Approximation Algorithms

Princeton University • COS 423 • Theory of Algorithms • Spring 2001 • Kevin Wayne

Coping With NP-Hardness

Suppose you need to solve NP-hard problem X.

- Theory says you aren't likely to find a polynomial algorithm.
- Should you just give up?
  - Probably yes, if you're goal is really to find a polynomial algorithm.
  - Probably no, if you're job depends on it.

Coping With NP-Hardness

Brute-force algorithms.

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

Heuristics.

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

Approximation algorithms.

- Guaranteed to run in polynomial time.
- Guaranteed to find "high quality" solution, say within 1% of optimum.
- Obstacle: need to prove a solution's value is close to optimum, without even knowing what optimum value is!

Approximation Algorithms and Schemes

ρ-approximation algorithm.

- An algorithm A for problem P that runs in polynomial time.
- For every problem instance, A outputs a feasible solution within ratio ρ of true optimum for that instance.

Polynomial-time approximation scheme (PTAS).

- A family of approximation algorithms $\{A_\varepsilon : \varepsilon > 0\}$ for a problem P.
- A_ε is a $(1 + \varepsilon)$ - approximation algorithm for P.
- A_ε is runs in time polynomial in input size for a fixed ε.

Fully polynomial-time approximation scheme (FPTAS).

- PTAS where A_ε is runs in time polynomial in input size and $1 / \varepsilon$.

Approximation Algorithms and Schemes

Types of approximation algorithms.

- Fully polynomial-time approximation scheme.
- Constant factor.

Knapsack Problem

Knapsack problem.

- Given N objects and a "knapsack."
- Item i weighs $w_i > 0$ Newtons and has value $v_i > 0$.
- Knapsack can carry weight up to W Newtons.
- Goal: fill knapsack so as to maximize total value.

v_i / w_i ↓

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

W = 11

Greedy = 35: { 5, 2, 1 }

OPT value = 40: { 3, 4 }

Knapsack is NP-Hard

KNAPSACK: Given a finite set X, nonnegative weights w_i , nonnegative values v_i , a weight limit W, and a desired 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 finite set X, nonnegative values u_i , and an integer t, is there a subset $S \subseteq X$ whose elements sum to t?

Claim. SUBSET-SUM $\leq_P$ KNAPSACK.

Proof: Given instance (X, t) of SUBSET-SUM, create KNAPSACK instance:

- $v_i = w_i = u_i$
- V = W = t

$$\sum_{i \in S} u_i \leq t$$
$$\sum_{i \in S} u_i \geq t$$

Knapsack: Dynamic Programming Solution 1

OPT(n, w) = max profit subset of items {1, . . . , n} with weight limit w.

- Case 1: OPT selects item n.
  - new weight limit = $w - w_n$
  - OPT selects best of {1, 2, . . . , n – 1} using this new weight limit
- Case 2: OPT does not select item n.
  - OPT selects best of {1, 2, . . . , n – 1} using weight limit w

$$OPT(n,w) = \begin{cases} 0 & \text{if } n = 0 \\ OPT(n-1,w) & \text{if } w_n > w \\ \max\{OPT(n-1,w),\ \ v_n + OPT(n-1,w-w_n)\} & \text{otherwise} \end{cases}$$

Directly leads to O(N W) time algorithm.

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

Knapsack: Dynamic Programming Solution 2

OPT(n, v) = min knapsack weight that yields value exactly v using subset of items {1, . . . , n}.

- Case 1: OPT selects item n.
  - new value needed = $v - v_n$
  - OPT selects best of {1, 2, . . . , n – 1} using new value
- Case 2: OPT does not select item n.
  - OPT selects best of {1, 2, . . . , n – 1} that achieves value v

$$OPT(n,v) = \begin{cases} 0 & \text{if } n = 0 \\ OPT(n-1, v) & \text{if } v_n > v \\ \min\{OPT(n-1, v),\ \ w_n + OPT(n-1, v-v_n)\} & \text{otherwise} \end{cases}$$

Directly leads to O(N V *) time algorithm.

- V* = optimal value.
- Not polynomial in input size!

