Advanced Core in Algorithm Design #11
算法設計要論 第11回

Yasushi Kawase
河瀬 康志

Dec. 20th, 2022

last update: 12:21pm, December 20, 2022

Schedule

Lec. #	Date	Topics
1	10/4	Introduction, Stable matching
2	10/11	Basics of Algorithm Analysis, Greedy Algorithms (1/2)
3	10/18	Greedy Algorithms (2/2)
4	10/25	Divide and Conquer (1/2)
5	11/1	Divide and Conquer (2/2)
6	11/8	Dynamic Programming (1/2)
7	11/15	Dynamic Programming (2/2)
—	11/22	Thursday Classes
8	11/29	Network Flow (1/2)
9	12/6	Network Flow (2/2)
10	12/13	NP and Computational Intractability
11	12/20	Approximation Algorithms (1/2)
12	12/27	Approximation Algorithms (2/2)
13	1/10	Randomized Algorithms

Outline

Coping with NP-hardness

What can we do for an NP-hard problem

- Exponential time algorithms
- Heuristic algorithms
- Approximation algorithms
- FPT (fixed parameter tractable) algorithms
- Parallelism
- Randomization
- Quantum computation

Approximation algorithm

Definition

For a maximization problem "$\max f(x)$ s.t. $x \in X$",
a solution $x^* \in X$ is an α-approximation solution if $f(x^*) \geq \alpha \cdot \mathrm{OPT}$ ($0 \leq \alpha \leq 1$)

Definition

For a minimization problem "$\min f(x)$ s.t. $x \in X$",
a solution $x^* \in X$ is an α-approximation solution if $f(x^*) \leq \alpha \cdot \mathrm{OPT}$ ($\alpha \geq 1$)

Definition

An α-approximation algorithm is a polynomial-time algorithm that finds an α-approximation solution for any instance

Outline

Load balancing

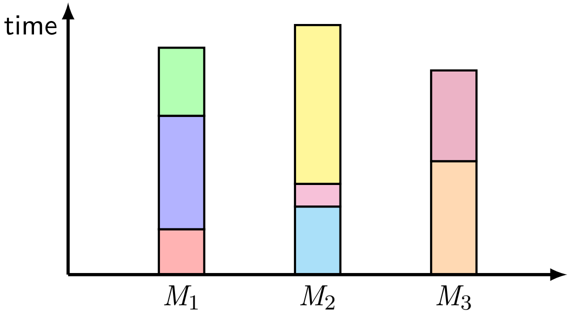
Problem

- Input: m identical machines, n jobs; job j has processing time t_j
- Goal: find an assignment that minimizes makespan

partition $(A(1), \ldots, A(m))$

$\max_i \sum_{j \in A(i)} t_j$

Example



Hardness of Load balancing

Theorem

Load balancing problem is NP-hard even if $m = 2$

Proof: PARTITION $\leq_{\mathrm{P}}$ Load-Balance

PARTITION Problem

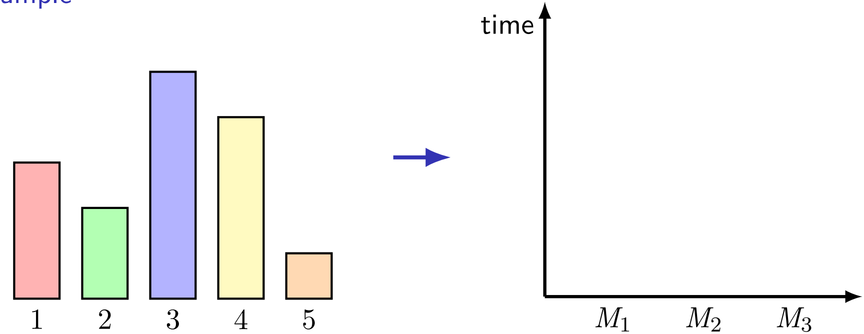
Given $a_1, a_2, \ldots, a_n \in \mathbb{Z}_+$, is there $I \subseteq [n]$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

List scheduling algorithm

Algorithm

for $j \leftarrow 1, 2, \ldots, n$ **do**
    assign job j to a machine i that has smallest load;

