Slido: **#ADA2021**

CSIE 2136 Algorithm Design and Analysis, Fall 2021

National Taiwan University 國立臺灣大學

Handling NP-completeness

Hsu-Chun Hsiao

Announcement

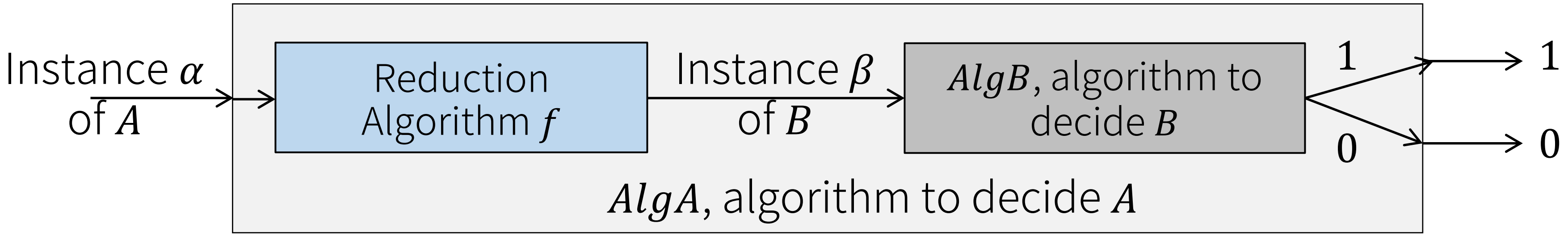
- HW4
  - Due Time (hand-written): 2022/01/04 14:20
  - Due Time (programming): 2022/01/13 14:20
- No TA hours on 1/12 and 1/13
- Final exam: 2022/01/06 14:20-17:20
  - Location: 綜合大講堂
  - Strongly recommend to review homework problems and questions asked in class
  - Details will be released on COOL later
- We will try our best to release the final grades by 1/26. If you need to know your grade earlier, please email us.
- 優良助教問卷

Agenda

- Note on reduction
- Traveling salesman problem
  - Proving NP-completeness
  - Approximation algorithms for metric TSP
- Integer programing problem
  - Proving NP-completeness
  - Reduction to integer programming
- Randomized approximation algorithms
  - 3-CNF-SAT
  - MAX-CUT

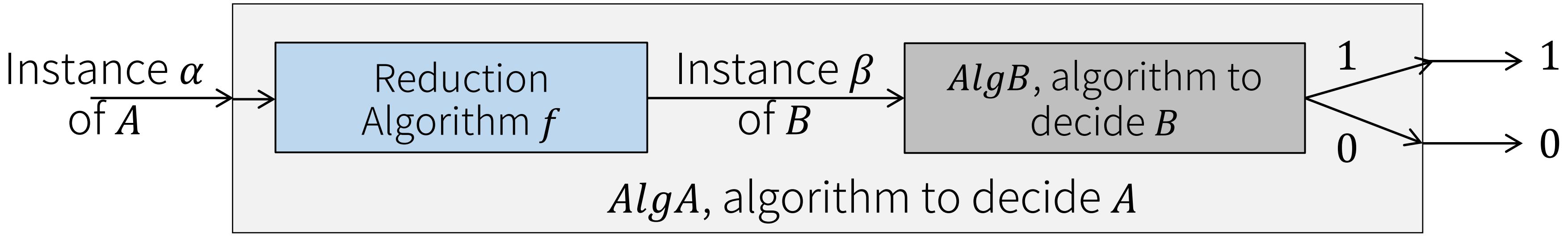
Reduction of decision problems

- A reduction f is an algorithm for transforming every instance of a problem A into an instance of another problem B, and, for all α, $AlgA(\alpha) = 1$ if and only if $AlgB(f(\alpha)) = 1$
  - Thus, we can use $AlgB$ to construct $AlgA$ for solving problem A
- For ease of understanding, try replacing A and B with simple yet concrete problems
  - Example: A is "Can 2 divide x?", and B is "Can y divide x?"



Polynomial-time reduction

- A polynomial-time reduction ($A \leq_p B$) is a polynomial-time algorithm for transforming every instance of a problem A into an instance of another problem B
  - Can help determine the hardness relationship between problems (within a polynomial-time factor)
  - $A \leq_p B$ implies A is no harder than B; equivalently, B is at least at hard as A



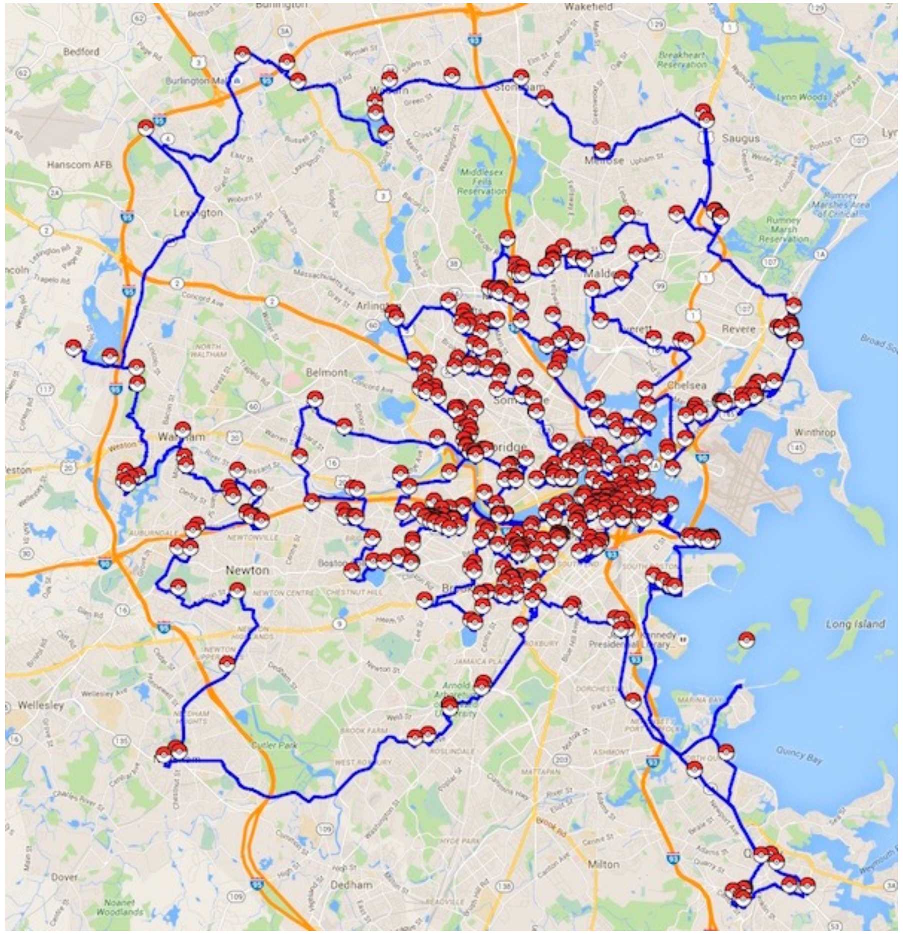
Common misconceptions

Wrong	Correct
❌ f transforms a subset of problem A's instances	✅ f must transform every problem A's instance
❌ $f(\alpha)$ must cover every problem B's instance	✅ $f(\alpha)$ may be a subset of problem B's instances
❌ $A \leq_p B$ implies $B \leq_p A$	✅ Reduction is directional; $A \leq_p B$ does not imply $B \leq_p A$
❌ To prove $A \leq_p B$, we only need to show $AlgA(\alpha) = 1$ implies $AlgB(f(\alpha)) = 1$	✅ While reduction is directional, the proof of correctness must be done both directions. That is, $AlgA(\alpha) = 1$ if and only if $AlgB(f(\alpha)) = 1$
❌ If A can be reduced to B in $O(n^2)$ and there is an algorithm $AlgB$ for B runs in $O(n^3)$, then we can construct an algorithm for A runs in $O(n^3)$	✅ We need to take into account possible size increase after f. If A can be reduced to B in $O(n^2)$ and the size increases to $O(n^2)$, and there is an algorithm $AlgB$ for B runs in $O(n^3)$, then we can construct an algorithm for A runs in $O(n^6)$

