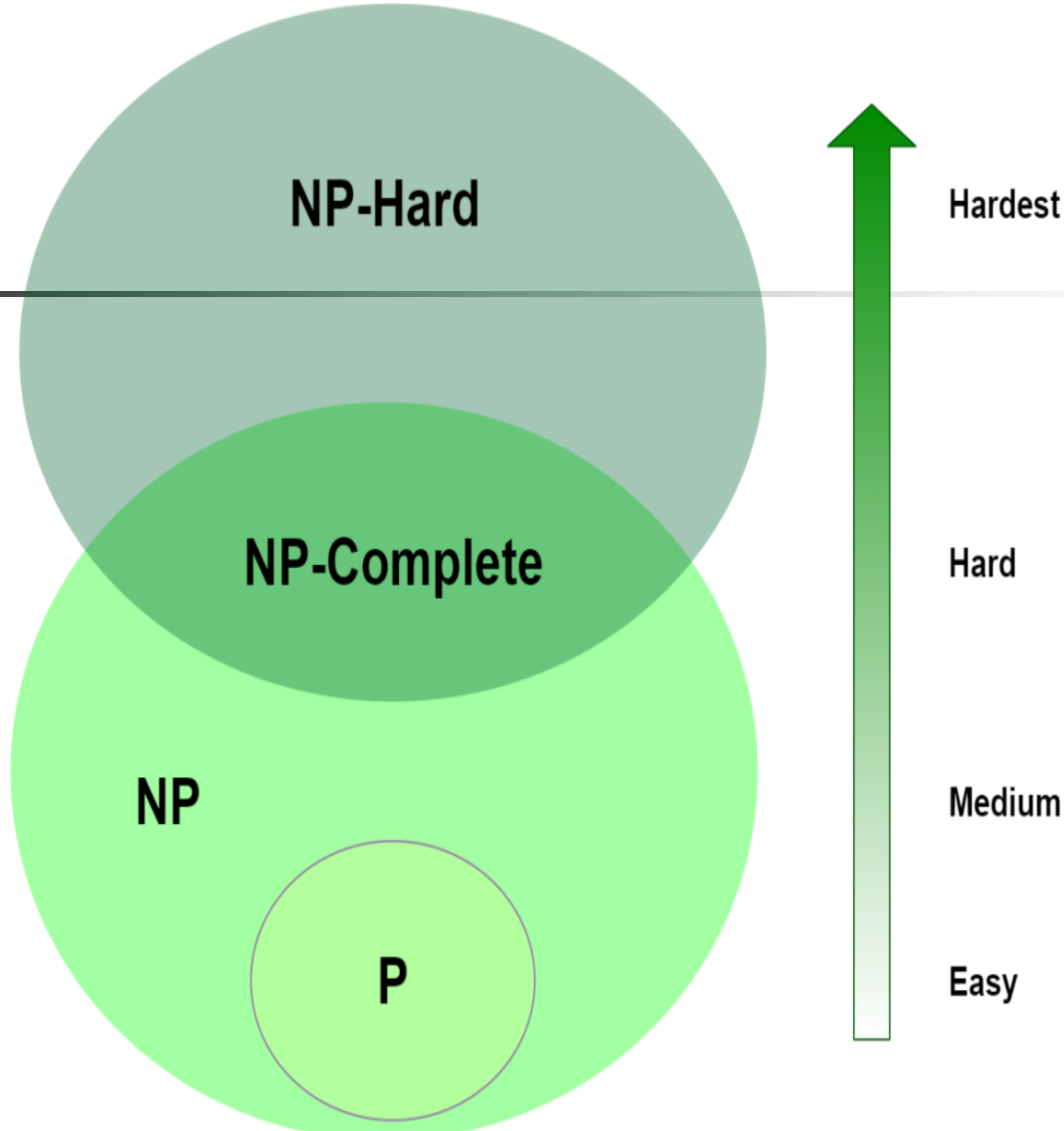
Approximation Algorithms


NP-Hard
NP-Complete
NP
P
Hardest
Hard
Medium
Easy

NP-Completeness

"I can't find an efficient algorithm, but neither can all these famous people."

Coping With NP-Hardness

Brute-force Algorithms.

- Develop clever enumeration strategies.
- Guaranteed to find optimal solution.
- No guarantees on the running time.

Heuristics.

- Develop intuitive algorithms.
- Guaranteed to run in polynomial time.
- No guarantees on the quality of solution.

Approximation Algorithms.

- Guaranteed to run in polynomial time.
- Guaranteed to find "high quality" solution, within 1% of optimum.

Obstacle: We need to prove a solution's value is close to the optimum, without even knowing what optimum value is!

Coping with NP-completeness

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

A. Sacrifice one of three desired features.

i. Runs in polynomial time.
ii. Solves arbitrary instances of the problem.
iii. Finds optimal solution to problem.

ρ-approximation algorithm.

- Runs in polynomial time.
- Solves arbitrary instances of the problem
- Finds solution that is within ratio ρ of optimum.

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

Approximation Algorithms

- The best algorithm for solving the NP-complete problem requires the exponential time in the worst-case. It is too time-consuming.
- To reduce the time required for solving a problem, we can relax the problem, and obtain a feasible solution "close" to optimal solution.

Approximation Algorithms

- One compromise is to use the “Heuristic“ Solutions.
- “Heuristic” may be interpreted as an “Educated Guess”.
- Approximation Algorithms return Near Optimal Solutions.
- Need to find an Approximation Ratio Bound for algorithm.

Approximation Ratio Bound

We say an approximation algorithm for the problem has a ratio bound of $\rho(n)$ if for any input size n, the cost C of the solution produced by the approximation algorithm is within a factor of $\rho(n)$ of the C^* of the optimal solution:

$$\max\{\frac{C}{C^*}, \frac{C^*}{C}\} = \rho(n)$$

This applies for both minimization and maximization problems.

Performance Guarantees

- An approximation algorithm is bounded by $\rho(n)$ if, for all input of size n, the cost c of the solution obtained by an approximation algorithm is within a factor $\rho(n)$ of c^* of an optimal solution.

ρ-approximation algorithm

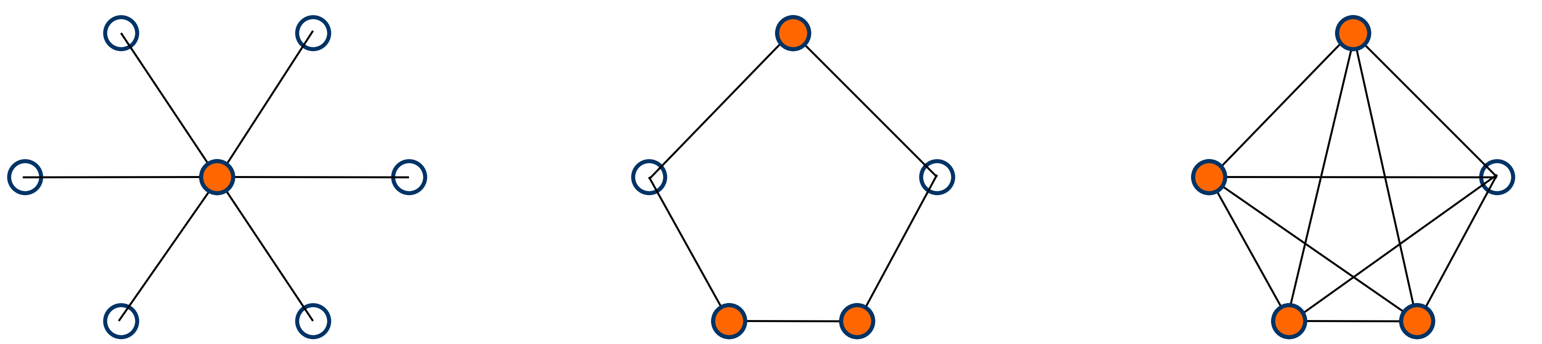
- An approximation algorithm with an approximation ratio bound of ρ is referred to as ρ-approximation algorithm or also known as $(1+\varepsilon)$-approximation algorithm.

- Note that ρ is always > 1 and $\varepsilon = \rho - 1$.

Vertex Cover

Vertex Cover: a subset of vertices which "covers" every edge.
An edge is covered if one of its endpoint is chosen.

The Minimum Vertex Cover Problem:
Find a vertex cover with minimum number of vertices.