Example

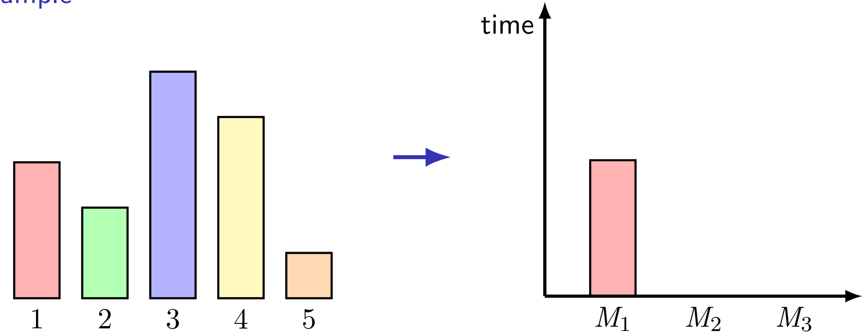


List scheduling algorithm

Algorithm

for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

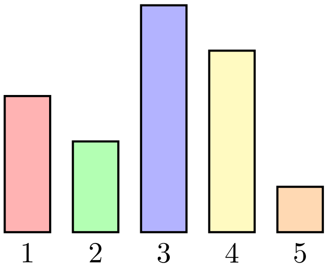


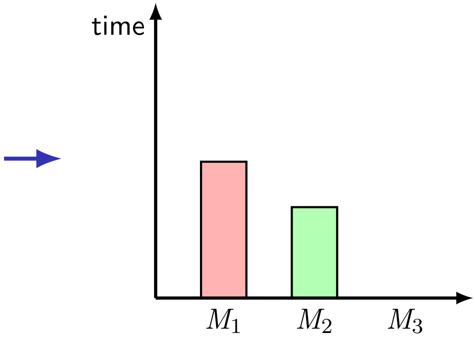
List scheduling algorithm

Algorithm

for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example



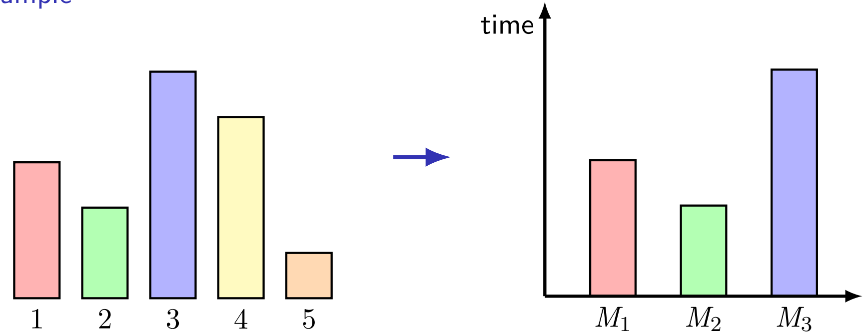


List scheduling algorithm

Algorithm

for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

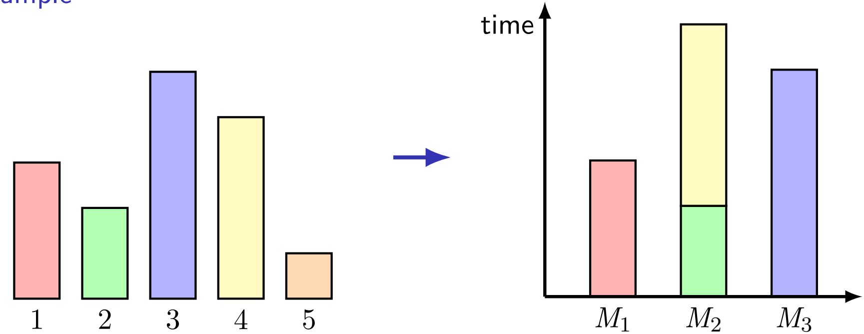


List scheduling algorithm

Algorithm

for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

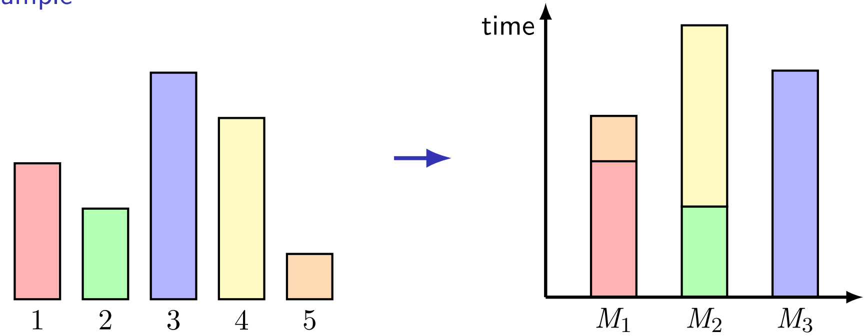


List scheduling algorithm

Algorithm

for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example



Approximation ratio of list scheduling

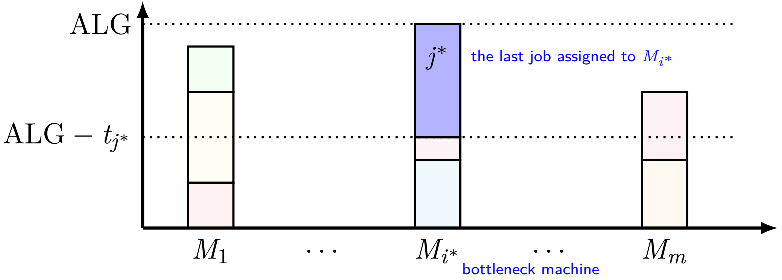
Theorem

List scheduling algorithm is a $(2-1/m)$-approximation

Proof