Knapsack: Bottom-Up

Bottom-Up Knapsack

```
INPUT: N, W, w1,…,wN, v1,…,vN

ARRAY: OPT[0..N, 0..V*]

FOR v = 0 to V
   OPT[0, v] = 0

FOR n = 1 to N
   FOR w = 1 to W
      IF (vn > v)
         OPT[n, v] = OPT[n-1, v]
      ELSE
         OPT[n, v] = min {OPT[n-1, v], wn + OPT[n-1, v-vn ]}

v* = max {v : OPT[N, v] ≤ W}
RETURN OPT[N, v*]
```


Knapsack: FPTAS

Intuition for approximation algorithm.

- Round all values down to lie in smaller range.
- Run O(N V*) dynamic programming algorithm on rounded instance.
- Return optimal items in rounded instance.

Item	Value	Weight
1	134,221	1
2	656,342	2
3	1,810,013	5
4	22,217,800	6
5	28,343,199	7

W = 11

Original Instance

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	222	6
5	283	7

W = 11

Rounded Instance

Knapsack: FPTAS

Knapsack FPTAS.

- Round all values: $\overline{v}_n = \left\lfloor \frac{v_n}{\theta} \right\rfloor$
  - V = largest value in original instance
  - ε = precision parameter
  - θ = scaling factor = ε V / N
- Bound on optimal value V *:

$$V \le V^* \le N V$$

assume $w_n \le W$ for all n

Running Time

$$O(N\,\overline{V}^*) \in O(N(N\overline{V})) \in O(N^2 (V/\theta)) \in O(N^3 \tfrac{1}{\varepsilon})$$

$\overline{V}$ = largest value in rounded instance

$\overline{V}^*$ = optimal value in rounded instance

Knapsack: FPTAS

Knapsack FPTAS.

- Round all values: $\overline{v_n} = \left\lfloor \frac{v_n}{\theta} \right\rfloor$
  - V = largest value in original instance
  - ε = precision parameter
  - θ = scaling factor = ε V / N
- Bound on optimal value V *:

$$V \le V^* \le N V$$

S^* = opt set of items in original instance
$\overline{S^*}$ = opt set of items in rounded instance

Proof of Correctness

$$\begin{aligned}
\sum_{n \in S^*} v_n &\ge \sum_{n \in S^*} \theta \overline{v_n} \\
&\ge \sum_{n \in \overline{S^*}} \theta \overline{v_n} \\
&\ge \sum_{n \in \overline{S^*}} (v_n - \theta) \\
&\ge \sum_{n \in \overline{S^*}} v_n - \theta N \\
&= V^* - (\varepsilon V / N) N \\
&\ge (1-\varepsilon) V^*
\end{aligned}$$

Knapsack: State of the Art

This lecture.

- "Rounding and scaling" method finds a solution within a (1 - ε) factor of optimum for any ε > 0.
- Takes $O(N^3 / \varepsilon)$ time and space.

Ibarra-Kim (1975), Lawler (1979).

- Faster FPTAS: $O(N \log (1 / \varepsilon) + 1 / \varepsilon^4)$ time.
- Idea: group items by value into "large" and "small" classes.
  - run dynamic programming algorithm only on large items
  - insert small items according to ratio v_n / w_n
  - clever analysis

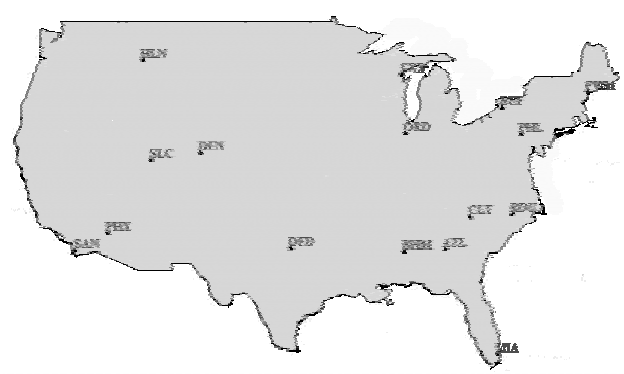
Approximation Algorithms and Schemes

Types of approximation algorithms.

- Fully polynomial-time approximation scheme.
- Constant factor.