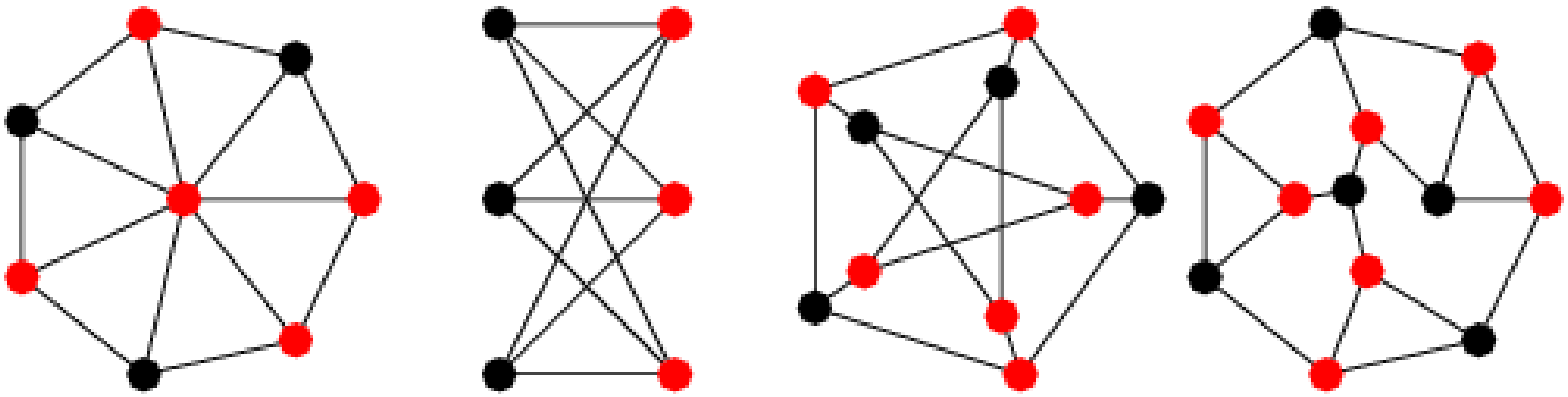


Vertex Cover Problem

- Let G=(V, E). The subset S of V that meets every edge of E is known as the Vertex Cover.

- The Vertex Cover Problem is solved for finding a vertex cover of the Minimum size. It is NP-Hard Problem or Optimization Problem version of an NP-Complete Decision Problem.

Examples of Vertex Cover



Vertex Cover Problem

APPROX_VERTEX_COVER(G)

$C \leftarrow \phi$

$E' \leftarrow E(G)$

while $E' \neq \phi$

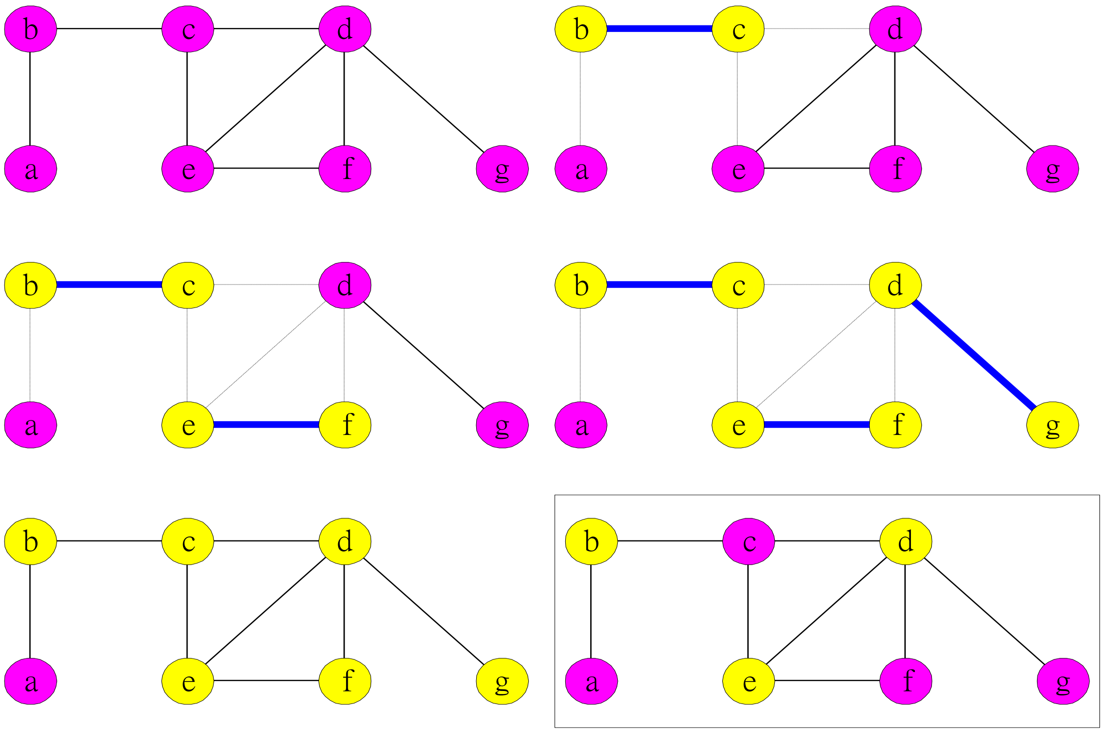
  do let (u,v) be an arbitrary edge of E'

    $C \leftarrow C \cup \{u,v\}$

    remove from E' every edge incident on either u or v

return C

Vertex Cover Problem



Complexity: O(E)

Vertex Cover Problem

Theorem: APPROX_VERTEX_COVER has ratio bound of 2.

Proof.

C*: optimal solution
C: approximate solution
A: the set of edges selected in step 4

Let A be the set of selected edges.

$|C| = 2|A|$ When one edge is selected, 2 vertices are added into C.

$|A| \leq |C^*|$ No two edges in A share a common endpoint.

$\Rightarrow |C| \leq 2|C^*|$

Traveling Salesperson Problem

Traveling Salesperson Problem (TSP) asks for the shortest Hamiltonian cycle in a weighted undirected graph.

Consider G be an arbitrary undirected graph with n vertices.

Length Function $l(e) = \begin{cases} 1 & \text{if } e \text{ is an edge in } G \\ 2 & \text{otherwise} \end{cases}$ for K_n

Where, G has a Hamiltonian cycle then there is an Hamiltonian cycle in K_n whose length is exactly n

Traveling Salesperson Problem is NP-hard (NP-complete) even if all the edge lengths are 1 or 2 due to polynomial time reduction from Hamiltonian cycle to this type of Traveling salesperson problem.

Traveling Salesperson Problem

We can replace the values in length function by any values we like

Length Function $l(e) = \begin{cases} 1 & \text{if } e \text{ is an edge in } G \\ n & \text{otherwise} \end{cases}$

G has a Hamiltonian cycle then there is an Hamiltonian cycle in K_n whose length is exactly *n* or has length at least *2n.*

Thus if we can approximate the shortest traveling salesman tour within a factor of 2 in polynomial time we would have a polynomial time algorithm for the Hamiltonian cycle problem

For any function *f(n)* that can be computed in polynomial in *n,* there is no polynomial time *f(n)* approx to TSP on general weighted graph unless P=NP.

TSP: A Special Case

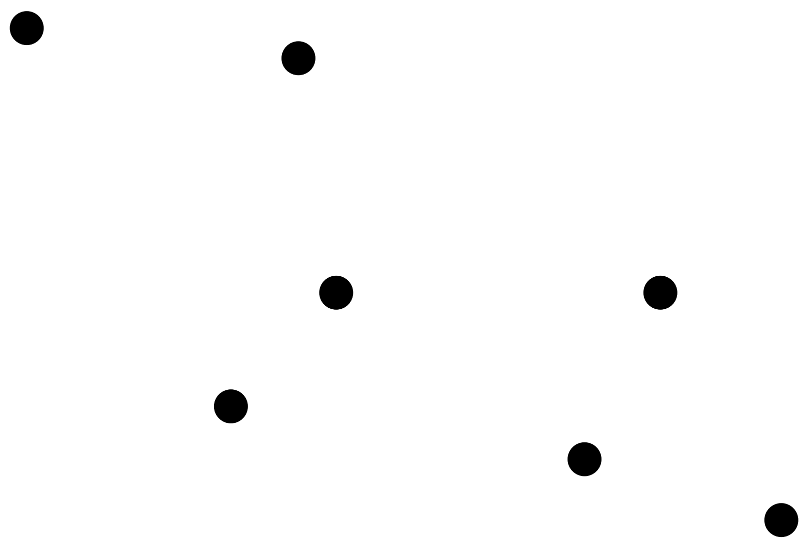
Edge lengths satisfy triangular inequality
$l(u,v) \leq l(u,w) + l(w,v)$

