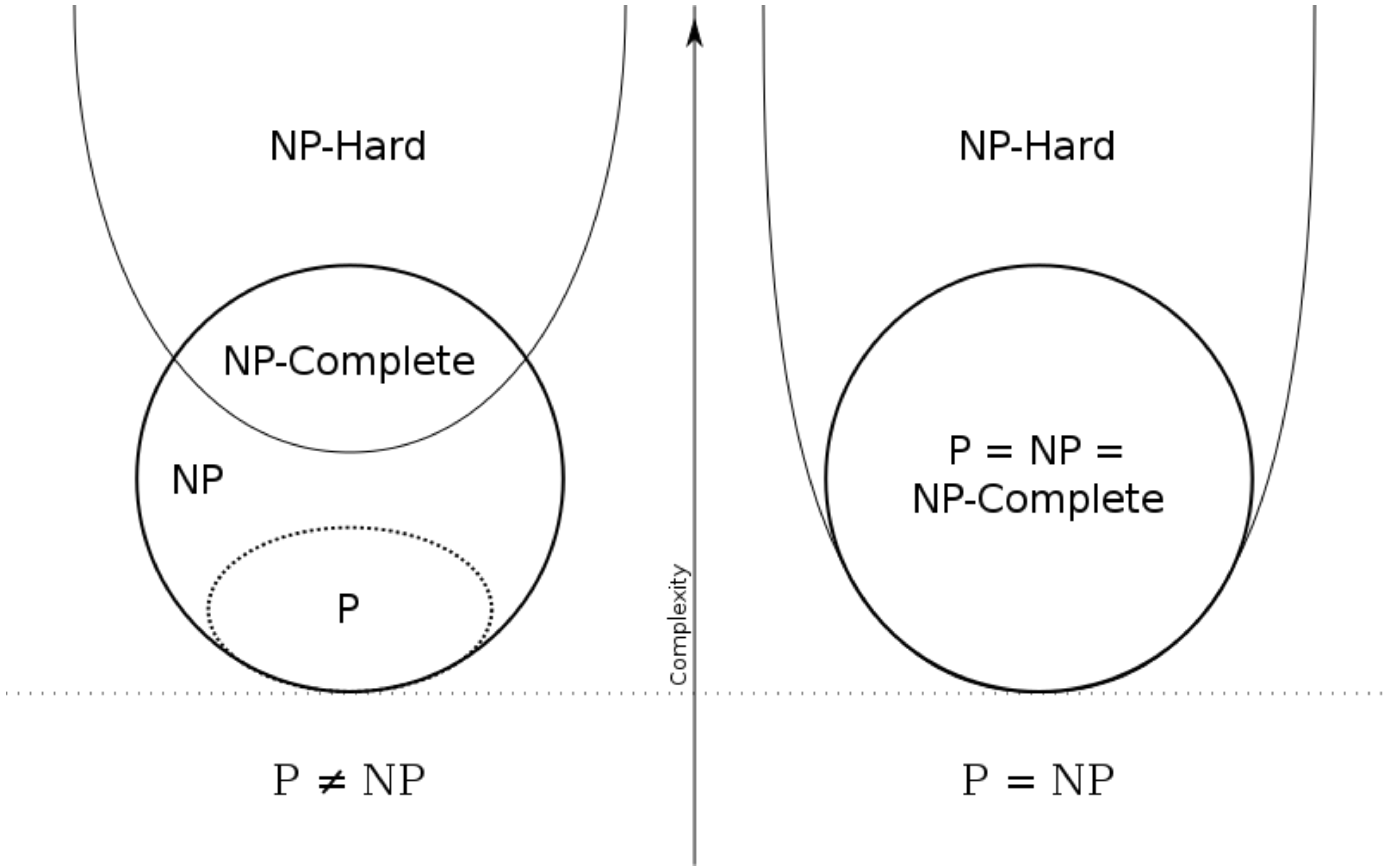
Algorithms

Proof of NP-Completeness

Preliminary

- Polynomial time vs. Exponential time
- Problem Complexity
- Universal Computer Model: Turing machine
- Deterministic vs. Non-deterministic Turing Machine
- Optimization Problem vs. Decision Problem


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

Problem Complexity

- **P Problem**
  - problem solved by deterministic algorithm in polynomial time.
  - problem solved by deterministic Turing machine in polynomial time.
- **NP Problem (Non-deterministic Polynomial)**
  - problem solved by non-deterministic algorithm in polynomial time
  - problem solved by non-deterministic Turing machine in polynomial time
- **NP-Hard**
  - problem which every NP problem is polynomial reducible to.
- **NP-Complete**
  - problem which is NP and is also NP-Hard