- $\mathrm{OPT} \geq \frac{1}{m}\sum_{j=1}^{n} t_j$ and $\mathrm{OPT} \geq \max_{j=1}^{n} t_j \geq t_{j^*}$
- $\sum_{j\in A(i^*)} t_j = \mathrm{ALG}$ and $\sum_{j\in A(i)} t_j \geq \mathrm{ALG} - t_{j^*}$ for all M_i
- $\sum_{i=1}^{m}\sum_{j\in A(i)} t_j \geq m\mathrm{ALG} - (m-1)t_{j^*} \longrightarrow \mathrm{ALG} \leq (2-\frac{1}{m})\mathrm{OPT}$

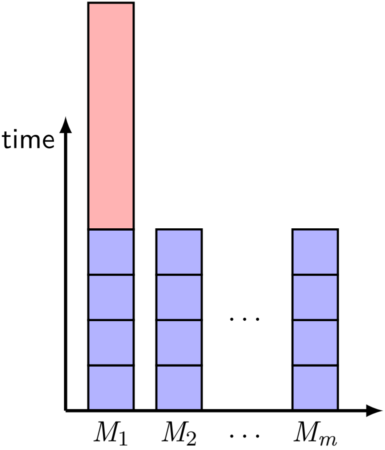
$m\mathrm{OPT} \geq$ $\leq \mathrm{OPT}$



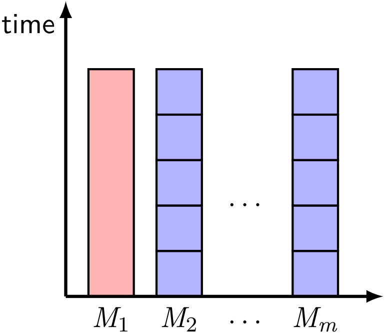
Worst case of list scheduling

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

→ list scheduling algorithm outputs a $(2-1/m)$-approximation solution



$\mathrm{ALG} = (m-1) + m$



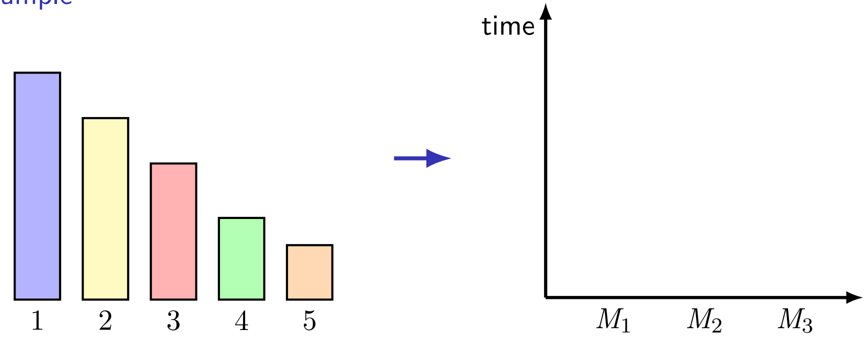
$\mathrm{OPT} = m$

Improved algorithm

Longest Processing Time (LPT) algorithm

Sort jobs and relabel so that $t_1 \geq t_2 \geq \cdots \geq t_n$;
for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