This is true for geometric graph

- Compute Minimum Spanning Tree *T* of the weighted input graph
- Depth First Traversal (Depth First Search) of the MST *T*
- Numbering the vertices in order that we first encounter them
- Return the cycle found by visiting vertices as per this numbering

TSP: A Special Case

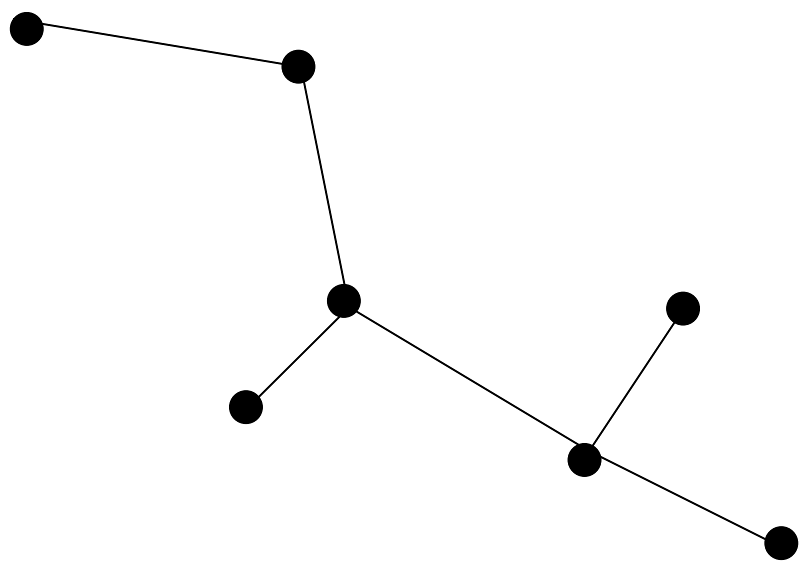
Demonstration



Set of points distributed in 2D

TSP: A Special Case

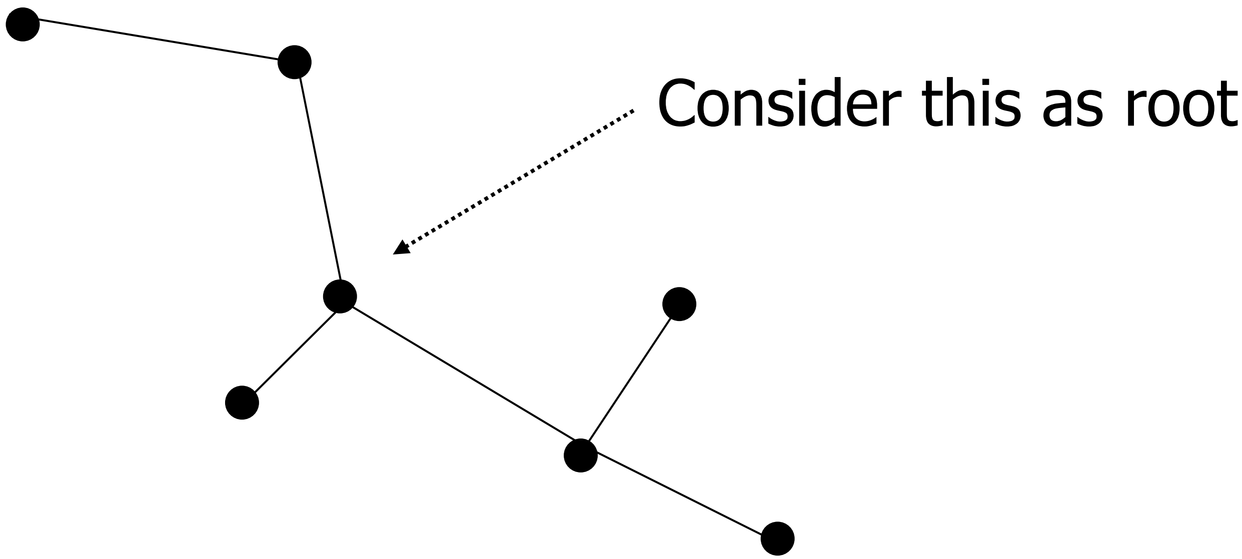
Demonstration



Minimum Spanning Tree

TSP: A Special Case

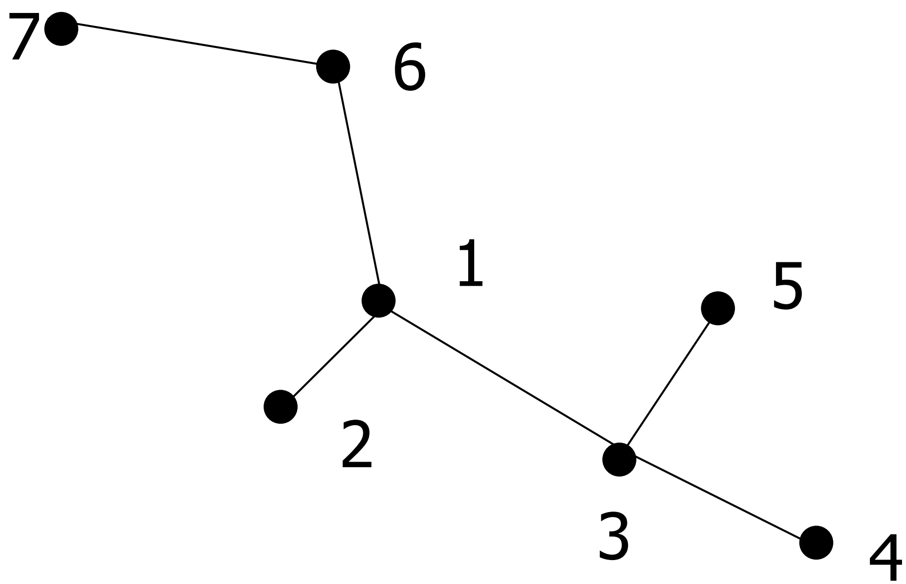
Demonstration



Depth First Traversal

TSP: A Special Case

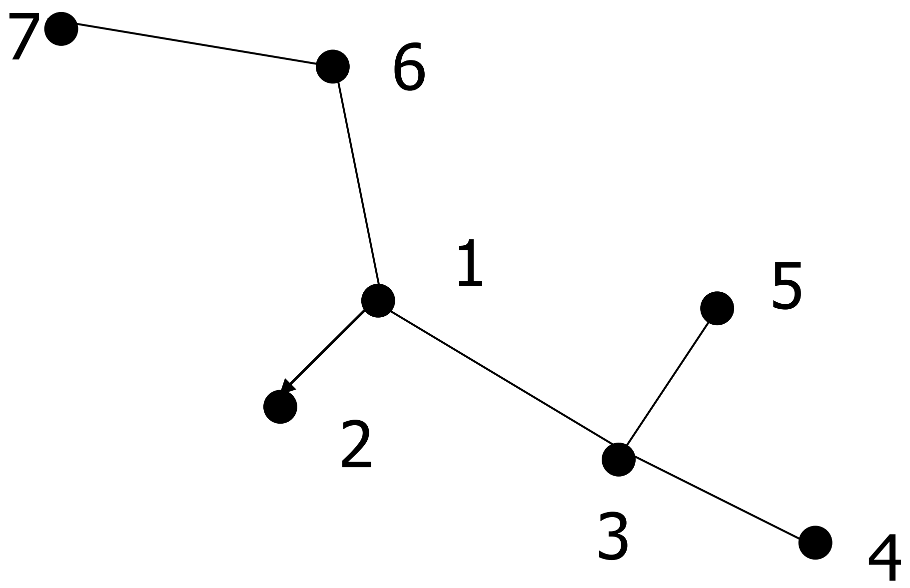
Demonstration



Depth First Traversal and Numbering of Vertices

TSP: A Special Case

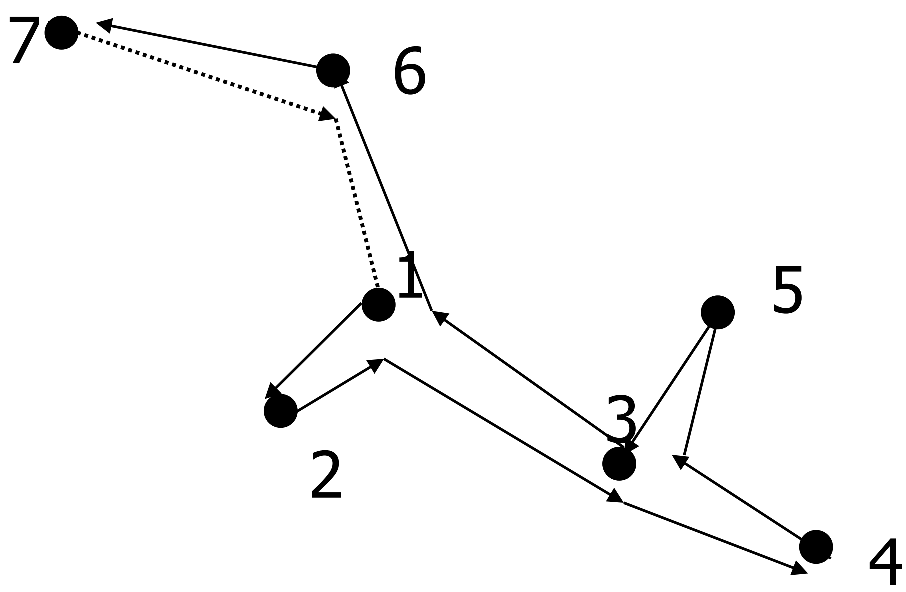
Demonstration



Traveling Salesperson Tour

TSP: A Special Case

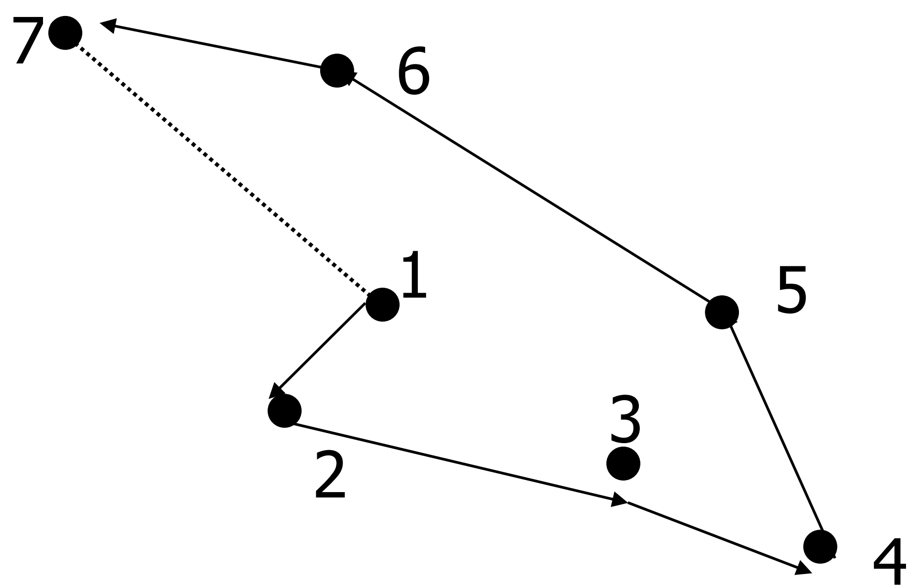