Traveling Salesman Problem (TSP)

Traveling Salesman Problem (TSP)

- Optimization problem: Given an undirected complete graph $G = (V, E)$ and a non-negative edge cost function w, find a tour of lowest cost
- Decision problem: Given an undirected complete graph $G = (V, E)$ and a non-negative edge cost function w, find a tour of cost at most k
- Tour = visit each vertex exactly once and return to the beginning



A tour to catch every (518) Pokémon in Boston
https://www.math.uwaterloo.ca/tsp/poke/index.html

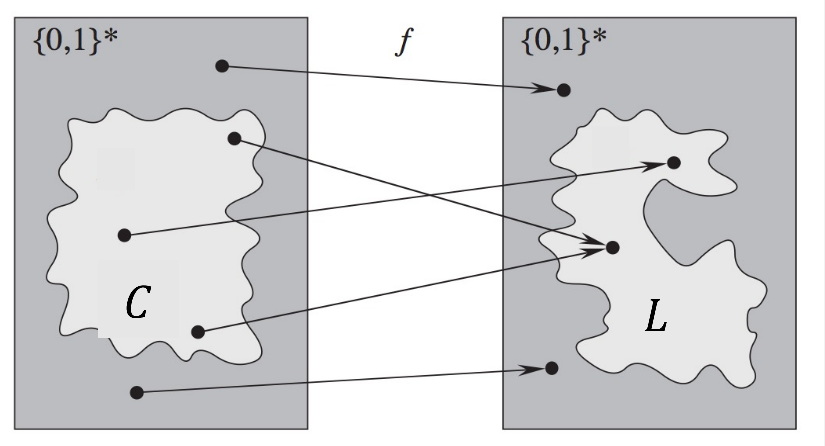
The TSP Problem

$\text{TSP} = \{\langle G, w, k\rangle : G = (V, E)$ is a complete graph, w is a non-negative cost function for edges, G has a traveling-salesman tour with cost at most $k\}$

- Prove that $\text{TSP} \in$ NP-COMPLETE
- Polynomial-time reduction: HAM-CYCLE $\leq_p$ TSP

Step-by-step approach for proving L in NPC:

1. Prove $L \in$ NP
2. Prove $L \in$ NP-hard ($C \leq_p L$)
   ① Select a known NPC problem C
   ② Construct a reduction f transforming every instance of C to an instance of L
   ③ Prove that x in C if and only if $f(x)$ in L for all x in $\{0,1\}^*$
   ④ Prove that f is a polynomial time transformation

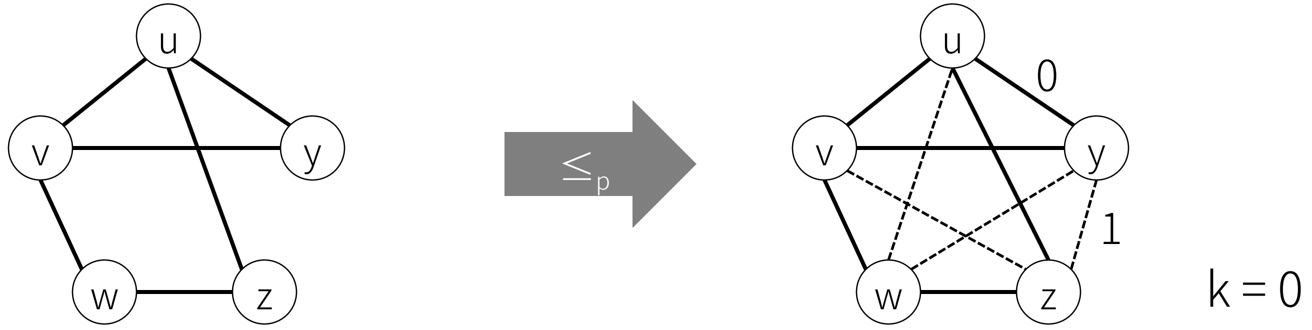


The TSP Problem

$\mathrm{TSP} = \{\langle G, w, k\rangle : G = (V, E)$ is a complete graph, w is a non-negative cost function for edges, G has a traveling-salesman tour with cost at most $k\}$

② Construct a reduction f transforming every HAM-CYCLE's instance $G_H = (V_H, E_H)$ to a TSP instance with cost at most k

- We construct a TSP instance in which G is a complete graph with $V = V_H$, and $w(i,j) = 0$ if $(i,j) \in E_H$; $w(i,j) = 1$, otherwise.
- With this reduction function, we set $k = 0$



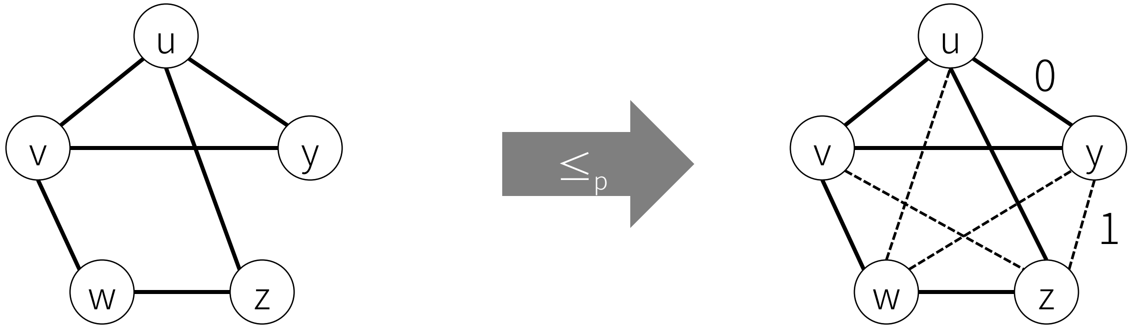
HAM-CYCLE $\leq_p$ TSP

③ Prove that $x \in$ HAM_CYCLE $\Leftrightarrow f(x) \in$ TSP

Correctness proof: $x \in$ HAM-CYCLE $\Leftrightarrow f(x) \in \mathbf{TSP}$

- More specifically, we want to prove that G contains a Hamiltonian cycle $h = \langle v_1, v_2, \ldots, v_n, v_1 \rangle$ if and only if $\langle v_1, v_2, \ldots, v_n, v_1 \rangle$ is a traveling-salesman tour with cost at most 0

u, y, v, w, z, u is a Hamiltonian cycle ⇔ u, y, v, w, z, u is a traveling-salesman tour with cost 0