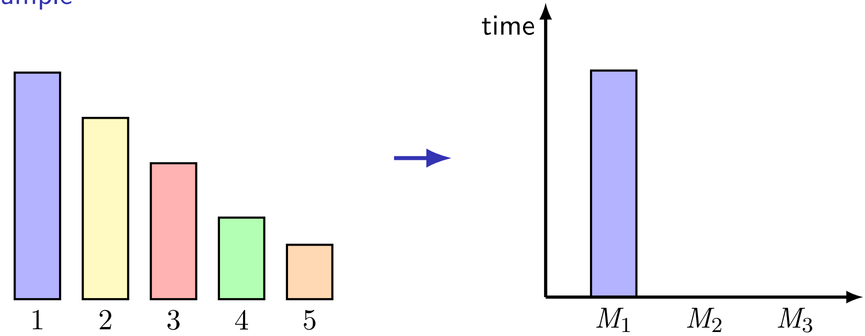


Improved algorithm

Longest Processing Time (LPT) algorithm

Sort jobs and relabel so that $t_1 \geq t_2 \geq \cdots \geq t_n$;
for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

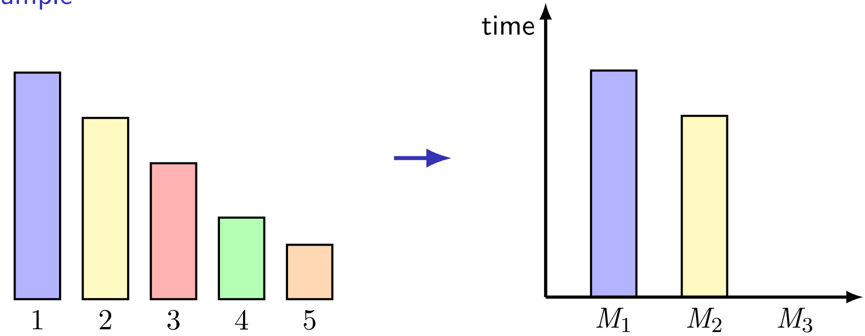


Improved algorithm

Longest Processing Time (LPT) algorithm

Sort jobs and relabel so that $t_1 \geq t_2 \geq \cdots \geq t_n$;
for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

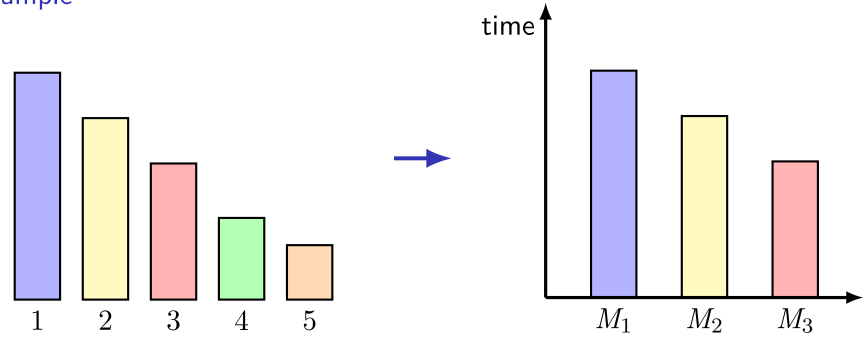


Improved algorithm

Longest Processing Time (LPT) algorithm

Sort jobs and relabel so that $t_1 \geq t_2 \geq \cdots \geq t_n$;
for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