Proof of
NP Complete Problems

Satisfiability Problem

- **Cook's theorem**
  - **The satisfiability problem is NP-complete**
  - **NP=P iff the satisfiability problem is a P problem**
- **Boolean formula**
  - **literal: x_1, $\sim x_1$**
  - **clause: $(\sim x_1 \vee x_2)$**
  - **formula: conjunctive normal form, $(\sim x_1 \vee x_2) \wedge x_1 \wedge x_3$**
  - **Every Boolean formula can be transformed into CNF**
  - **Logical consequence**
    - e.g. $x_1=0$, $x_2=1$, $x_3=1$, $F=(\sim x_1 \vee x_2) \wedge x_1 \wedge x_3 = 0$
    - **A formula *G* is a logical consequence of formula *F* iff whenever *F* is true, *G* is true**

Satisfiability Problem (cont.)

- **A boolean expression is said to be satisfiable if there exists an assignment of 0s and 1s to its variable such that the values of the expression is 1.**
- **SAT**
  - **Given a Boolean formula, determine whether this formula is satisfiable or not**

    e.g. $F = (\sim x_1 \vee x_2) \wedge x_1 \wedge x_3$
  - **the first NP-complete problem ever found**

Proof of NP-Completeness for SAT

- **SAT is NP-Complete**
  - **SAT is NP**

    ∵guess a truth assignment &

    check that it satisfies the expression in polynomial time
  - **SAT is NP-Hard**

    ∵Turing machine can be described by a Boolean expression

    i.e. expression is satisfiable iff the Turing machine will terminate for the given input.