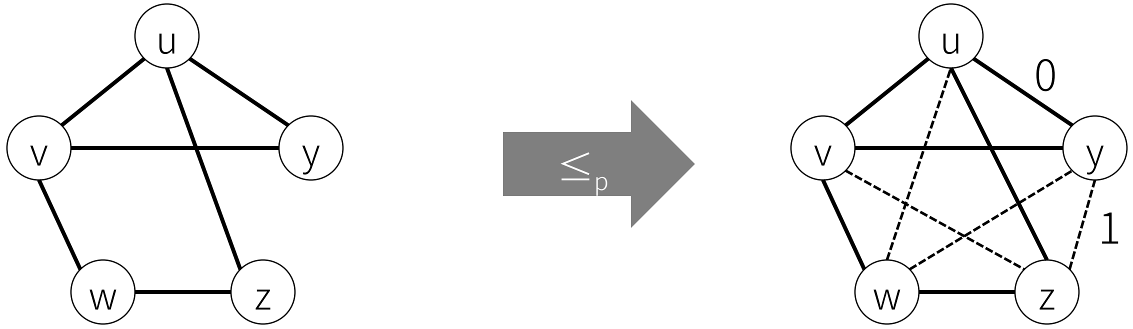


HAM-CYCLE $\leq_p$ TSP

③ Prove that x ∈ HAM_CYCLE ⇔ $f(x)$ ∈ TSP

Correctness proof: x ∈ HAM-CYCLE ⇒ $f(x) \in \mathbf{TSP}$

- Suppose the Hamiltonian cycle is $h = \langle v_1, v_2, \dots, v_n, v_1 \rangle$
- $\Rightarrow h$ is also a tour in the transformed TSP instance
- $\Rightarrow$ The cost of the tour h is 0 since there are n consecutive edges in E, and so has cost 0 in $f(x)$
- $\Rightarrow f(x) \in$ TSP ($f(x)$ has a TSP tour with cost ≤ 0)



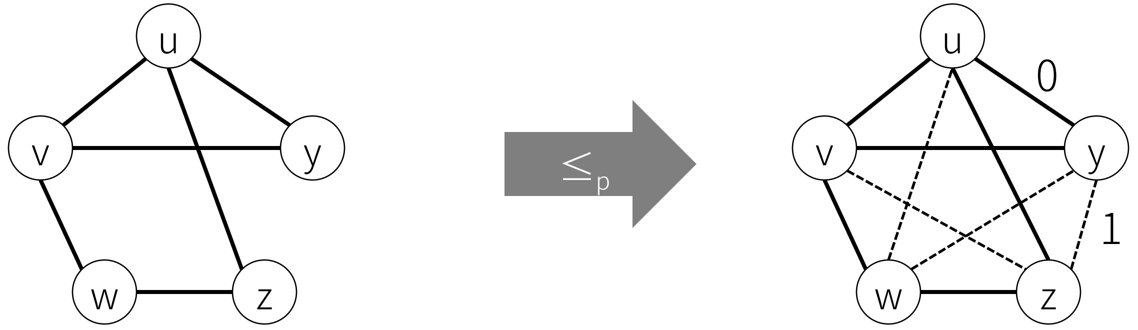
u, y, v, w, z, u is a Hamiltonian cycle   u, y, v, w, z, u is a traveling-salesman tour with cost 0

HAM-CYCLE $\leq_p$ TSP

③ Prove that x ∈ HAM_CYCLE ⇔ $f(x)$ ∈ TSP

Correctness proof: $f(x) \in \mathbf{TSP} \Rightarrow x \in$ HAM−CYCLE

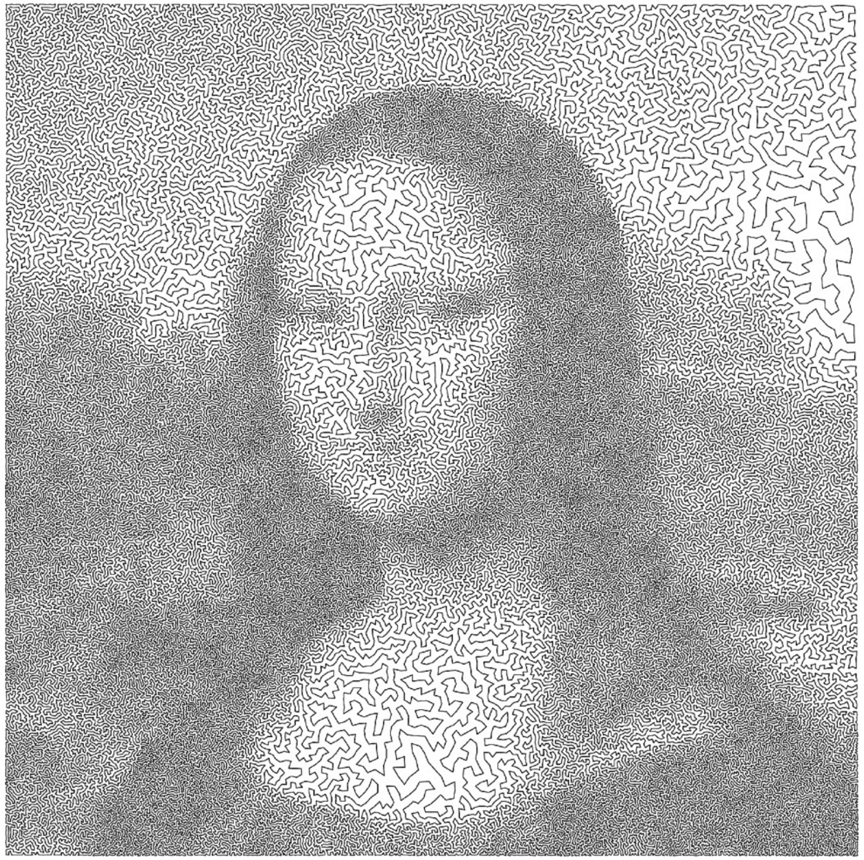
- Suppose after reduction, there is a TSP tour with cost ≤ 0. Let it be $\langle v_1, v_2, \ldots, v_n, v_1 \rangle$
- ⇒ The TSP tour contains only edges in E_H
- ⇒ Thus, $\langle v_1, v_2, \ldots, v_n, v_1 \rangle$ is a Hamiltonian cycle ($x \in$ HAM-CYCLE).