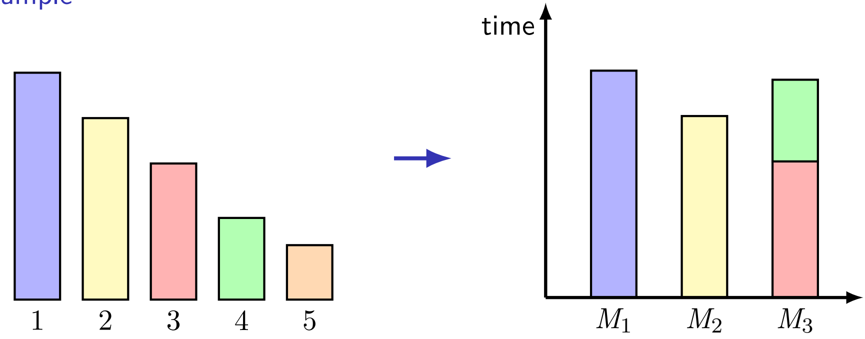


Improved algorithm

Longest Processing Time (LPT) algorithm

Sort jobs and relabel so that $t_1 \geq t_2 \geq \cdots \geq t_n$;
for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example

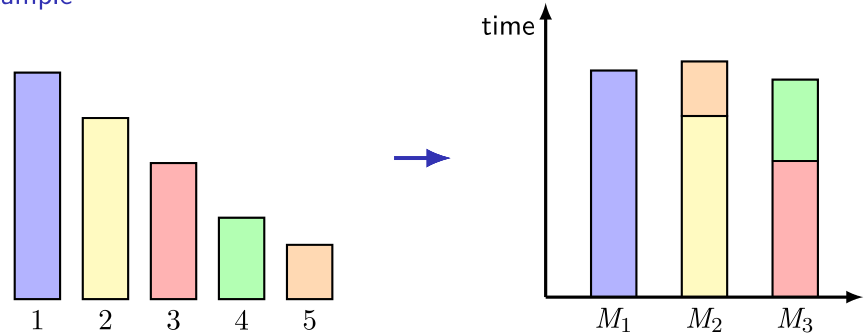


Improved algorithm

Longest Processing Time (LPT) algorithm

Sort jobs and relabel so that $t_1 \geq t_2 \geq \cdots \geq t_n$;
for $j \leftarrow 1, 2, \ldots, n$ **do**
  assign job j to a machine i that has smallest load;

Example



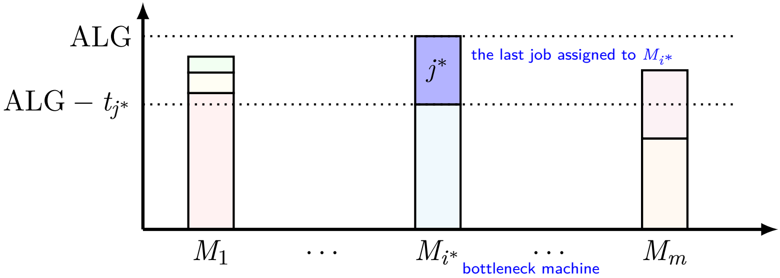
Approximation ratio of LPT

Theorem

LPT algorithm is a 1.5-approximation

Proof W.L.O.G. $|A(i^*)| \geq 2$ otherwise $\mathrm{OPT} = \mathrm{ALG} = t_1$

- $\mathrm{OPT} \geq \frac{1}{m}\sum_{j=1}^{n} t_j = \frac{1}{m}\sum_{i=1}^{m}\sum_{j\in A(i)} t_j \geq \mathrm{ALG} - t_{j^*}$
- $\mathrm{OPT} \geq t_m + t_{m+1} \geq 2t_{j^*}$ because $\exists$machine gets two jobs from $1, 2, \ldots, m+1$
- $\mathrm{ALG} = (\mathrm{ALG} - t_{j^*}) + t_{j^*} \leq \mathrm{OPT} + \frac{1}{2}\mathrm{OPT} = 1.5 \cdot \mathrm{OPT}$



Approximation ratio of LPT (improved)