Traveling Salesperson Problem

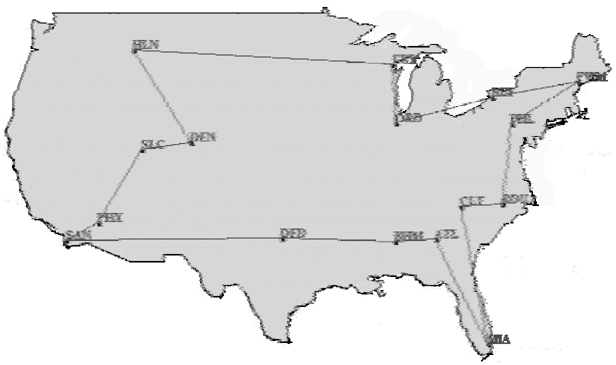
TSP: Given a graph G = (V, E), nonnegative edge weights c(e), and an integer C, is there a Hamiltonian cycle whose total cost is at most C?



Is there a tour of length at most 1570?

Traveling Salesperson Problem

TSP: Given a graph G = (V, E), nonnegative edge weights c(e), and an integer C, is there a Hamiltonian cycle whose total cost is at most C?



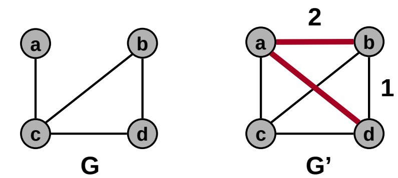
Is there a tour of length at most 1570? Yes, red tour = 1565.

Hamiltonian Cycle Reduces to TSP

HAM-CYCLE: given an undirected graph G = (V, E), does there exists a simple cycle C that contains every vertex in V.

TSP: Given a complete (undirected) graph G, integer edge weights $c(e) \geq 0$, and an integer C, is there a Hamiltonian cycle whose total cost is at most C?

Claim. HAM-CYCLE is NP-complete.



Proof. (HAM-CYCLE transforms to TSP)

- Given G = (V, E), we want to decide if it is Hamiltonian.
- Create instance of TSP with G' = complete graph.
- Set c(e) = 1 if $e \in E$, and c(e) = 2 if $e \notin E$, and choose C = |V|.
- Γ Hamiltonian cycle in G $\Leftrightarrow$ Γ has cost exactly |V| in G'.
  Γ not Hamiltonian in G $\Leftrightarrow$ Γ has cost at least |V| + 1 in G'.

TSP

TSP-OPT: Given a complete (undirected) graph G = (V, E) with integer edge weights $c(e) \geq 0$, find a Hamiltonian cycle of minimum cost?

Claim. If $P \neq NP$, there is no ρ-approximation for TSP for any $\rho \geq 1$.

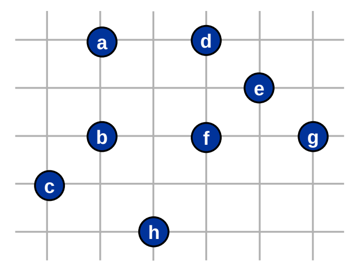
Proof (by contradiction).

- Suppose A is ρ-approximation algorithm for TSP.
- We show how to solve instance G of HAM-CYCLE.
- Create instance of TSP with G' = complete graph.
- Let C = |V|, c(e) = 1 if $e \in E$, and c(e) = ρ |V | + 1 if $e \notin E$.
- Γ Hamiltonian cycle in G $\Leftrightarrow$ Γ has cost exactly |V| in G'
  Γ not Hamiltonian in G $\Leftrightarrow$ Γ has cost more than ρ |V| in G'
- Gap $\Rightarrow$ If G has Hamiltonian cycle, then A must return it.

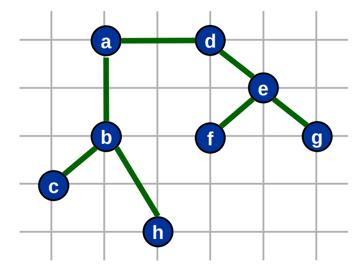
TSP Heuristic

APPROX-TSP(G, c)

- Find a minimum spanning tree T for (G, c).