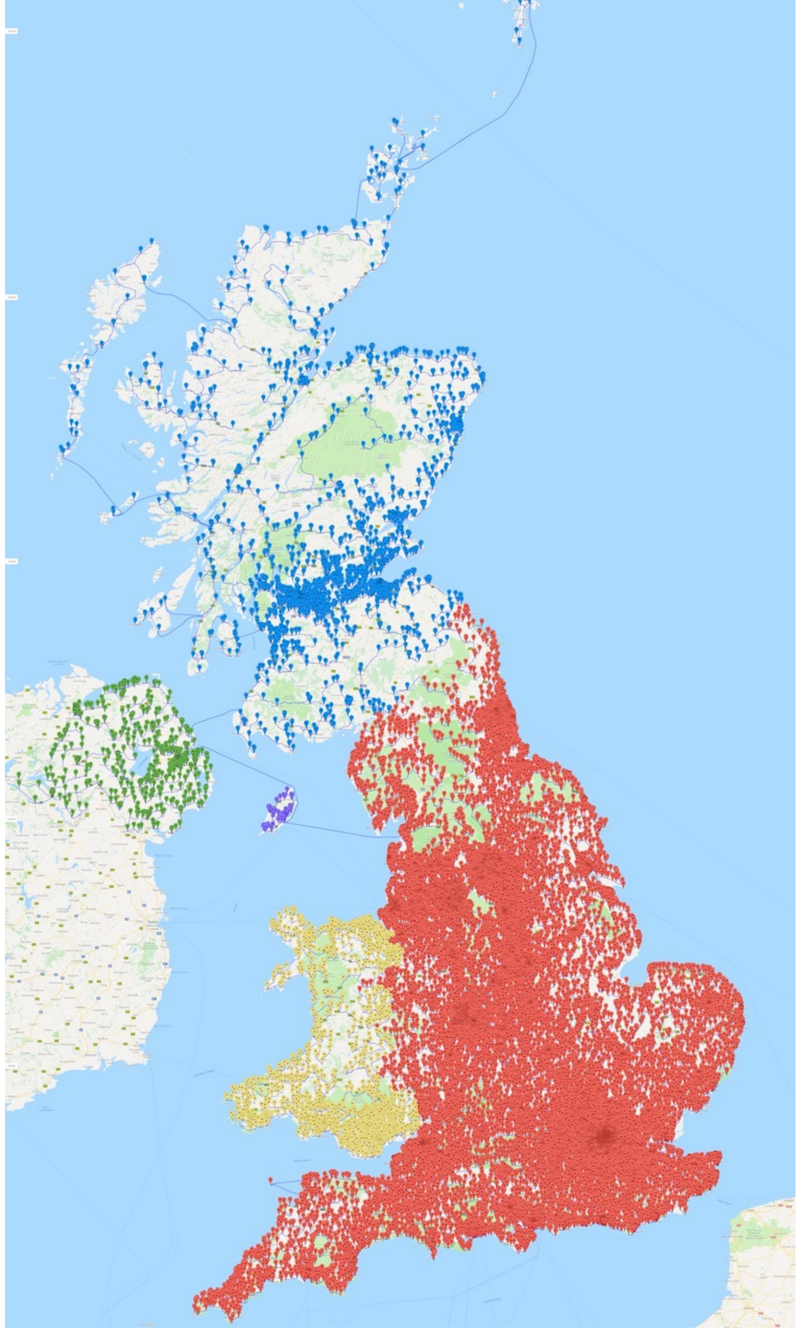
u, y, v, w, z, u is a Hamiltonian cycle

u, y, v, w, z, u is a traveling-salesman tour with cost 0

TSP arts and challenges



Mona Lisa TSP: $1,000 Prize for a 100,000-city challenge problem
http://www.math.uwaterloo.ca/tsp/



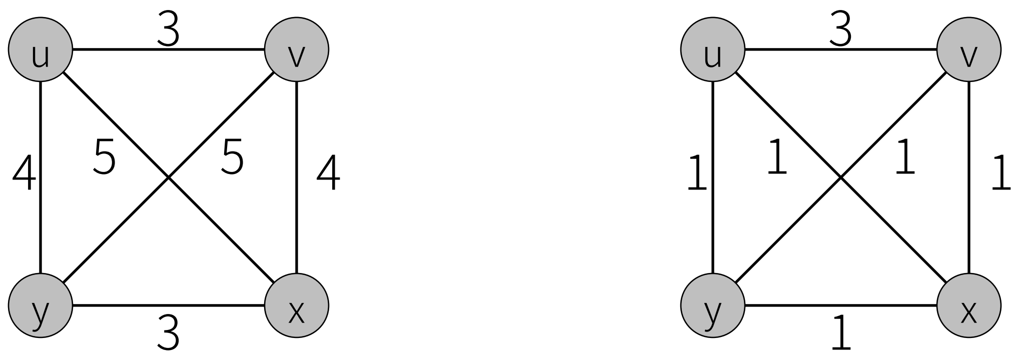
Shortest possible tour to nearly every (49687) pub in the United Kingdom
https://www.math.uwaterloo.ca/tsp/uk/

Traveling Salesman Problem (TSP): Approximation

Metric TSP

- Optimization problem: Given a set of cities and their pairwise distances, find a tour of lowest cost that visits each city exactly once, and the pairwise distances satisfy triangle inequality.
  - Triangle inequality: $\forall u, v, w \in V, d(u, w) \leq d(u, v) + d(v, w)$.