    ∴ Any NP problem can be described by an instance of a SAT

Proof of NP-Complete

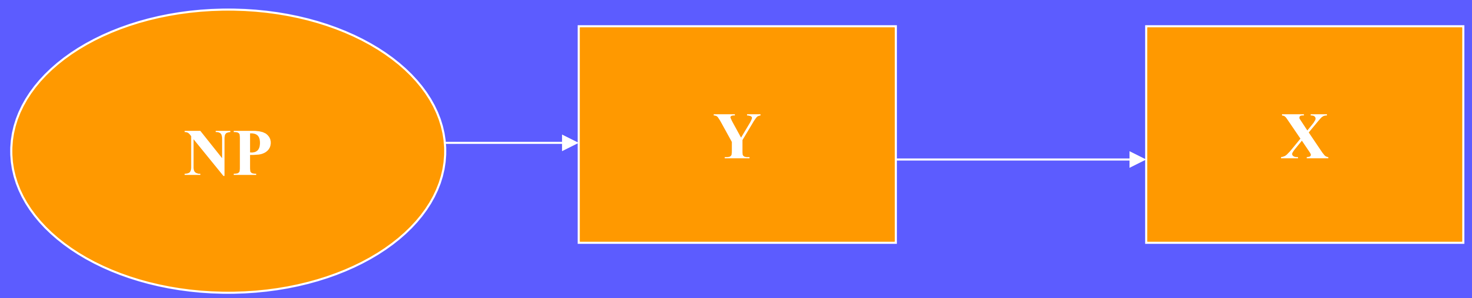
- **Lemma 11.3**

  A problem X is an NP-Complete problem

  if

  (1) X belongs to NP

  (2) Y is polynomial reducible to X, for some NP-complete problem Y



NP-Complete

Clique Problem

- **Clique: complete subgraph**

  *** complete: each pair of vertices is adjacent**

- **Clique problem**

  □ **optimization problem:**

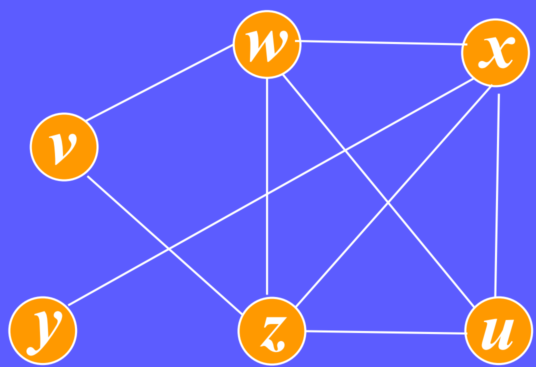
  given an undirected graph G,

  find the maximum clique

  □ **decision problem**

  given an undirected graph G and an integer k,

  determine whether *G* has a clique of size ≥ *k*



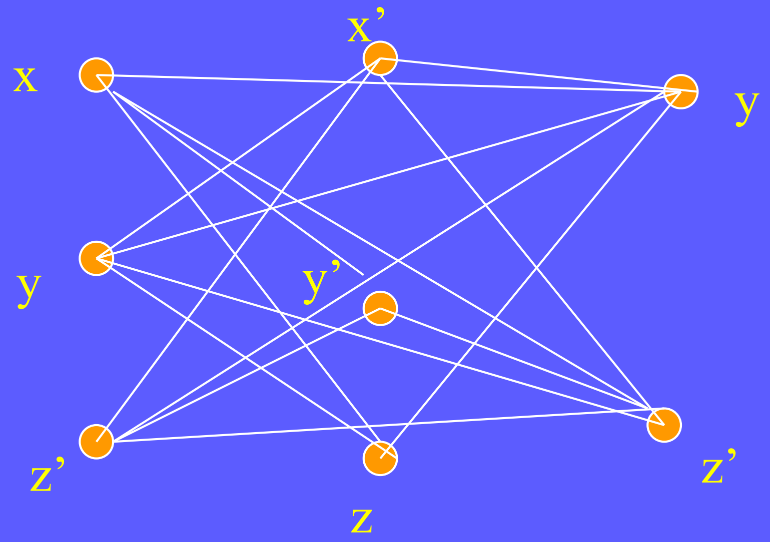
maximum clique:
{w, x, u, z}

{v, w, z}: maximal
{u, w, z}: not maximal

Proof of NP-Completeness of Clique Problem

- **clique is NP**
  - **∵ guess a clique of size >= k & check it in polynomial time**
- **clique is NP-hard**
  - **reduce SAT to clique in polynomial time**
  - **construct a graph G for an Boolean expression in CNF , $E = E_1 \wedge E_2 \ldots \wedge E_m$**

example: $E = (x \vee y \vee z') \wedge (x' \vee y' \vee z) \wedge (y \vee z')$



Clique

{x, z, y} ⇒ x=1, z=1, y=1

⇒ $E = (1 \vee 1 \vee 0) \wedge (0 \vee 0 \vee 1) \wedge (1 \vee 0)$

{y, x', z'} ⇒ y=1, x=0, z=0 ⇒ E=1

{y, z, y} ⇒ y=1, z=1 ⇒ E=1

{z', x', z'} ⇒ z=0, x=0 ⇒ E=1

Proof of NP-Completeness of Clique Problem (cont.)

(1) If SAT 有解(Satisfiable)，對應的 Graph 就有size ≥ m 的 Clique。

如果 SAT 有解，那麼每個 Clause 至少有一個出現的變數是 1。

我們只要在 Graph 中每個 Column 選取這些變數所對應的 Vertices，就會形成 Clique。

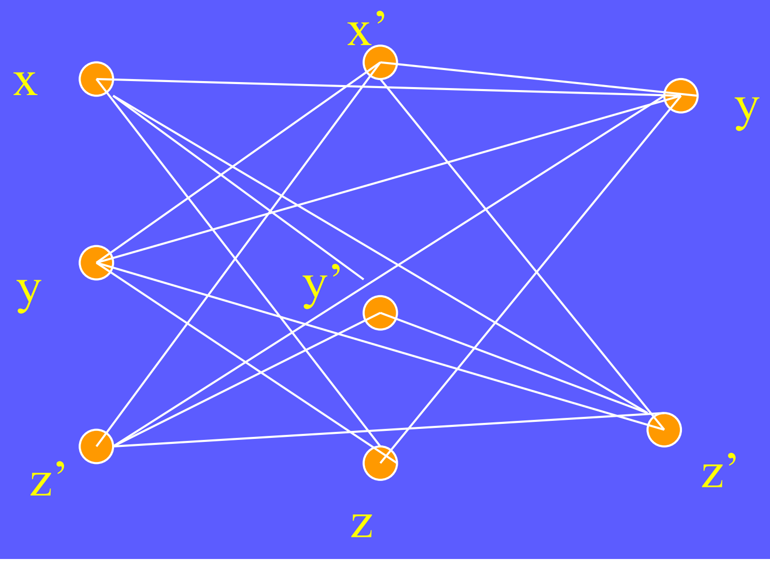
(2) If Graph 有 ≥ m 的 Clique，對應的 SAT 就有解。

如果 Graph 有 ≥ m的 Clique,

那麼每個 Column 都會有一個 Clique 的 Vertex (因為同一個 Column 不會有 Edges)。

我們只要將 CNF 中，這些 Vertex 所對應的變數，Assign 為 True, CNF 就會是 True，

也就是 Satisfiable。(因為互補的變數之間不會有 Edges, 因此不會發生互相矛盾的情形)