Demonstration



Traveling Salesperson Tour with Cost 2.MST

TSP: A Special Case

Demonstration



Traveling Salesperson Tour with Reduced Cost ≤ 2.MST

TSP: A Special Case

Output Quality :

Cost of the tour using this algorithm

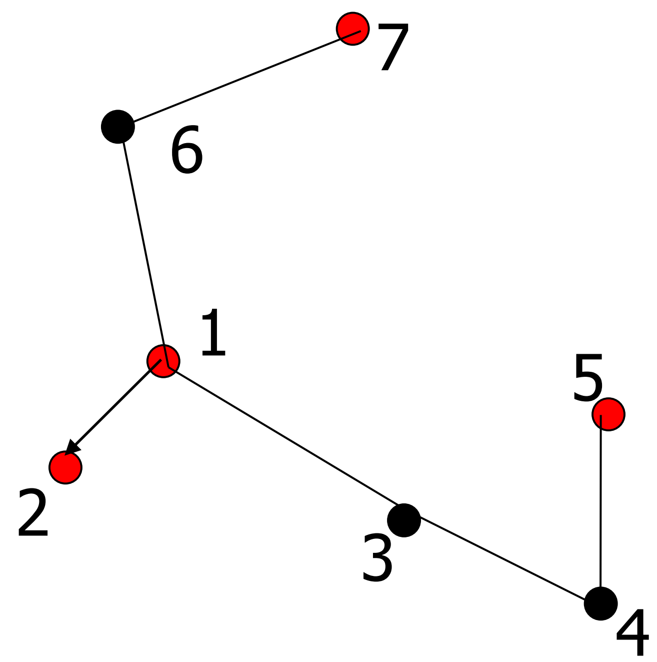
$\leq$ 2* cost of minimum spanning tree

$\leq$ 2* cost of optimal solution

Conclusion: The algorithm outputs 2 approximation of the minimum traveling salesperson problem

TSP: An Improved Heuristic

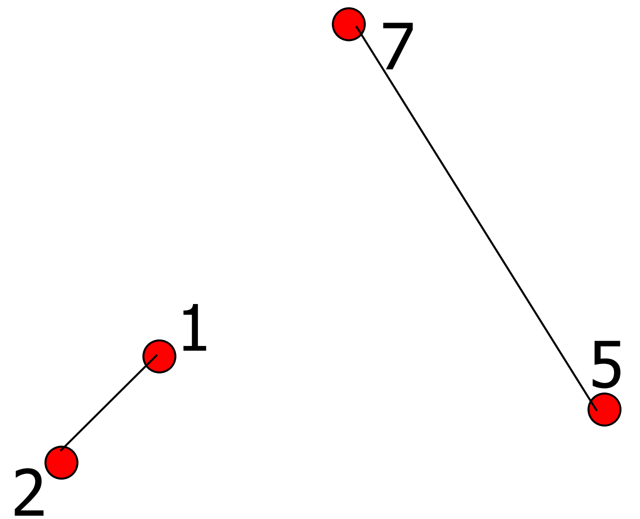
Locate odd degree vertices in minimum spanning tree (MST)



Number of odd degree vertices is even

TSP: An Improved Heuristic

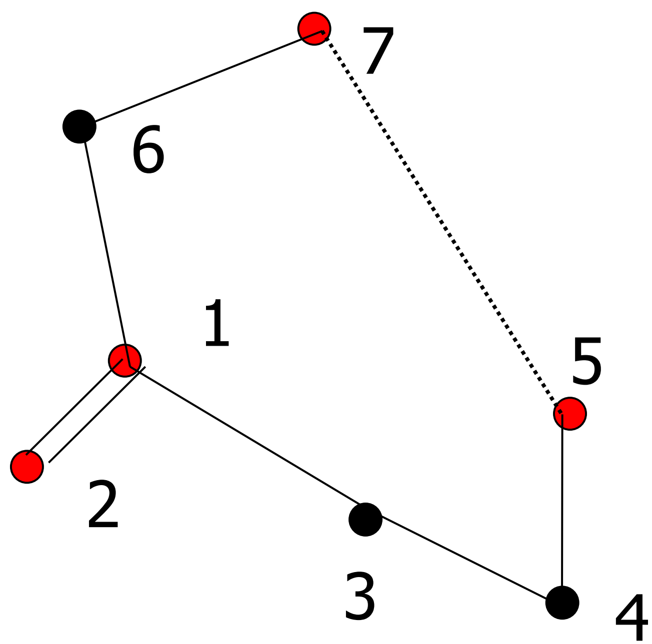
Locate odd degree vertices in minimum spanning tree (MST)



Perfect matching of odd degree vertices

TSP: An Improved Heuristic

Locate odd degree vertices in minimum spanning tree (MST)