Satisfy triangle inequality   Do not satisfy triangle inequality



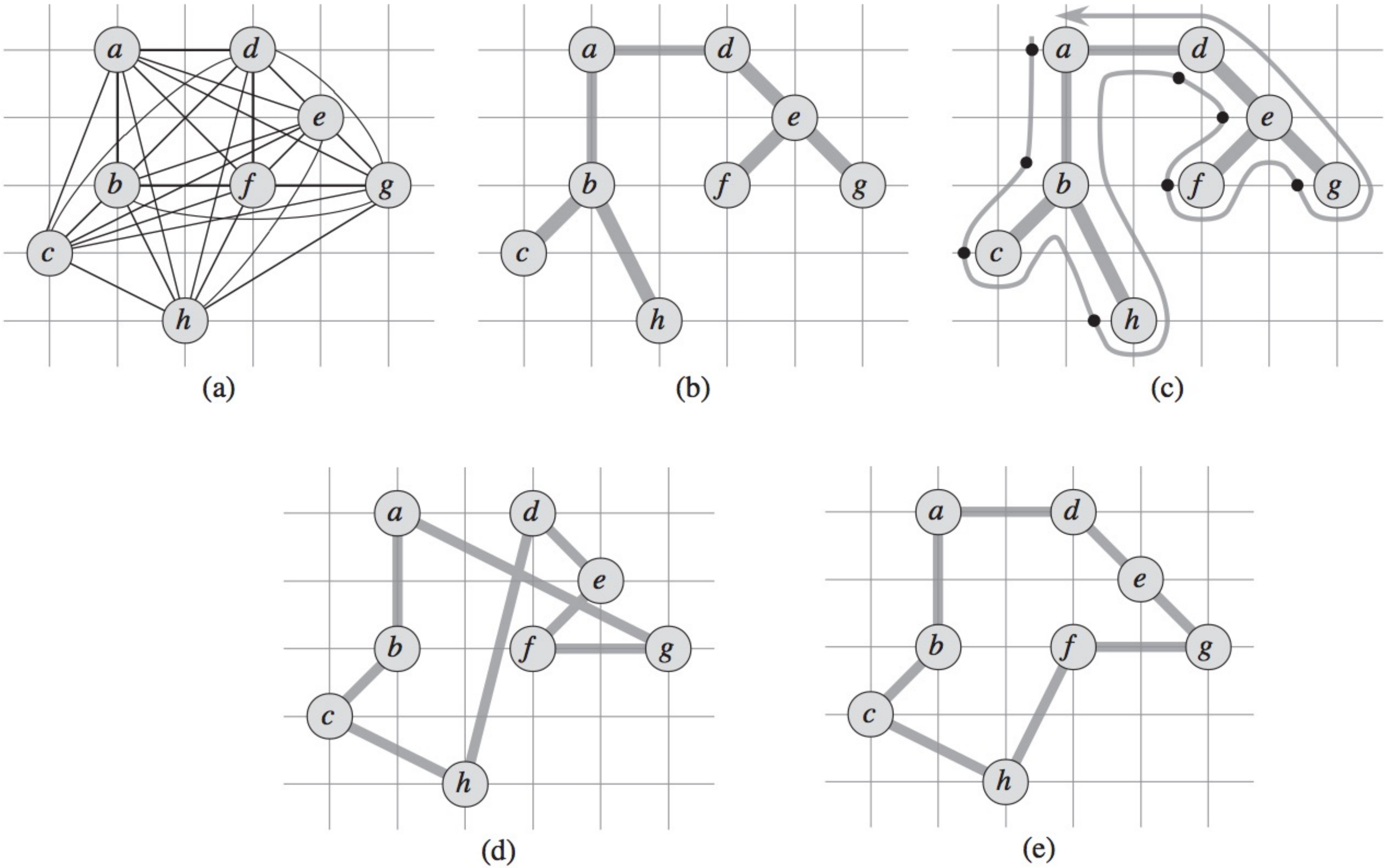
Show that Metric TSP is also in NPC

Hint: reduce from either HAM-CYCLE or general TSP

2-approximation algorithm for Metric TSP

```
APPROX-TSP-TOUR(G)
1. select a vertex r ∈ G.V to be a "root" vertex
2. grow a minimum spanning tree T for G
3. let H be the list of vertices visited in a preorder tree walk of T
4. return H
```

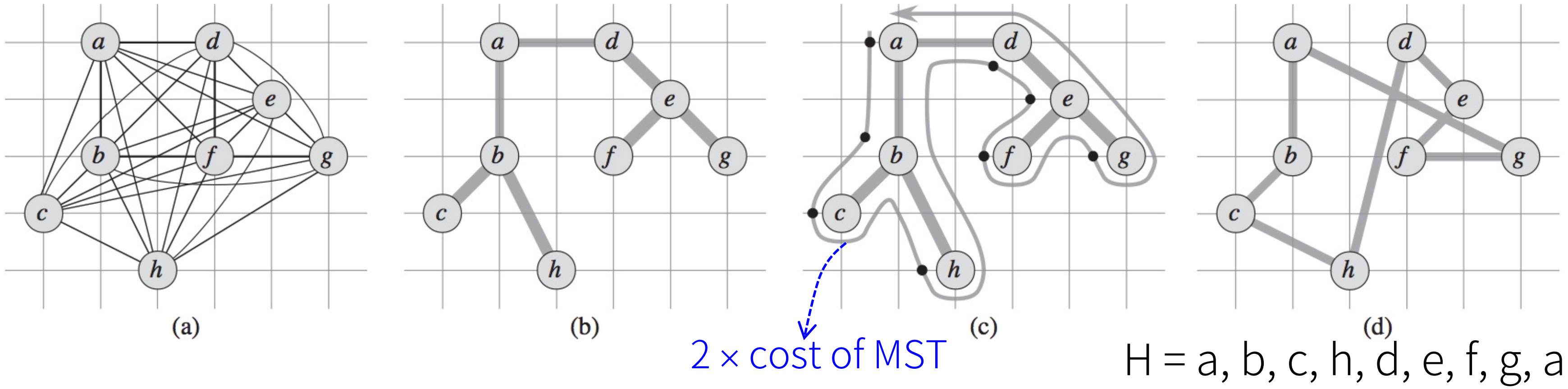
- Running time is dominated by finding a MST
  - MST is in P: $O(V^2)$ when using adjacency matrix
- Claim: Approximation ratio $\rho(n) = 2$



H = a, b, c, h, d, e, f, g, a

OPT H^* = a, b, c, h, f, g, e, d, a

2-approximation algorithm for Metric TSP

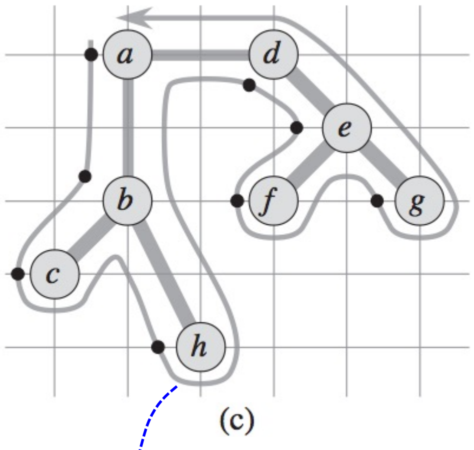


H = a, b, c, h, d, e, f, g, a

- Let H^* denote an optimal tour, H the TSP tour found by the algorithm, T^* a MST
- With triangle inequality, immediately we have $w(H) \leq 2w(T^*)$
- Also, H^* is formed by some tree T plus some edge e, i.e., $w(H^*) = w(T) + w(e)$
- $\Rightarrow w(T^*) \leq w(H^*)$
- $\Rightarrow w(H) \leq 2w(T^*) \ \leq 2w(H^*)$
- $\Rightarrow \rho(n) = 2$

1.5-approximation algorithm for Metric TSP

- Can we do better than $\rho(n) = 2$?
  - A better intermediary than $2w(T^*)$ in $w(H) \leq 2w(T^*) \leq 2w(H^*)$?
- Homework



$2 \times$ cost of MST

Theorem 35.3 General TSP (when triangle inequality may not hold)

If P ≠ NP, there is no polynomial-time approximation algorithm with a constant ratio bound ρ for the general TSP.

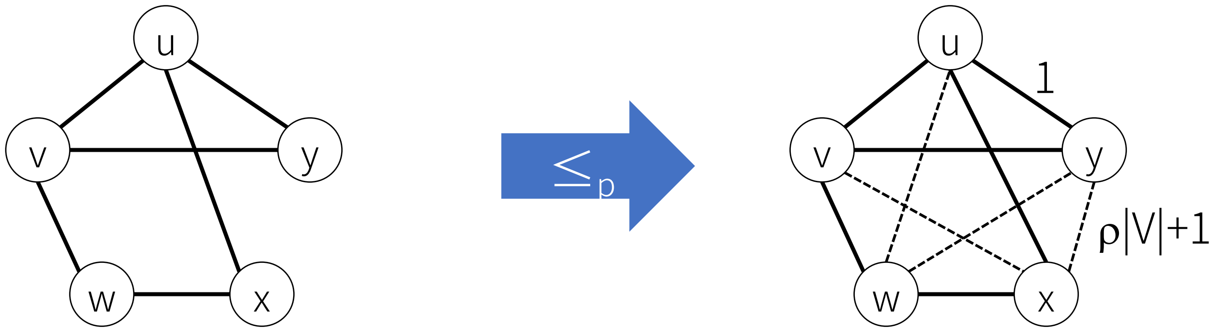
Proof by contradiction

- Suppose there is such an algorithm Alg_{AT} to approximate TSP with a constant ρ. We will use Alg_{AT} to construct Alg_{HC} to solve HAM-CYCLE in polynomial time.
- Consider the following reduction algorithm f converting an instance of HAM-CYCLE α into an instance of TSP $f(\alpha)$
  - $\alpha = \{G = (V, E)\}; f(\alpha) = \{G' = (V, E'), w, k = \rho|V|\}$
  - That is, we construct a TSP instance with a complete graph $G' = (V, E')$, where $w(u, v) = 1$ if $(i, j) \in E$; $w(u, v) = \rho|V| + 1$, otherwise.
  - Run Alg_{AT} on $f(\alpha)$
  - If $Alg_{AT}\big(f(\alpha)\big)$ returns a tour whose cost ≤ $\rho|V|$, then $Alg_{HC}(\alpha) = 1$ (i.e., G contains a Hamiltonian cycle); otherwise, $Alg_{HC}(\alpha) = 0$.