Clique

{x, z, y} ⇒ x=1, z=1, y=1

$\Rightarrow E = (1 \vee 1 \vee 0) \wedge (0 \vee 0 \vee 1) \wedge (1 \vee 0)$

{y, x', z'} ⇒ y=1, x=0, z=0 ⇒ E=1

{y, z, y} ⇒ y=1, z=1 ⇒ E=1

{z', x', z'} ⇒ z=0, x=0 ⇒ E=1

Vertex Cover Problem

- Let G=(V, E) be an undirected graph

  A vertex cover C of G

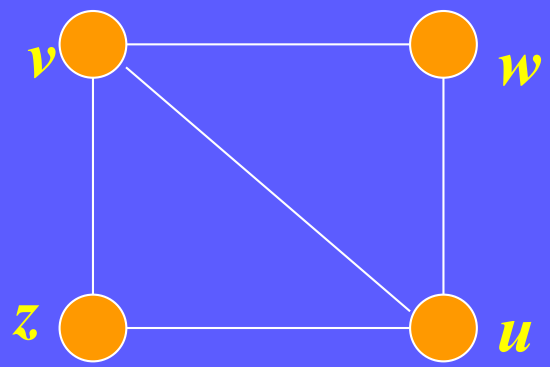
  = a set of vertices C such that ∀ edge in G is incident to at least one of vertices in C
- Example: {v, u} is a vertex cover

  {w, z} is not vertex cover, ∵ edge <v, u>

  {w, v, z} is a vertex cover

  {w, v, z, u} is a vertex cover

  {v, u} is the minimum vertex cover



Vertex Cover Problem

- Vertex cover problem
  - optimization problem:

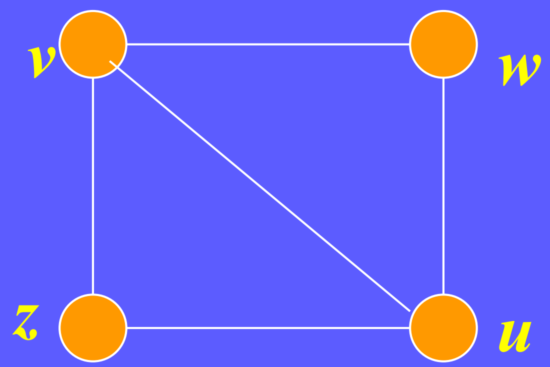
    given an undirected graph G,

    find the minimum vertex cover
  - decision problem

    given an undirected graph G and an integer k,

    determine whether *G* has a vertex cover containing ≤ *k* vertices
    - e.g. whether *G* has a vertex cover containing ≤ *3* vertices

      Yes, {w, v, x}, {v, u}



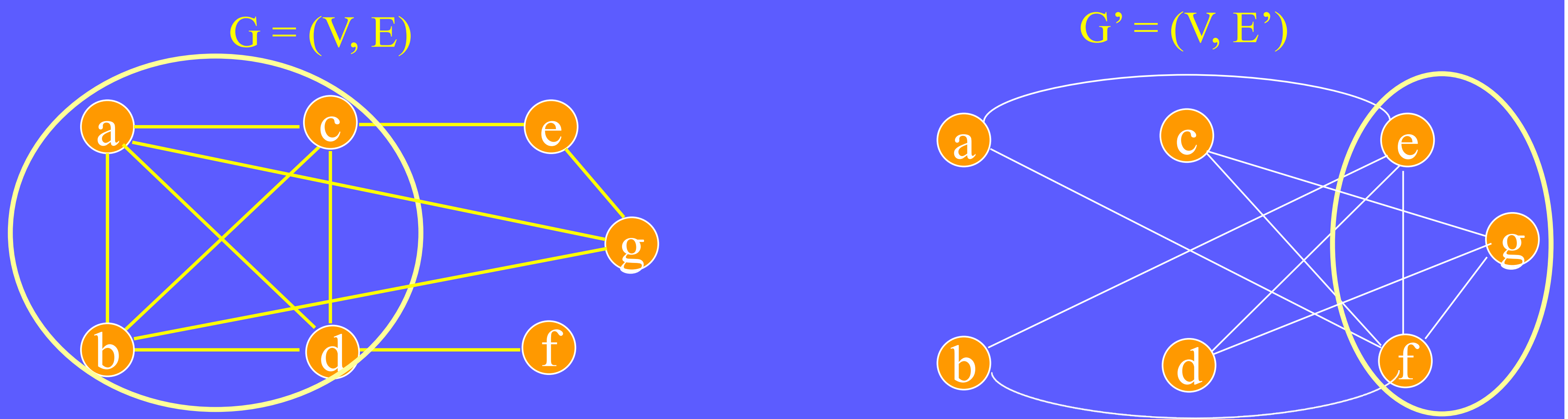
Proof of NP-Completeness of Vertex Cover Problem