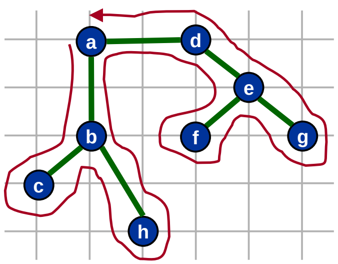
Input
(assume Euclidean distances)

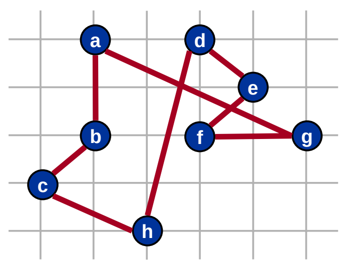


MST

TSP Heuristic

APPROX-TSP(G, c)
- Find a minimum spanning tree T for (G, c).
- **W ← ordered list of vertices in preorder walk of T.**
- **H ← cycle that visits the vertices in the order L.**





Preorder Traversal Full Walk W

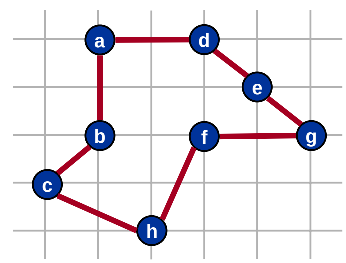
a b c b h b a d e f e g e d a

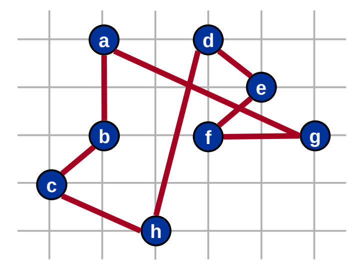
Hamiltonian Cycle H

a b c h d e f g a

TSP Heuristic

APPROX-TSP(G, c)
- Find a minimum spanning tree T for (G, c).
- W ← ordered list of vertices in preorder walk of T.
- H ← cycle that visits the vertices in the order L.





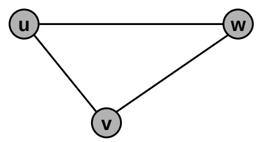
An Optimal Tour: 14.715

Hamiltonian Cycle H: 19.074

(assuming Euclidean distances)

TSP With Triangle Inequality

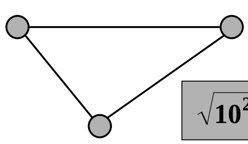
Δ-TSP: TSP where costs satisfy Δ-inequality:
- **For all u, v, and w: $c(u,w) \le c(u,v) + c(v,w)$.**



Claim. Δ-TSP is NP-complete.
Proof. Transformation from HAM-CYCLE satisfies Δ-inequality.

Ex. Euclidean points in the plane.
- **Euclidean TSP is NP-hard, but not known to be in NP.**

(-10, 5) **(5, 9)**



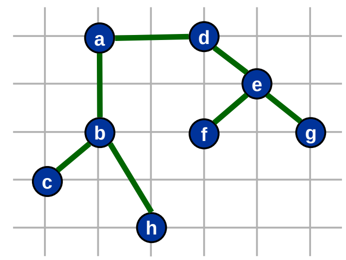
(0,0)

$$\sqrt{10^2+5^2}+\sqrt{5^2+9^2}+\sqrt{15^2+4^2}=37.000\ldots$$

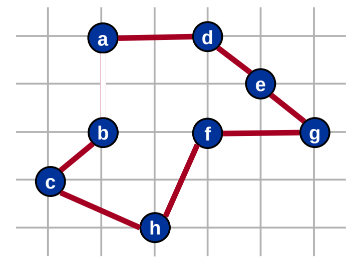
- **PTAS for Euclidean TSP. (Arora 1996, Mitchell 1996)**

TSP With Triangle Inequality

Theorem. APPROX-TSP is a 2-approximation algorithm for Δ-TSP.
Proof. Let H* denote an optimal tour. Need to show $c(H) \le 2c(H^*)$.
- **$c(T) \le c(H^*)$ since we obtain spanning tree by deleting any edge from optimal tour.**



MST T



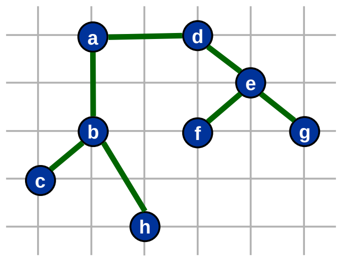
An Optimal Tour

TSP With Triangle Inequality

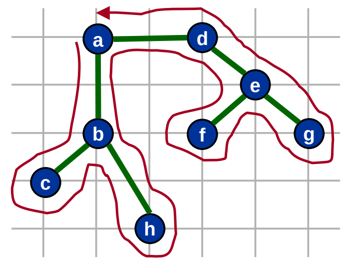
Theorem. APPROX-TSP is a 2-approximation algorithm for Δ-TSP.