Proof by contradiction (cont'd)

- Correctness of reduction
  - If G has an HC: G' contains a tour of cost $|V|$ by picking edges in E, each with cost of 1. Then, Alg_{AT} guarantees to return a tour whose cost $\leq \rho|V|$.
  - If G' has a tour whose cost $\leq \rho|V|$: the tour must consist of edges in E only, and thus it's also a HC in G

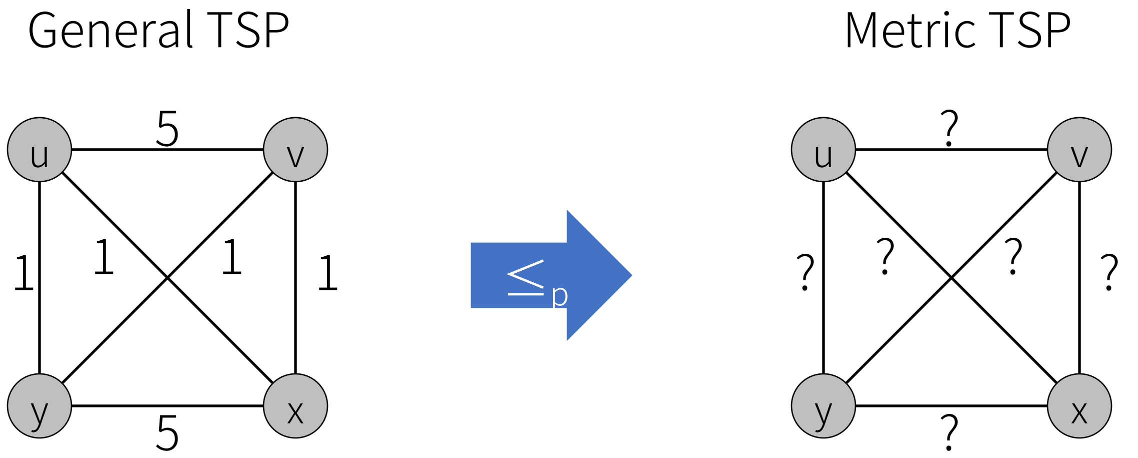
$\Rightarrow$ Alg_{HC} can solve HAM-CYCLE in polynomial time, contradiction!



u, y, v, w, x, u is a Hamiltonian Cycle

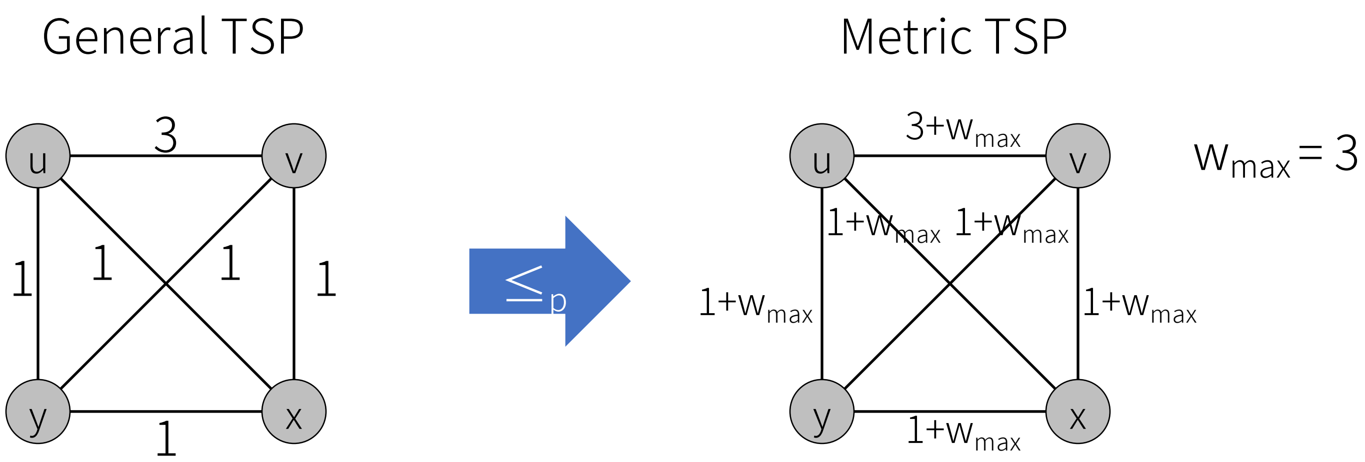
u, y, v, w, x, u is a traveling-salesman tour with cost |V|

Exercise 35.2-2 Show how in polynomial time we can transform one instance of the traveling-salesman problem into another instance whose cost function satisfies the triangle inequality. The two instances must have the same set of optimal tours. Explain why such a polynomial-time transformation does not contradict Theorem 35.3, assuming that P≠NP.