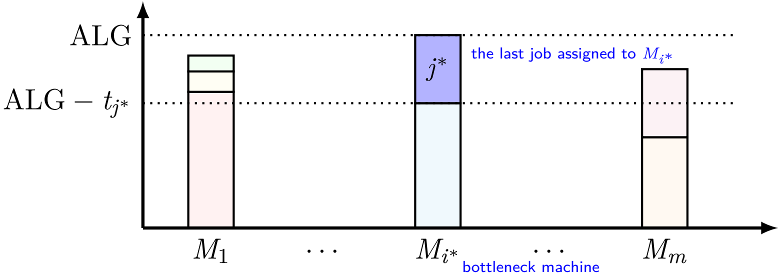
Theorem

LPT algorithm is a $4/3$-approximation

Proof W.L.O.G. $|A(i^*)| \geq 2$ otherwise $\mathrm{OPT} = \mathrm{ALG} = t_1$

- If $t_{j^*} \leq \mathrm{OPT}/3$, then $\mathrm{ALG} = (\mathrm{ALG} - t_{j^*}) + t_{j^*} \leq \frac{4}{3} \cdot \mathrm{OPT}$
- If $t_{j^*} > \mathrm{OPT}$, opt. sol assigns ≤ 2 jobs from $1, 2, \ldots, j^*$ on every machine

→ $\mathrm{ALG} = t_{2m+1-j^*} + t_{j^*} = \mathrm{OPT}$



Outline

Weighted Vertex Cover Problem

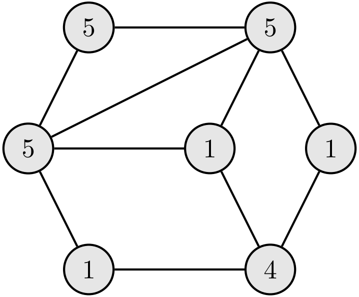
Problem

- Input: Undirected graph $G = (V, E)$ with cost $c\colon V \to \mathbb{R}_+$
- Goal: find a minimum weight vertex cover

$S \subseteq V$ is a vertex cover if each edge is incident to at least one vertex in S

This problem is **NP**-hard even when $c_v = 1$ $(\forall v \in V)$

Example



Weighted Vertex Cover Problem

Problem

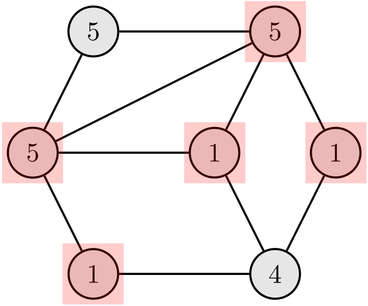
- Input: Undirected graph $G = (V, E)$ with cost $c\colon V \to \mathbb{R}_+$
- Goal: find a minimum weight vertex cover

$S \subseteq V$ is a vertex cover if each edge is incident to at least one vertex in S

This problem is **NP**-hard even when $c_v = 1$ $(\forall v \in V)$

Example

weight $= 13$



LP relaxation

(Integral) Vertex cover

$$\min \quad \sum_{v\in V} c_v x_v \quad \text{s.t.} \quad x_u + x_v \geq 1 \ (\forall\{u,v\}\in E),\ \ x_v \in \{0,1\} \ (\forall v \in V)$$

Relaxed vertex cover

$$\min \quad \sum_{v\in V} c_v x_v \quad \text{s.t.} \quad x_u + x_v \geq 1 \ (\forall\{u,v\}\in E),\ \ x_v \in [0,1] \ (\forall v \in V)$$

Observations

- $\mathrm{OPT}^{\mathsf{int}} \geq \mathrm{OPT}^{\mathsf{relax}}$
- Relaxed vertex cover can be solved in polynomial time
  (ellipsoid algorithm or interior point algorithm for LP)

LP rounding algorithm

Relaxed vertex cover

$$\min \quad \sum_{v \in V} c_v x_v \quad \text{s.t.} \quad x_u + x_v \geq 1 \ (\forall \{u, v\} \in E), \quad x_v \in [0, 1] \ (\forall v \in V)$$

Algorithm

1 Solve the relaxed vertex cover and let x^* be the optimal solution;
2 **Return** $S = \{v \in V \mid x_v^* \geq 1/2\}$;

Theorem

The LP rounding algorithm is a 2-approximation

- Feasibility: $\forall \{u, v\} \in E$, $x_u^* \geq 1/2$ or $x_v^* \geq 1/2$ ⟶ $\{u, v\}$ is covered
- Approx. ratio: $\sum_{v \in S} w_v \leq 2 \cdot \sum_{v \in V} c_v x_v^* = 2 \cdot \mathrm{OPT}^{\text{relax}} \leq 2 \cdot \mathrm{OPT}^{\text{int}}$