Proof. Let H* denote an optimal tour. Need to show c(H) ≤ 2c(H*).

- c(T) ≤ c(H*) since we obtain spanning tree by deleting any edge from optimal tour.
- c(W) = 2c(T) since every edge visited exactly twice.



MST T



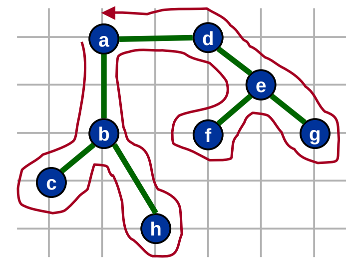
Walk W
a b c b h b a d e f e g e d a

TSP With Triangle Inequality

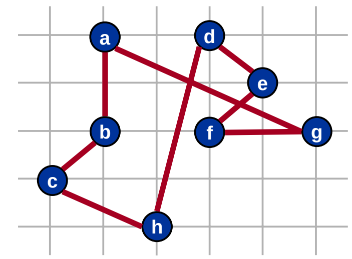
Theorem. APPROX-TSP is a 2-approximation algorithm for Δ-TSP.

Proof. Let H* denote an optimal tour. Need to show c(H) ≤ 2c(H*).

- c(T) ≤ c(H*) since we obtain spanning tree by deleting any edge from optimal tour.
- c(W) = 2c(T) since every edge visited exactly twice.
- c(H) ≤ c(W) because of Δ-inequality.



Walk W
a b c b h b a d e f e g e d a



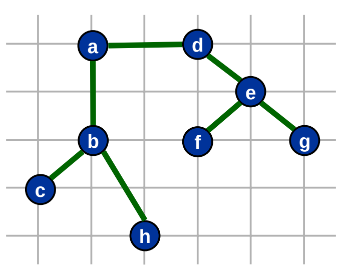
Hamiltonian Cycle H
a b c h d e f g a

TSP: Christofides Algorithm

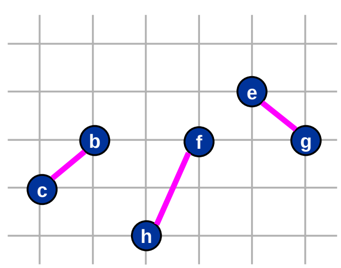
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

CHRISTOFIDES(G, c)

- Find a minimum spanning tree T for (G, c).
- M ← min cost perfect matching of odd degree nodes in T.



MST T



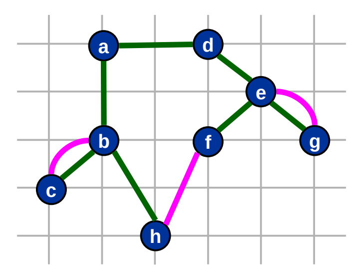
Matching M

TSP: Christofides Algorithm

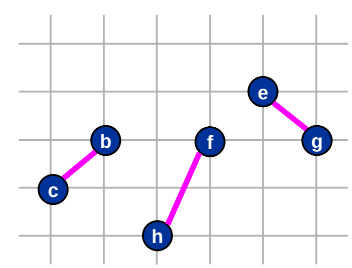
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

CHRISTOFIDES(G, c)

- Find a minimum spanning tree T for (G, c).
- M ← min cost perfect matching of odd degree nodes in T.
- G' ← union of spanning tree and matching edges.