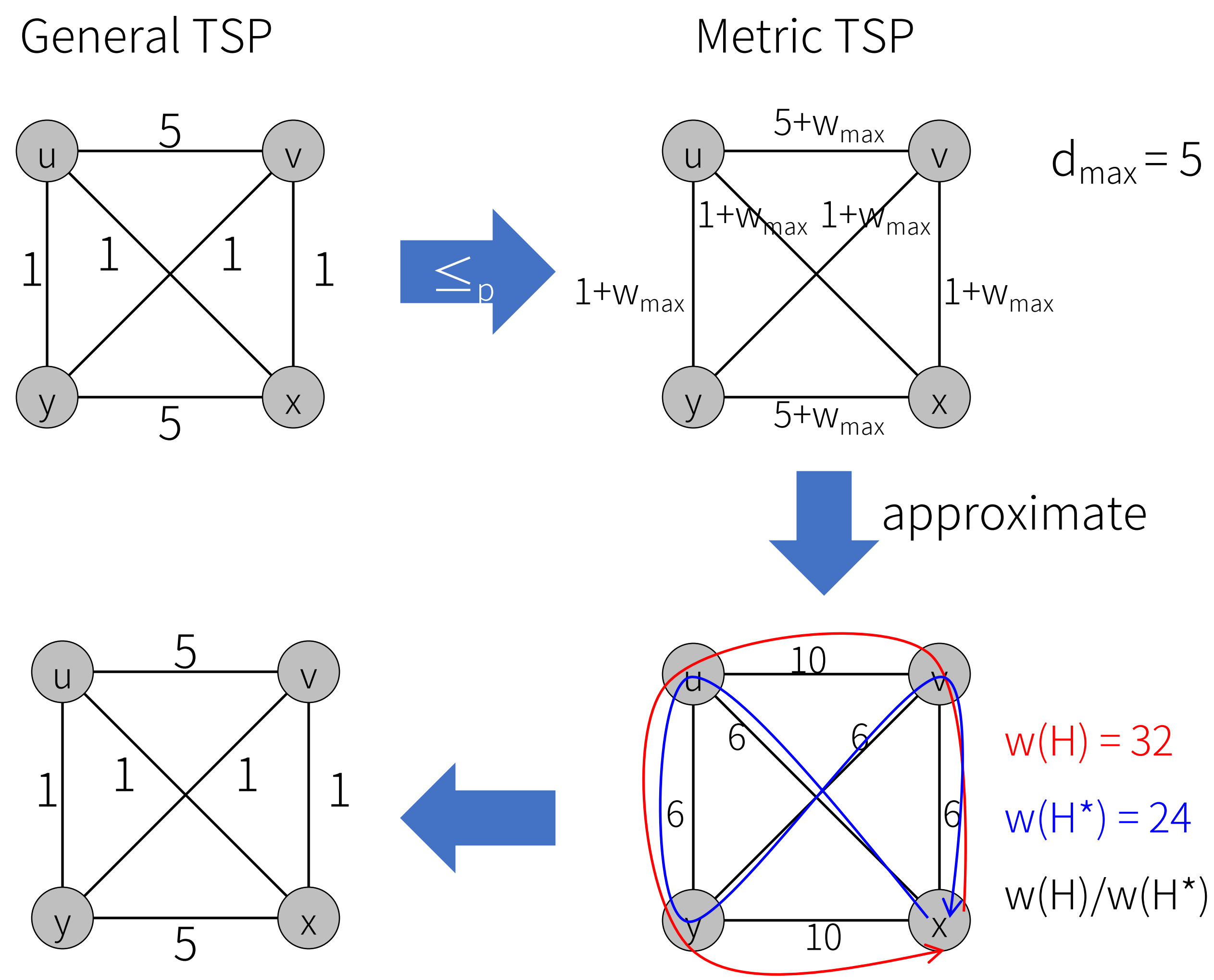


Exercise 35.2-2

- For example, we can add w_{max} (the largest cost) to each edge
- G contains a tour of minimum cost k ⇔ G' contains a tour of minimum cost $k\ + w_{max} * |V|$
- G' satisfies triangle inequality because $\forall t, u, v \in V$,
$$w'(u, v) = w(u, v) + w_{max} \leq 2 * w_{max} \leq w'(t, u) + w'(t, v)$$



Exercise 35.2-2



$w(H) = 12$

$w(H^*) = 4$

$w(H)/w(H^*) > 2$!

$w(H) = 32$

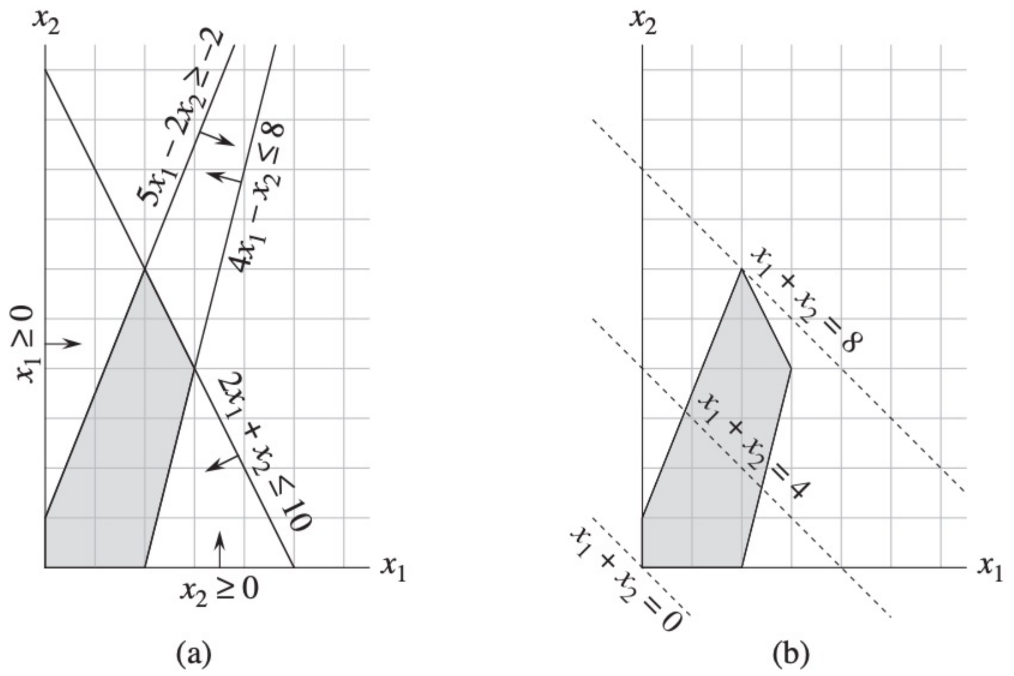
$w(H^*) = 24$

$w(H)/w(H^*) \leq 2$

(Linear) Integer Programming

Linear programming [Ch. 29]

- Optimize a linear function subject to a set of linear inequalities
- Example
  - Maximize $x_1 + x_2$
  - Subject to
    - $4x_1 - x_2 \leq 8$
    - $2x_1 + x_2 \leq 10$
    - $5x_1 - 2x_2 \geq -2$
    - $x_1, x_2 \geq 0$



Linear programming [Ch. 29]

Maximize $c^T x$

Subject to $Ax \leq b$, and $x \geq 0$

Notation

- $A = (a_{ij})$: $m \times n$ coefficient matrix
- $b = (b_i)$: $\mathrm{m} \times 1$ requirement vector
- $c = (c_j)$: $n \times 1$ cost vector
- $x = (x_j)$: $n \times 1$ vector of unknown variables

(Linear) integer programming [ch. 35.4]

- Integer programing: x_j are integers
  - Decision problem: whether there is a feasible solution $x = (x_j)$ subject to $Ax \le b$ and $x_i \in \mathbb{Z}_0^+$
- Mixed integer programming: some of x_j are integers
- While the linear programing problem is in class P, the integer programming problem (decision version) is NP-complete
- Fortunately, there are powerful integer programming solvers, which haven solved many integer programming instances

Show that the integer programming problem is NP-complete

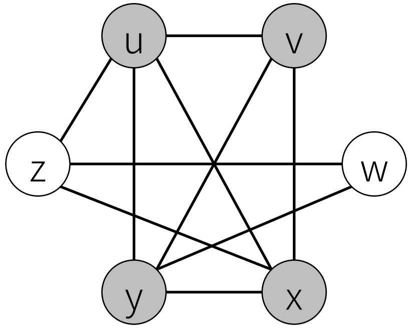
Hint: Consider a reduction from 3-CNF-SAT. How would you reduce an 3-CNF-SAT instance, say, $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_4 \vee x_5)$, to an integer programming instance?

Modeling vertex cover via integer programming

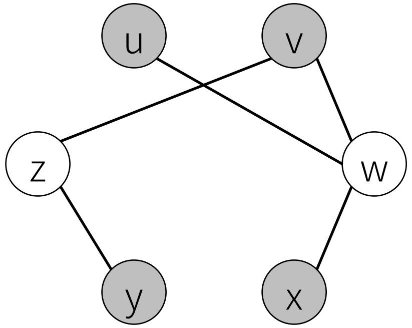
- Vertex cover (optimization): find a vertex cover of minimum size in G
  - A vertex cover of $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(w, v) \in E$, then $w \in V'$ or $v \in V'$ or both
- Integer programming formulation
  - Variables: $x_i \in \{0,1\}$ represents whether vertex v_i is covered
  - Minimize: $\Sigma_{i=1}^{n} x_i$
  - Subject to
    - $x_i + x_j \geq 1, \forall e = (v_i, v_j) \in E$
    - $x_i \in \{0,1\}, \forall i = 1,2, \dots, |V|$