Merging the perfect edges with MST

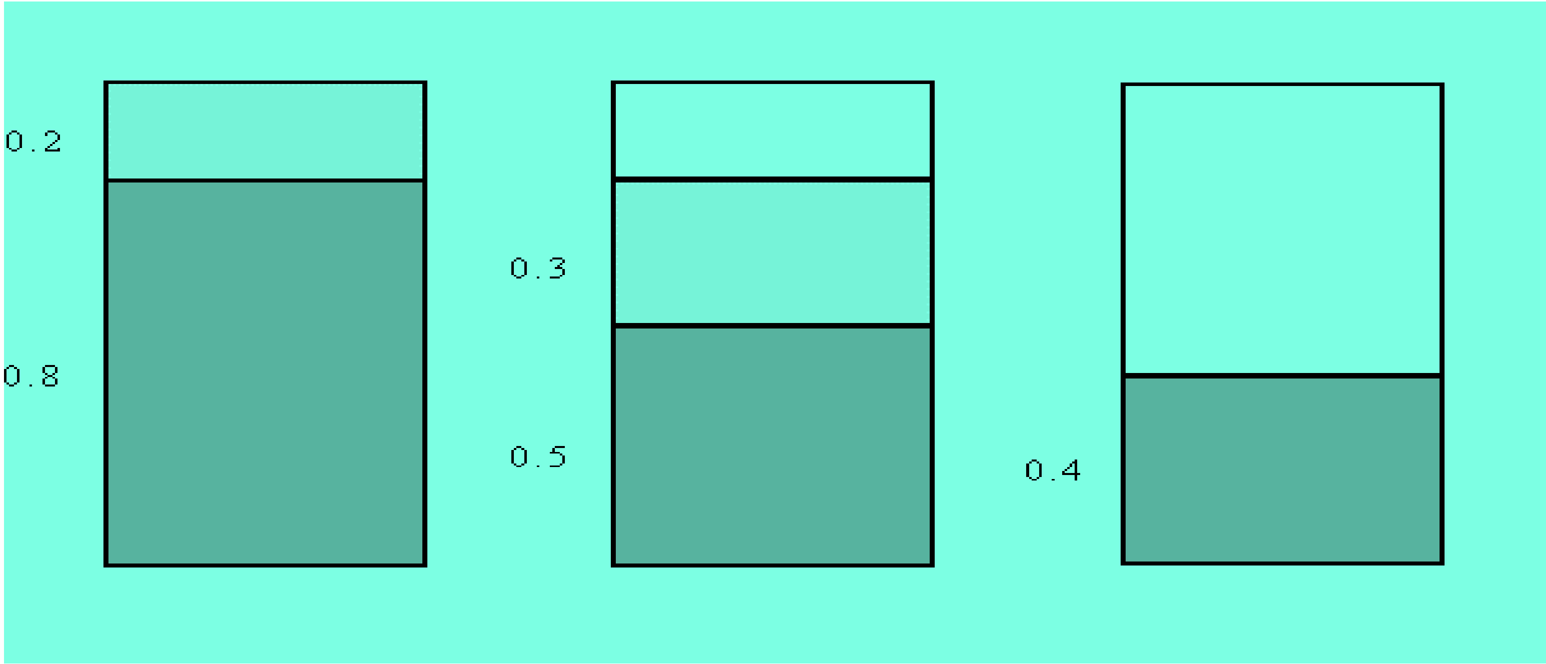
Bin Packing Problem

- Given n items of sizes a_1, a_2, …, a_n, $0 < a_i \leq 1$ for $1 \leq i \leq n$, which are to be placed in bins of unit capability, then the bin packing problem can be solved by determining the minimum number of bins to accommodate all the items.

- Consider the items of different sizes with lengths of time of executing different jobs on a standard processor, then we need to use minimum number of processors which can finish all the jobs within a fixed time. // Assume that the longest job takes one unit time i.e. equal to fixed time.

Example of Bin Packing Problem

- Ex. Given n = 5 items with sizes 0.3, 0.5, 0.8, 0.2, 0.4, then the Optimal Solution is 3 Bins.



The Bin Packing Problem is NP-Hard Optimization Problem.

An Approximation Algorithm for the Bin Packing Problem

- An Approximation Algorithm: (First-Fit (FF)) place the item i into the lowest-indexed bin which can accommodate the item i.
- OPT: The number of bins of the Optimal Solution
- FF: The number of bins in the First-Fit Algorithm
- $C(B_i)$: The sum of the sizes of items packed in the bin B_i in the First-Fit Algorithm
- Let FF=m.

An Approximation Algorithm for the Bin Packing Problem

- OPT $\geq \left\lceil \sum_{i=1}^{n} a_i \right\rceil$, ceiling of sum of sizes of all items
- $C(B_i) + C(B_{i+1}) > 1$ (a)(Otherwise, the items in B_{i+1} will be put in B_i). C(Bi): The sum of sizes of items packed in bin Bi
- $C(B_1) + C(B_m) > 1$ (b)(Otherwise, the items in B_m will be put in B_1.)
- For m nonempty bins,

  $C(B_1)+C(B_2)+\ldots+C(B_m) > m/2$, (a)+(b) for i=1,..,m

  $\Rightarrow$ FF = m < $2\sum_{i=1}^{m} C(B_i) = 2\sum_{i=1}^{n} a_i \leq 2$ OPT

  FF < 2 OPT

Load balancing

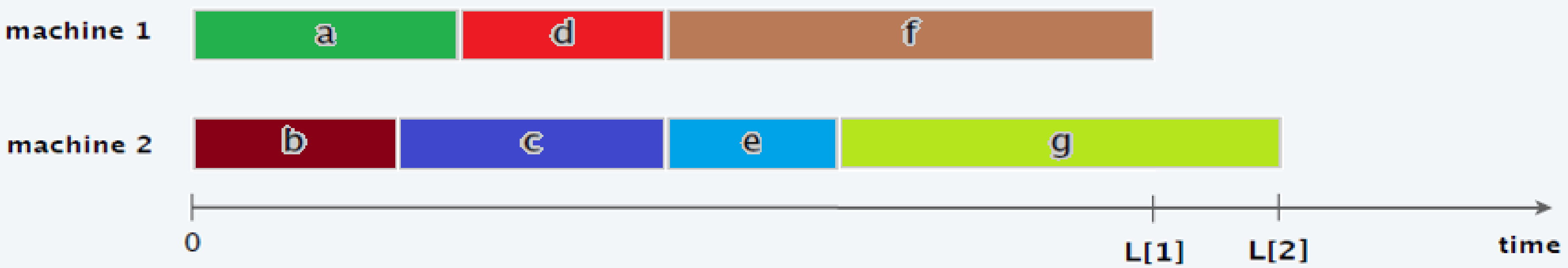
Input. m identical machines; $n \geq m$ jobs, job j has processing time t_j.

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

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

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

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

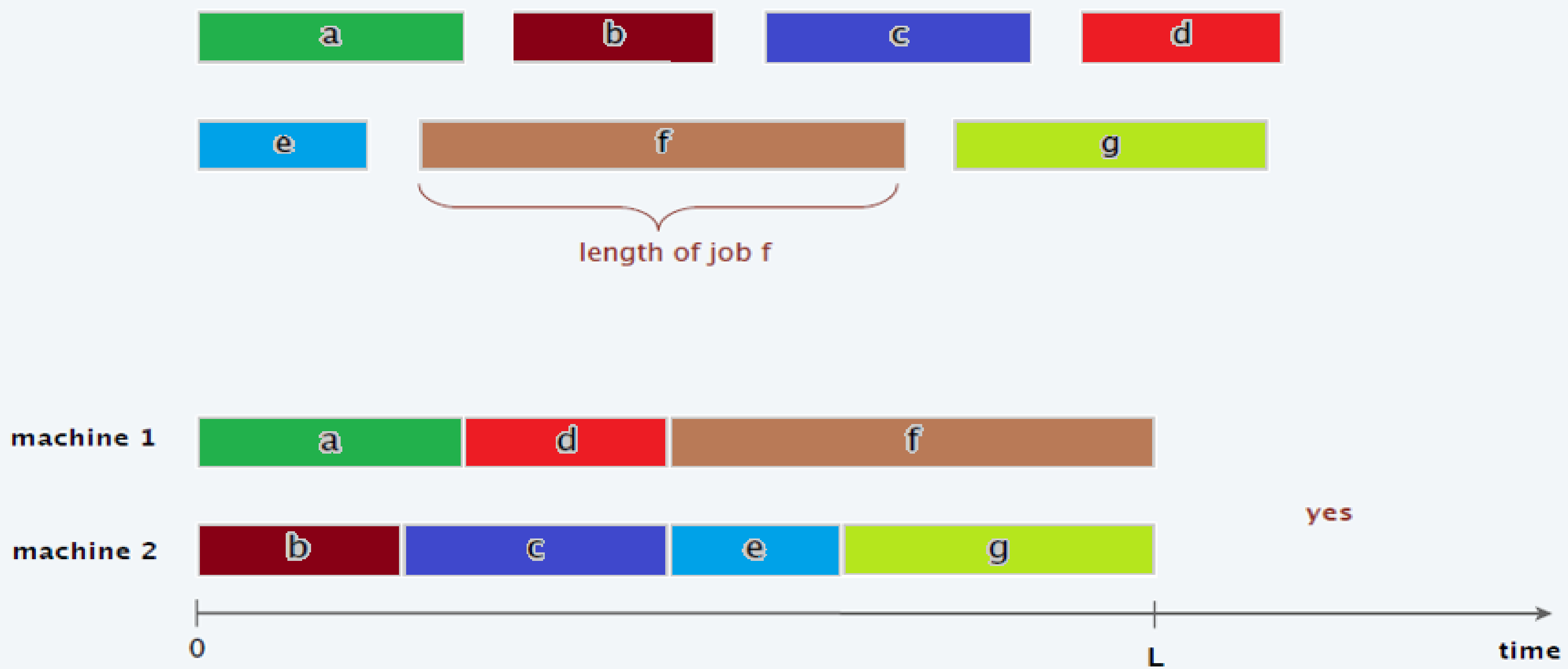


Load balancing on 2 machines is NP-Hard/NP-Complete Problem

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

Pf. PARTITION $\leq_P$ LOAD-BALANCE.