G' = MST + Matching



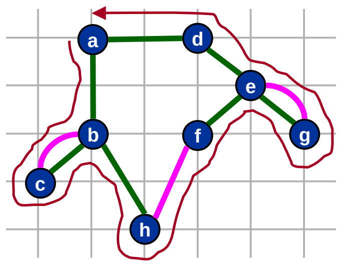
Matching M

TSP: Christofides Algorithm

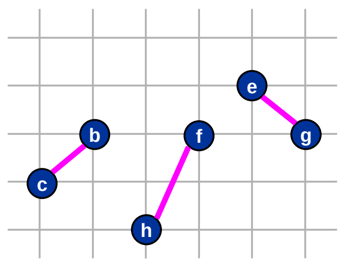
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

CHRISTOFIDES(G, c)

- Find a minimum spanning tree T for (G, c).
- M ← min cost perfect matching of odd degree nodes in T.
- G' ← union of spanning tree and matching edges.
- E ← Eulerian tour in G'.



E = Eulerian tour in G'



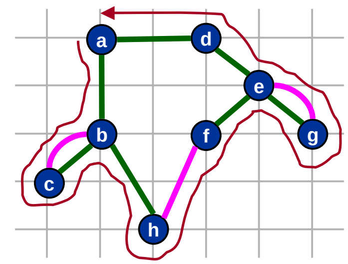
Matching M

TSP: Christofides Algorithm

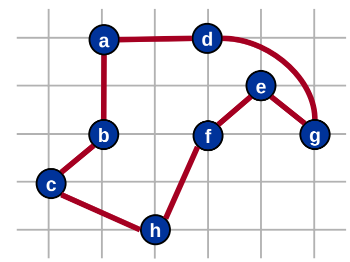
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

CHRISTOFIDES(G, c)

- Find a minimum spanning tree T for (G, c).
- M ← min cost perfect matching of odd degree nodes in T.
- G' ← union of spanning tree and matching edges.
- E ← Eulerian tour in G'.
- H ← short-cut version of Eulerian tour in E.



E = Eulerian tour in G'

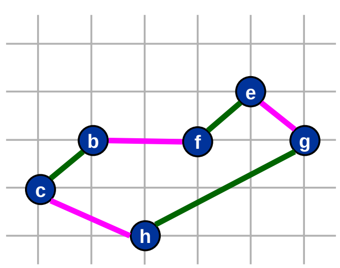


Hamiltonian Cycle H

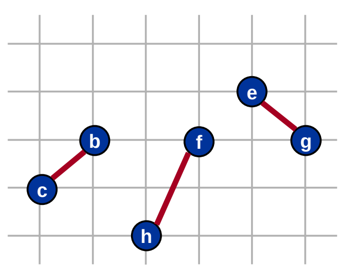
TSP: Christofides Algorithm

Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.
Proof. Let H* denote an optimal tour. Need to show c(H) ≤ 1.5 c(H*).

- c(T) ≤ c(H*) as before.
- c(M) ≤ ½ c(Γ*) ≤ ½ c(H*).
  - second inequality follows from Δ-inequality
  - even number of odd degree nodes
  - Hamiltonian cycle on even # nodes comprised of two matchings



Optimal Tour Γ* on Odd Nodes

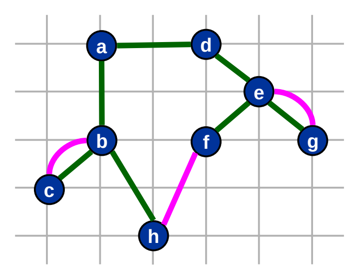


Matching M

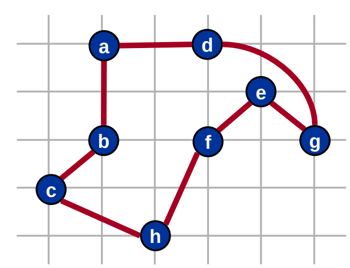
TSP: Christofides Algorithm

Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.
Proof. Let H* denote an optimal tour. Need to show c(H) ≤ 1.5 c(H*).

- c(T) ≤ c(H*) as before.
- c(M) ≤ ½ c(Γ*) ≤ ½ c(H*).
- Union of MST and and matching edges is Eulerian.
  - every node has even degree
- Can shortcut to produce H and c(H) ≤ c(M) + c(T).



MST + Matching



Hamiltonian Cycle H

Load Balancing

Load balancing input.