LP rounding algorithm

Relaxed vertex cover

$\min \quad \sum_{v\in V} c_v x_v$ s.t. $x_u + x_v \geq 1 \ (\forall\{u,v\} \in E)$, $x_v \in [0,1] \ (\forall v \in V)$

Algorithm

1 Solve the relaxed vertex cover and let x^* be the optimal solution;
2 **Return** $S = \{v \in V \mid x_v^* \geq 1/2\}$;

Theorem

The LP rounding algorithm is a 2-approximation

- Feasibility: $\forall\{u,v\} \in E$, $x_u^* \geq 1/2$ or $x_v^* \geq 1/2$ ⟶ $\{u,v\}$ is covered
- Approx. ratio: $\sum_{v\in S} w_v \leq 2 \cdot \sum_{v\in V} c_v x_v^* = 2 \cdot \mathrm{OPT}^{\text{relax}} \leq 2 \cdot \mathrm{OPT}^{\text{int}}$

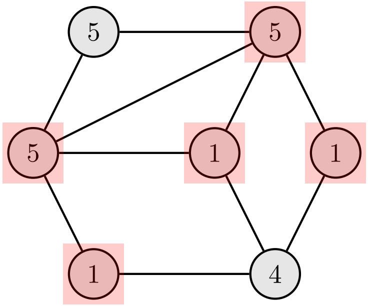
$\nexists(2-\epsilon)$-approx. alg. if the unique game conjecture is true [Khot and Regev 2008]

a complexity assumption stronger than $\mathbf{P} \neq \mathbf{NP}$

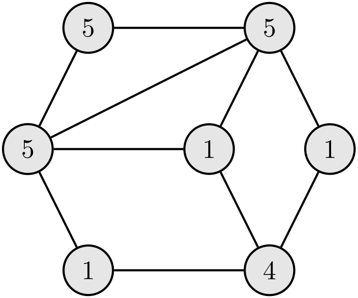
Quiz

Compute the output of the LP rounding algorithm

OPT weight $= 13$



LP rounding weight $=$???



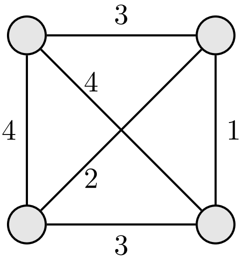
Outline

Traveling Salesman Problem

Problem

- Input: Complete undirected graph $G = (V, E)$ with distance $d\colon E \to \mathbb{R}_+$
- Goal: find a shortest cycle that visits all vertices exactly once

Example

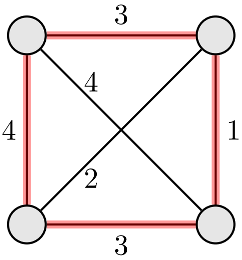


Traveling Salesman Problem

Problem

- Input: Complete undirected graph $G = (V, E)$ with distance $d\colon E \to \mathbb{R}_+$
- Goal: find a shortest cycle that visits all vertices exactly once

Example



length $= 11$

Inapproximability

Theorem

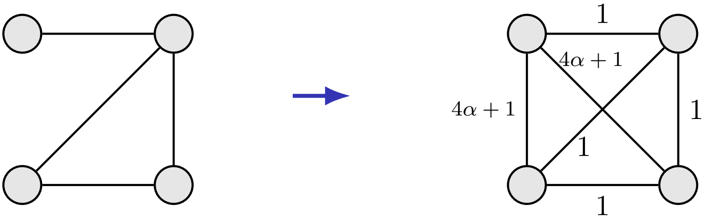
Unless $\mathbf{P} = \mathbf{NP}$, there is no α-approximation algorithm for any $\alpha \geq 1$

- From a Hamiltonian-cycle instance $G = (V, E)$, construct

  NP-complete

$$d(u,v) = \begin{cases} 1 & \text{if } \{u,v\} \in E \\ \alpha|V| + 1 & \text{if } \{u,v\} \notin E \end{cases}$$

- $\mathrm{OPT} = |V|$ if "yes" and $\mathrm{OPT} \geq \alpha|V| + 1$ if "no"