NP-Complete Optimization Problem



Load balancing: list scheduling

List-scheduling algorithm.

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

LIST-SCHEDULING $(m, n, t_1, t_2, \ldots, t_n)$

FOR $i = 1$ TO m

  $L[i] \leftarrow 0.$ ← load on machine i

  $S[i] \leftarrow \varnothing.$ ← jobs assigned to machine i

FOR $j = 1$ TO n

  $i \leftarrow \text{argmin}_k\, L[k].$ ← machine i has smallest load

  $S[i] \leftarrow S[i] \cup \{j\}.$ ← assign job j to machine i

  $L[i] \leftarrow L[i] + t_j.$ ← update load of machine i

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

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

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. 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.

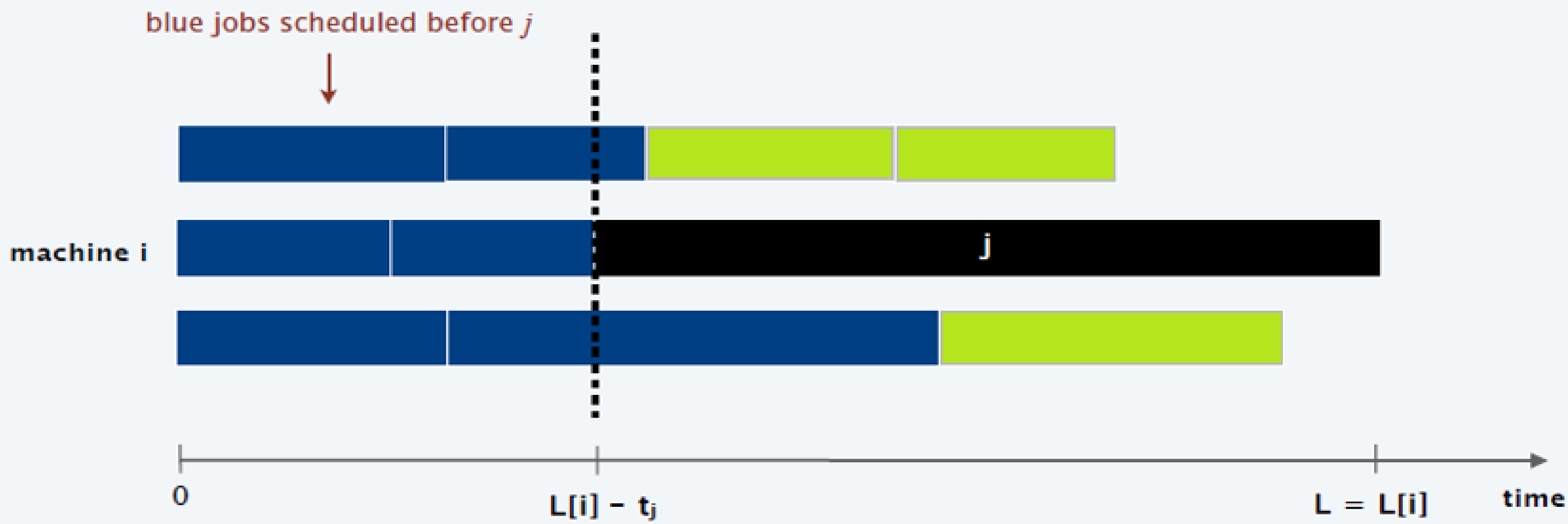
- The total processing time is $\Sigma_k t_k$.
- One of m machines must do at least a $1 / m$ fraction of total work. ▪

Load balancing: list scheduling analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load $L[i]$ of bottleneck machine i. ← machine that ends up with highest load

- 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 $1 \leq k \leq m$.



Load balancing: list scheduling analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load $L[i]$ of bottleneck machine i. ← machine that ends up with highest load

- 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:

$$
\begin{aligned}
L[i] - t_j &\le \frac{1}{m} \sum_k L[k] \\
&= \frac{1}{m} \sum_k t_k \\
&\le L^*. \quad \text{(Lemma 2)}
\end{aligned}
$$