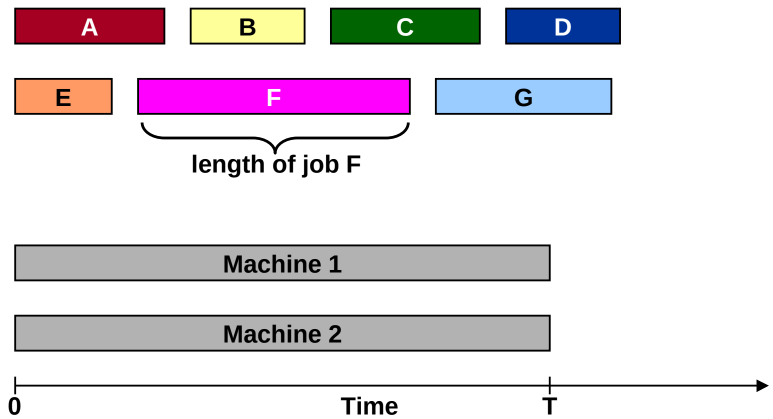
- m identical machines.
- n jobs, job j has processing time p_j.

Goal: assign each job to a machine to minimize makespan.

- If subset of jobs S_i assigned to machine i, then i works for a total time of $T_i = \sum_{j \in S_i} p_j$.
- Minimize maximum T_i.

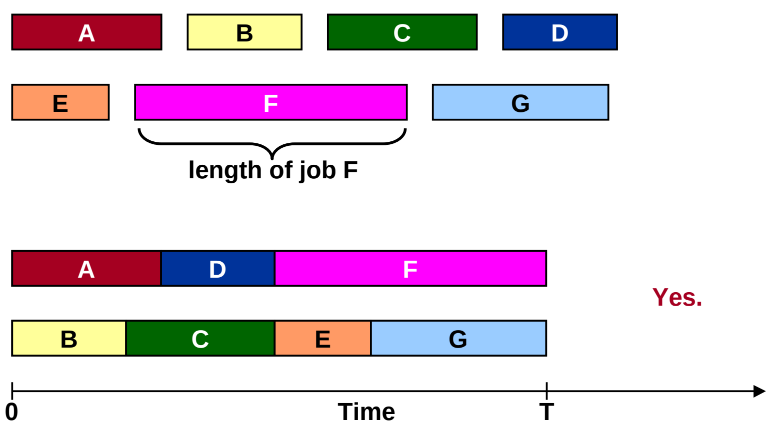
Load Balancing on 2 Machines

2-LOAD-BALANCE: Given a set of jobs J of varying length $p_j \geq 0$, and an integer T, can the jobs be processed on 2 identical parallel machines so that they all finish by time T.



Load Balancing on 2 Machines

2-LOAD-BALANCE: Given a set of jobs J of varying length $p_j \geq 0$, and an integer T, can the jobs be processed on 2 identical parallel machines so that they all finish by time T.



Load Balancing is NP-Hard

PARTITION: Given a set X of nonnegative integers, is there a subset $S \subseteq X$ such that $\sum_{a \in S} a = \sum_{a \in X \setminus S} a$.

2-LOAD-BALANCE: Given a set of jobs J of varying length p_j, and an integer T, can the jobs be processed on 2 identical parallel machines so that they all finish by time T.

Claim. PARTITION $\leq_P$ 2-LOAD-BALANCE.

Proof. Let X be an instance of PARTITION.

- For each integer $x \in X$, include a job j of length $p_j = x$.
- Set $T = \frac{1}{2} \sum_{a \in X} a$.

Conclusion: load balancing optimization problem is NP-hard.

Load Balancing

Greedy algorithm.

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

LIST-SCHEDULING (m, n, p_1, . . . , p_n)

```
FOR i = 1 to m
   Ti ← 0, Si ← φ

FOR j = 1 to n
   i = argmink Tk        ← machine with smallest load
   Si ← Si ∪ {j}         ← assign job j to machine i
   Ti ← Ti + pj
```

- Note: this is an "on-line" algorithm.

Load Balancing

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 T*.

Lemma 1. The optimal makespan is at least $T^* \geq \frac{1}{m}\sum_j p_j$.

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

Lemma 2. The optimal makespan is at least $T^* \geq \max_j p_j$.