Clique, Independent-Set, Vertex-Cover

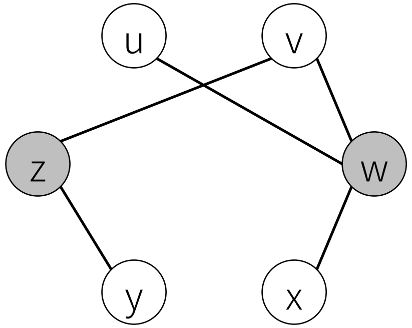
- The following are equivalent for $G = (V, E)$ and a subset V' of V:
  1. V' is a clique of G
  2. V' is an independent set of G_c
  3. $V - V'$ is a vertex cover of G_c



Clique
$V' = \{u, v, x, y\}$ in G



Independent set
$V' = \{u, v, x, y\}$ in G_c



Vertex cover
$V - V' = \{z, w\}$ in G_c

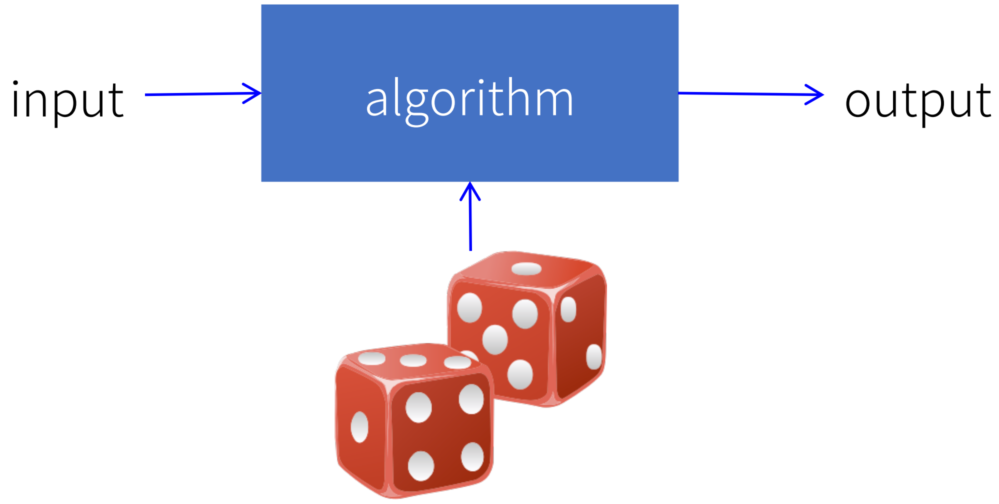
Show the integer programming formulation for the clique problem (optimization)

Show that the independent-set problem is NP-Complete.
Show the integer programming formulation for the independent-set problem (optimization)

Randomized Approximation Algorithms

Randomness

- A randomized algorithm is an algorithm that employs a degree of randomness as part of its logic
- A randomized data structure is a data structure that employs a degree of randomness as part of its logic
- A randomized algorithm's behavior is determined not only by its input but also by values produced by a random-number generator



Randomized approximation algorithm

	Exact	Approximate
Deterministic	MST	APPROX-TSP-TOUR
Randomized	Quick Sort	MAX-3-CNF-SAT MAX-CUT

MAX-3-CNF

- 3-CNF-SAT: Satisfiability of Boolean formulas in 3-conjunctive normal form (3-CNF)
  - 3-CNF = AND of clauses, each is the OR of exactly 3 distinct literals
  - A literal is an occurrence of a variable or its negation, e.g., x_1 or $\neg x_1$
- 3-CNF-SAT is a decision problem. What should be an optimization version of 3-CNF-SAT?

MAX-3-CNF

- MAX-3-CNF: find an assignment of the variables that satisfies as many clauses as possible
  - Closeness to optimum is measured by the fraction of satisfied clauses
- Can you design a randomized 8/7-approximation algorithm?

$$\underline{(x_1 \vee \neg x_1 \vee \neg x_2)} \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

$\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 1 \rangle$ satisfies 3 clauses
$\langle x_1, x_2, x_3, x_4 \rangle = \langle 1, 0, 1, 1 \rangle$ satisfies 2 clauses

Note that this clause is always SAT since it always evaluates to 1. For simplicity, we can assume no clause containing both of a literal and its negation.

Randomized 7/8-approximation algorithm for MAX-3-CNF-SAT

- A randomized 8/7-approximation algorithm:
  - 丟硬幣決定變數要設成 0 或是 1
  - That's it!

Theorem 35.6

Given an instance of MAX-3-CNF-SAT with n variables $x_1, x_2, \dots, x_n$ and m clauses, the randomized algorithm that independently sets each variable to 1 with probability 1/2 and to 0 with probability 1/2 is a randomized 8/7-approximation algorithm.

* Satisfying 7/8 of the clauses in expectation

Theorem 35.6

Given an instance of MAX-3-CNF-SAT with n variables $x_1, x_2, \dots, x_n$ and m clauses, the randomized algorithm that independently sets each variable to 1 with probability 1/2 and to 0 with probability 1/2 is a randomized 8/7-approximation algorithm.

Proof

- A clause that contains both a variable and its negation is always evaluated to 1
- The rest of the clauses is the OR of exactly 3 distinct literals, and no variable and its negation appear at the same time
  - $\Pr[x_i = 0] = \Pr[x_i = 1] = 1/2$
  - $\Rightarrow$ for all $x_1 \neq x_2 \neq x_3$, $\Pr[(x_1 \vee x_2 \vee \neg x_3) = 0] = 1/8$
- $\Rightarrow$ E[# of satisfied clauses] = m * E[clause j is satisfied] $\geq m * (1 - \frac{1}{8}) = \frac{7}{8} m$
- $\Rightarrow$ $\rho(n)$ = max # of satisfied clauses / E[# of satisfied clauses] = 8/7

MAX-CUT

- Optimization problem: Given an unweighted undirected graph $G = (V, E)$, find a cut whose size is maximized
  - A cut partitions V into V_0 and V_1; a cut consists of the edges across the partition
- Decision problem: Given an unweighted undirected graph $G = (V, E)$, there exists a cut whose size is k
- MAX-CUT problem is NP-complete
  - C.f. MIN-CUT is in P
- Can you design a randomized 2-approximation algorithm?

Randomized 2-approximation algorithm for MAX-CUT

- Randomly assign each vertex to either V_1 and V_2 with equal probablity
- Done!

Proof

- Let C be the cut found by the algorithm; $C^*$is the maximum cut
- For any edge $e = (u, v)$ on G, the probability that $e \in C$ is ½
- Let x_e be an indicator variable for event $e \in C$; that is, $x_e = 1$ if $e \in C$, otherwise, 0.
- => $\mathrm{E}[|C|] = E[\Sigma_{e\in E} x_e] = \Sigma_{e\in E} E[x_e]$

$$= \Sigma_{e\in E} Pr[e \in C] * 1 + Pr[e \notin C] * 0 = \frac{|E|}{2} \leq \frac{|C^*|}{2}$$

Suppose vertices are numbered from 1 to n, show that the following greedy algorithm achieves 2-approximation max cut:

1. Initially $V_0^1 = \{1\}, \mathrm{V}_1^1 = \{\}$
2. Adding the ith vertex to the subset that results in a better cut, say V_x:$V_x^i = V_x^{i-1} \cup \{i\}, V_{1-x}^i = V_{1-x}^{i-1}$
3. Output V_0^n, V_1^n

Hint: consider the edges introduced by adding the ith vertex