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

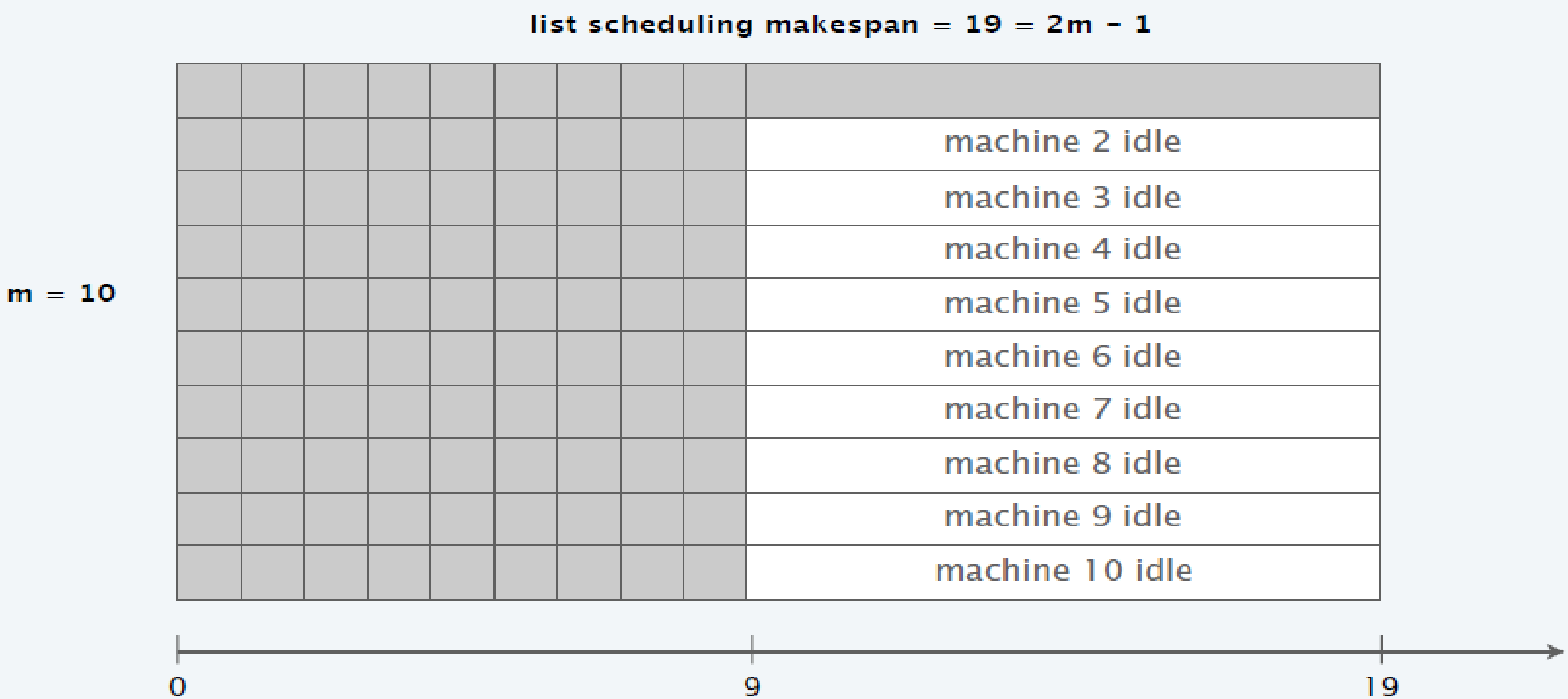
above inequality Lemma 1

Load balancing: list scheduling analysis

Q. Is our analysis tight?

A. Essentially yes.

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

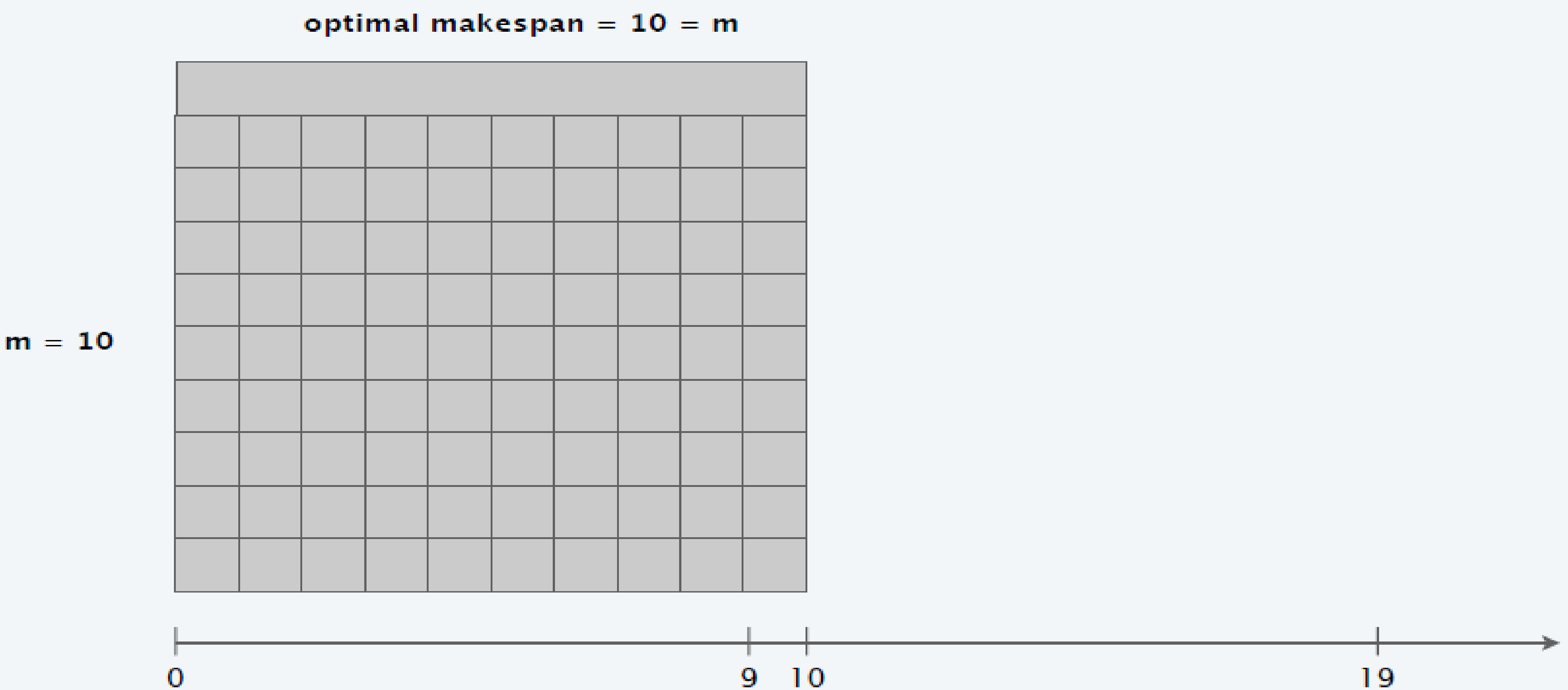


Load balancing: list scheduling analysis

Q. Is our analysis tight?

A. Essentially yes.

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



Load balancing: LPT rule

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

LPT-LIST-SCHEDULING $(m, n, t_1, t_2, \ldots, t_n)$

SORT jobs and renumber so that $t_1 \geq t_2 \geq \ldots \geq t_n$.

FOR $i = 1$ TO m

$L[i] \leftarrow 0.$ ⟵ load on machine i

$S[i] \leftarrow \varnothing.$ ⟵ jobs assigned to machine i

FOR $j = 1$ TO n

$i \leftarrow \operatorname{argmin}_k L[k].$ ⟵ machine i has smallest load

$S[i] \leftarrow S[i] \cup \{ j \}.$ ⟵ assign job j to machine i

$L[i] \leftarrow L[i] + t_j.$ ⟵ update load of machine i

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

Load balancing: LPT rule

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

Pf. Any solution must schedule that job. ▪

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

Pf.

- Consider processing times of first m+1 jobs $t_1 \geq t_2 \geq \ldots \geq t_{m+1}$.
- 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 $j \geq m + 1$

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

as before

Lemma 3 (since $t_{m+1} \geq t_j$)

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.
- Then, $L / L^* = (4m - 1) / (3m)$

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.

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

LP relaxation.

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

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 $L^* \geq L$.

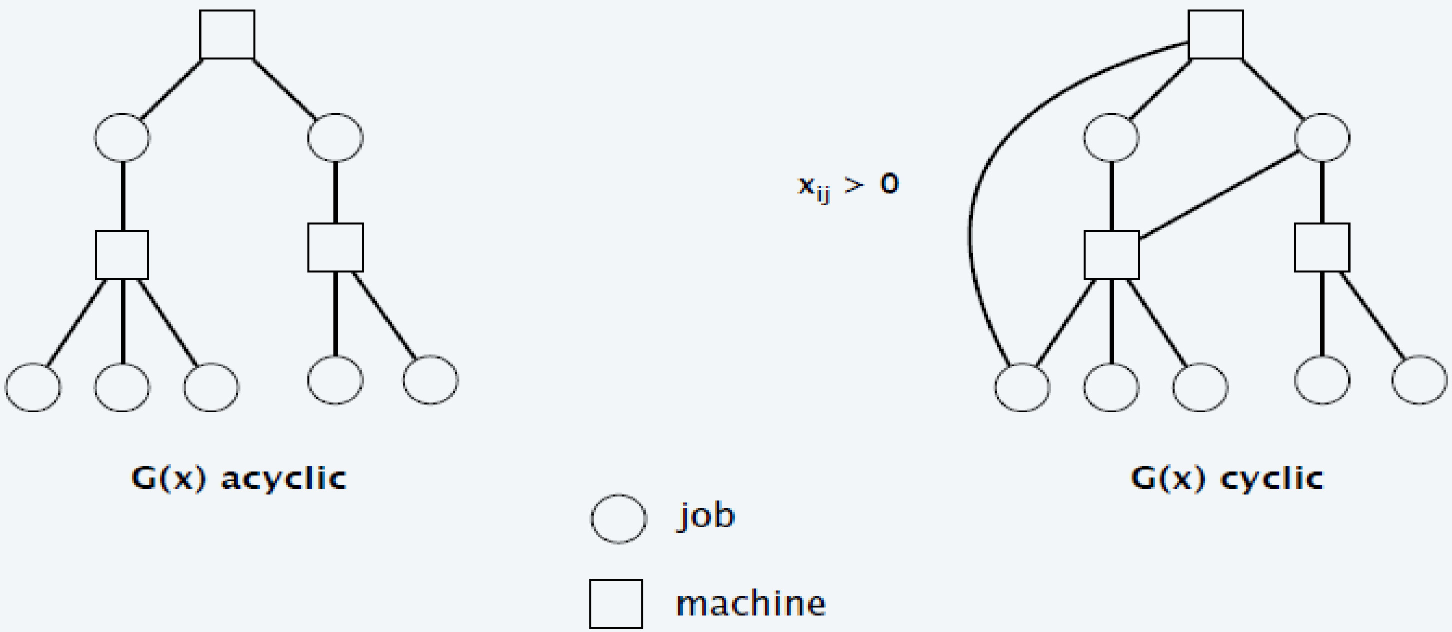
Pf. LP has fewer constraints than ILP formulation. ▪

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)