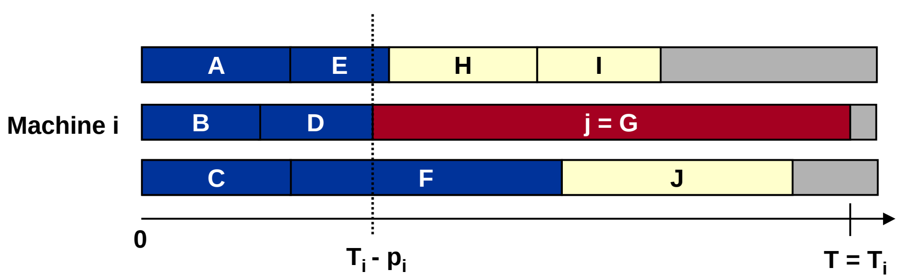
- Some machine must process the most time-consuming job.

Load Balancing

Lemma 1. The optimal makespan is at least $T^* \geq \frac{1}{m}\sum_j p_j$.
Lemma 2. The optimal makespan is at least $T^* \geq \max_j p_j$.

Theorem. Greedy algorithm is a 2-approximation.
Proof. Consider bottleneck machine i that works for T units of time.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i has smallest load. It's load before assignment is $T_i - p_j \Rightarrow T_i - p_j \leq T_k$ for all $1 \leq k \leq m$.



Load Balancing

Lemma 1. The optimal makespan is at least $T^* \geq \frac{1}{m}\sum_j p_j$.
Lemma 2. The optimal makespan is at least $T^* \geq \max_j p_j$.

Theorem. Greedy algorithm is a 2-approximation.
Proof. Consider bottleneck machine i that works for T units of time.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i has smallest load. It's load before assignment is $T_i - p_j \Rightarrow T_i - p_j \leq T_k$ for all $1 \leq k \leq n$.
- Sum inequalities over all k and divide by m, and then apply L1.

$$\begin{aligned} T_i - p_j &\leq \frac{1}{m}\sum_k T_k \\ &= \frac{1}{m}\sum_k p_k \\ &\leq T^* \end{aligned}$$

- Finish off using L2.

$$\begin{aligned} T_i &= (T_i - p_j) + p_j \\ &\leq T^* + T^* \\ &= 2T^* \end{aligned}$$

Load Balancing

Is our analysis tight?

- Essentially yes.
- We give instance where solution is almost factor of 2 from optimal.
  - m machines, m(m-1) jobs with of length 1, 1 job of length m
  - 10 machines, 90 jobs of length 1, 1 job of length 10

1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	Machine 2
3	13	23	33	43	53	63	73	83	Machine 3
4	14	24	34	44	54	64	74	84	Machine 4
5	15	25	35	45	55	65	75	85	Machine 5
6	16	26	36	46	56	66	76	86	Machine 6
7	17	27	37	47	57	67	77	87	Machine 7
8	18	28	38	48	58	68	78	88	Machine 8
9	19	29	39	49	59	69	79	89	Machine 9
10	20	30	40	50	60	70	80	90	Machine 10

List Schedule makespan = 19

Load Balancing

Is our analysis tight?

- Essentially yes.
- We give instance where solution is almost factor of 2 from optimal.
  - m machines, m(m-1) jobs with of length 1, 1 job of length m
  - 10 machines, 90 jobs of length 1, 1 job of length 10

1	11	21	31	41	51	61	71	81	10	Machine 1
2	12	22	32	42	52	62	72	82	20	Machine 2
3	13	23	33	43	53	63	73	83	30	Machine 3
4	14	24	34	44	54	64	74	84	40	Machine 4
5	15	25	35	45	55	65	75	85	50	Machine 5
6	16	26	36	46	56	66	76	86	60	Machine 6
7	17	27	37	47	57	67	77	87	70	Machine 7
8	18	28	38	48	58	68	78	88	80	Machine 8
9	19	29	39	49	59	69	79	89	90	Machine 9
91										Machine 10

Optimal makespan = 10

Load Balancing: State of the Art

What's known.

- 2-approximation algorithm.
- 3/2-approximation algorithm: homework.
- 4/3-approximation algorithm: extra credit.
- PTAS.