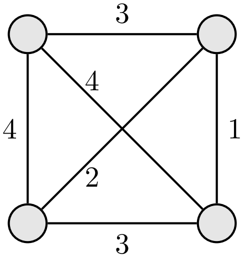


Metric Traveling Salesman Problem

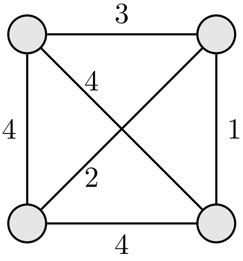
Problem

- Input: Complete undirected graph $G = (V, E)$ with distance $d\colon E \to \mathbb{R}_+$ where $d(u, w) \leq d(u, v) + d(v, w)$ for every $u, v, w \in V$
- Goal: find a shortest cycle that visits all vertices exactly once

Example



Metric



Not metric

Hardness of metric TSP

Theorem

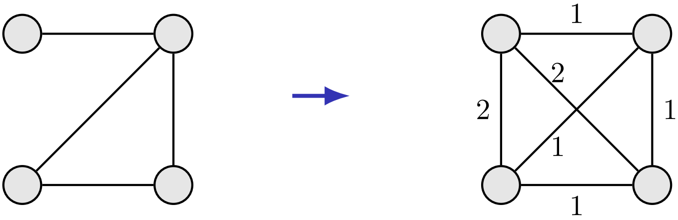
Metric TSP is $\mathbf{NP}$-hard

- From a Hamiltonian-cycle instance $G = (V, E)$, construct

$\mathbf{NP}$-complete

$$d(u,v) = \begin{cases} 1 & \text{if } \{u,v\} \in E \\ 2 & \text{if } \{u,v\} \notin E \end{cases}$$

- $\mathrm{OPT} = |V|$ if "yes" and $\mathrm{OPT} \geq |V| + 1$ if "no"



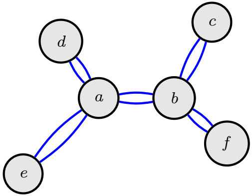
Simple 2-approximation algorithm

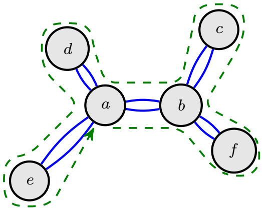
1. Find a minimum spanning tree T;
2. Double each edge in T (making Eulerian graph);
3. Find an Eulerian tour W on this graph (by DFS);
4. Delete all duplicates in W by keeping the first visit to each vertex u;

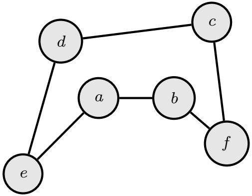
Theorem

The above algorithm is 2-approximation

$\because \mathrm{ALG} \le 2d(T) \le 2\mathrm{OPT}$







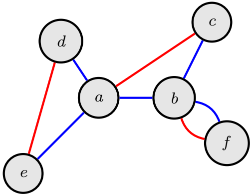
Christofides algorithm [Christofides 1976]

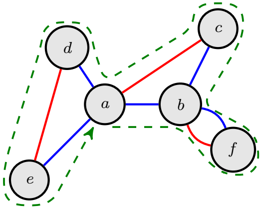
1. Find a minimum spanning tree T;
2. Compute a minimum weight perfect matching M in the complete graph over the odd-degree vertices in T;
3. Find an Eulerian tour W on $T \dot{\cup} M$;
4. Delete all duplicates in W by keeping the first visit to each vertex u;

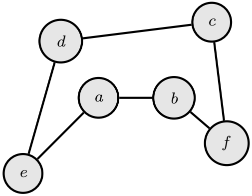
Theorem

The above algorithm is 1.5-approximation

$\because \mathrm{ALG} \leq d(T) + d(M) \leq \mathrm{OPT} + 0.5\mathrm{OPT} = 1.5\mathrm{OPT}$







metric TSP

Can we improve Christofides algorithm?

Theorem [Karlin, Klein, and Gharan 2020]

$\exists(1.5-\epsilon)$-approx. randomized algorithm for metric TSP for some $\epsilon > 10^{-36}$

Theorem [Karlin, Klein, and Gharan 2022/12/13]

$\exists(1.5-\epsilon)$-approx. deterministic algorithm for metric TSP for some $\epsilon > 10^{-36}$

Theorem [Karpinski, Lampis, and Schmied 2015]

$\nexists$ $123/122$-approximation algorithm for metric TSP unless $\mathbf{P} = \mathbf{NP}$