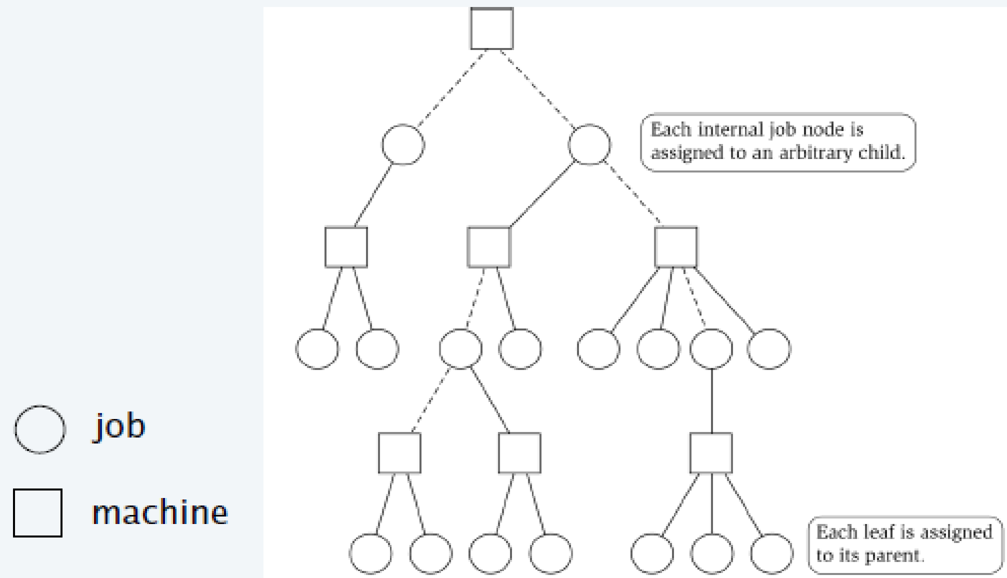
Generalized load balancing: rounding

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

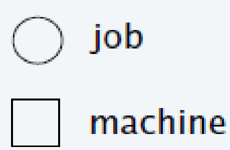
Lemma 5. If job j is a leaf node and machine $i = parent(j)$, then $x_{ij} = t_j$.

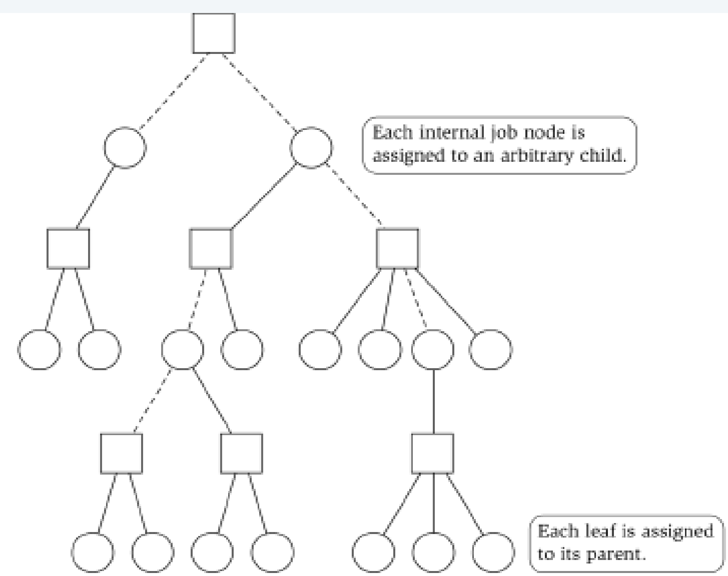
Pf.

- 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:

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

  - parent:

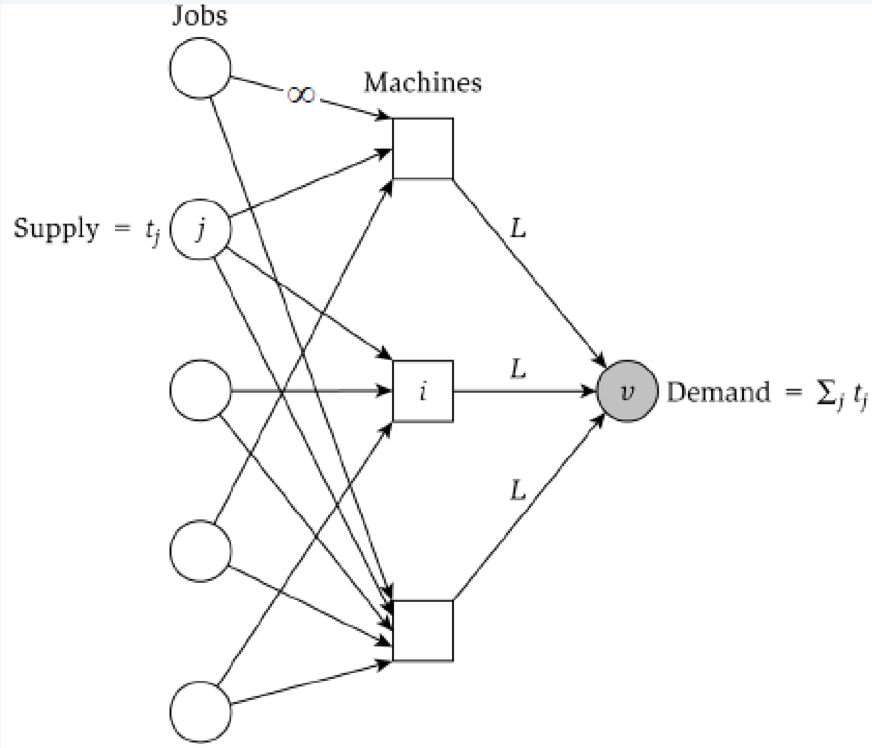
$$t_{\text{parent}(i)} \overset{\text{Lemma 1}}{\le} L^*$$

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

Generalized load balancing: flow formulation

Flow formulation of LP.

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



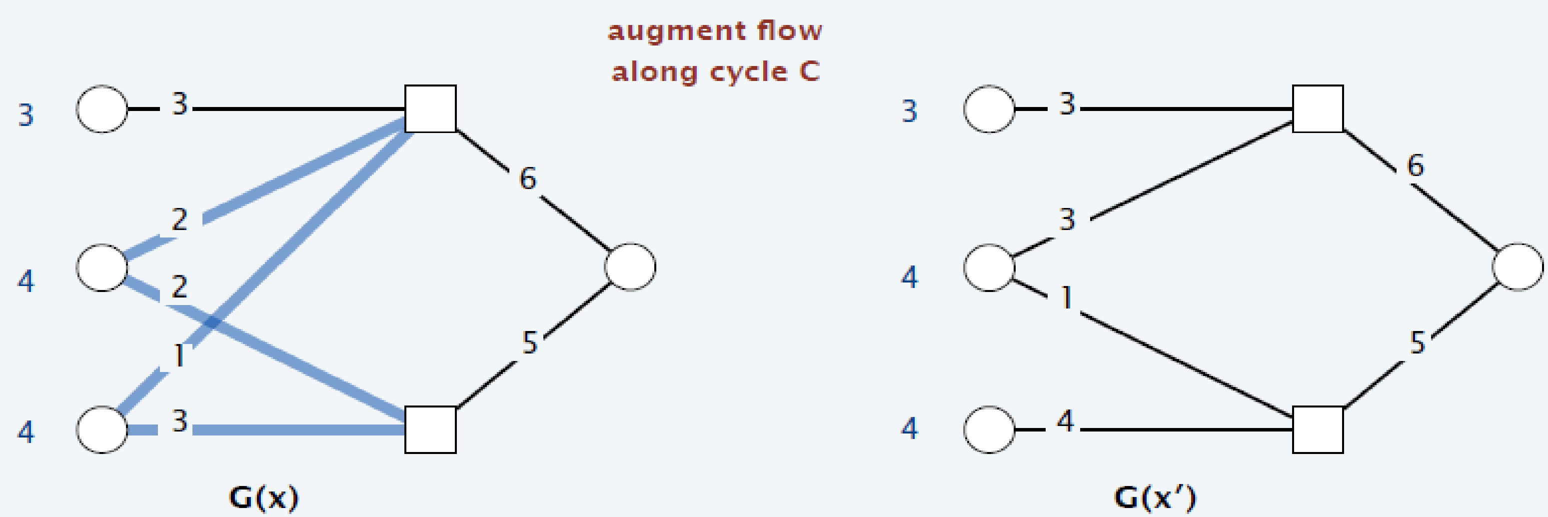
Observation. Solution to feasible flow problem with value L are in 1-to-1 correspondence with LP solutions of value L.

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** $\neq$ **NP**, then no no ρ-approximation exists for any $\rho < 3/2$.