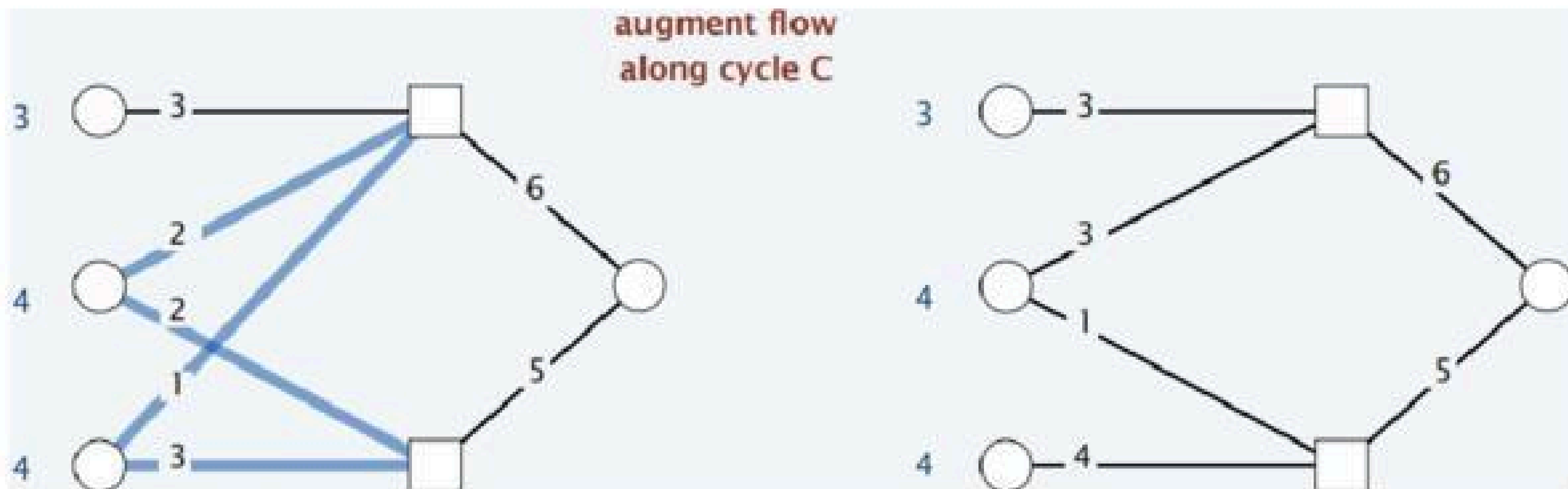


Observation. Solution to feasible flow problem with value L are in 1-to-1 correspondence with LP solutions of value L.

Generalized LB: 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 (maintain conservation).
- 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 $\mathcal{P} \neq \mathcal{NP}$, then no no ρ-approximation exists for any $\rho < 3/2$.

Knapsack problem

Polynomial-time approximation scheme

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

- 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 $w_i > 0$.
- 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

≡

Knapsack is NP-complete

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?

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:

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

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

$$v_i = w_i = u_i \quad \sum_{i \in S} u_i \leq U$$

$$V = W = U \quad \sum_{i \in S} u_i \leq U$$

Knapsack problem: DP I

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 $w - w_i$.

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

Theorem. Computes the optimal value in $O(nW)$ time.

- Not polynomial in input size.

- Polynomial in input size if weights are small integers.

Knapsack problem: DP II

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

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

Case 1. OPT does not select item i.

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

Case 2. OPT selects item i.

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

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

Knapsack problem: DP II (cont.)

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 $v \leq v_{max}$.
- It takes $O(1)$ time per subproblem.

Remark 1. Not polynomial in input size! (pseudo-polynomial)
Remark 2. Polynomial time if values are small integers.

Poly-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

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

Poly-time approximation scheme

Round up all values:

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

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

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.

Poly-time approximation scheme

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

Pf.

≡

$$
\begin{aligned}
\sum_{i \in S^*} v_i &\leq \sum_{i \in S^*} \bar{v}_i && \text{round up} \\
&\leq \sum_{i \in S} \bar{v}_i && \text{optimality} \\
&\leq \sum_{i \in S} (v_i + \theta) && \text{rounding gap} \\
&\leq \sum_{i \in S} v_i + n\theta && |S| \leq n \\
&= \sum_{i \in S} v_i + \frac{1}{2}\epsilon v_{max} && \theta = \epsilon v_{max}/2n \\
&\leq (1+\epsilon) \sum_{i \in S} v_i && v_{max} \leq 2 \sum_{i \in S} v_i
\end{aligned}
$$

Poly-time approximation scheme

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

Pf.

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

$$\hat{v}_{max} = \lceil \frac{v_{max}}{\theta} \rceil = \lceil \frac{2n}{\epsilon} \rceil$$