- **vertex cover is NP**

  ∵ guess a cover of size <= k &

  check it in polynomial time

- **vertex cover is NP-hard**
  - **reduce clique to vertex cover in polynomial time**
  - **construct a complement graph G'=(V, E')**
  - **if G has a clique of size k, G' has a vertex cover of size n-k**
  - **if G' has a vertex cover k, G has a clique of size n-k**

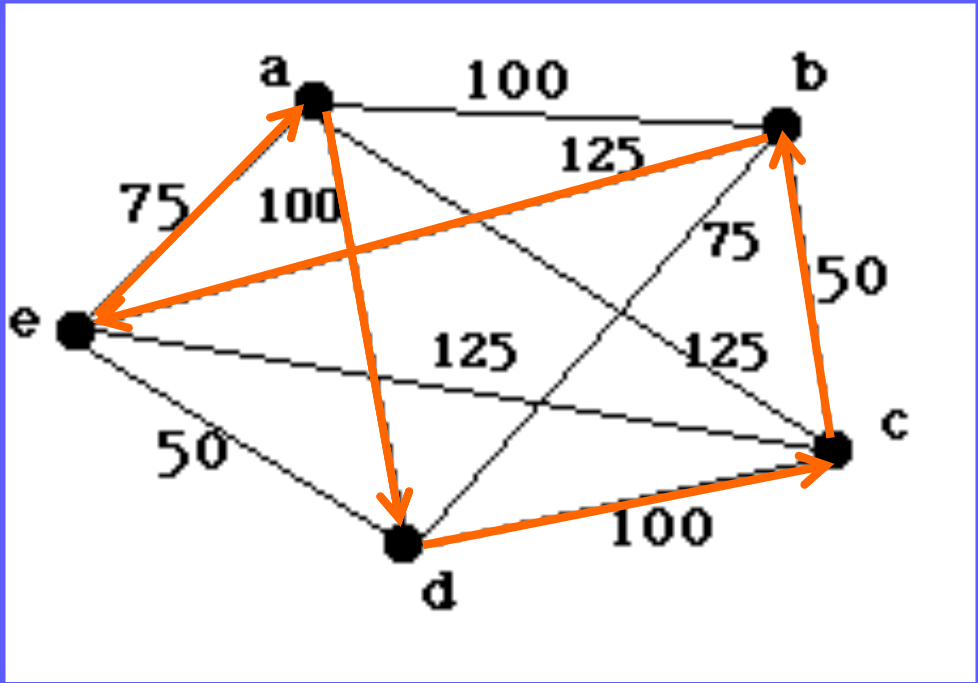


More NP-Complete Problems

- Hamiltonian cycle
  - a simple cycle that contains each vertex exactly once
  - determine whether a given graph contains a Hamiltonian cycle
- Traveling salesman
  - traveling salesman tour is a Hamiltonian cycle in a weighted complete graph
  - shortest path of traveling tour
- Hamiltonian path
- Independent set
- 3-dimensional matching
- Partition: partition into two subsets with the same size
- Knapsack
- Bin packing

Traveling Salesman Problem

- **Given a weighted directed graph, to determine a tour with minimum total weight on its edges**
  - **tour: a path that starts at one vertex, ends at that vertex & visits all the other vertices exactly once.**



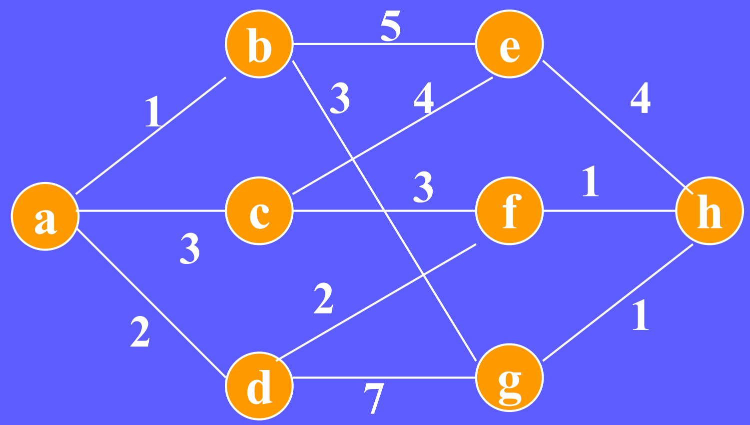
Techniques to Deal with NP-Complete Problems

Techniques for Dealing with NP-complete Problems

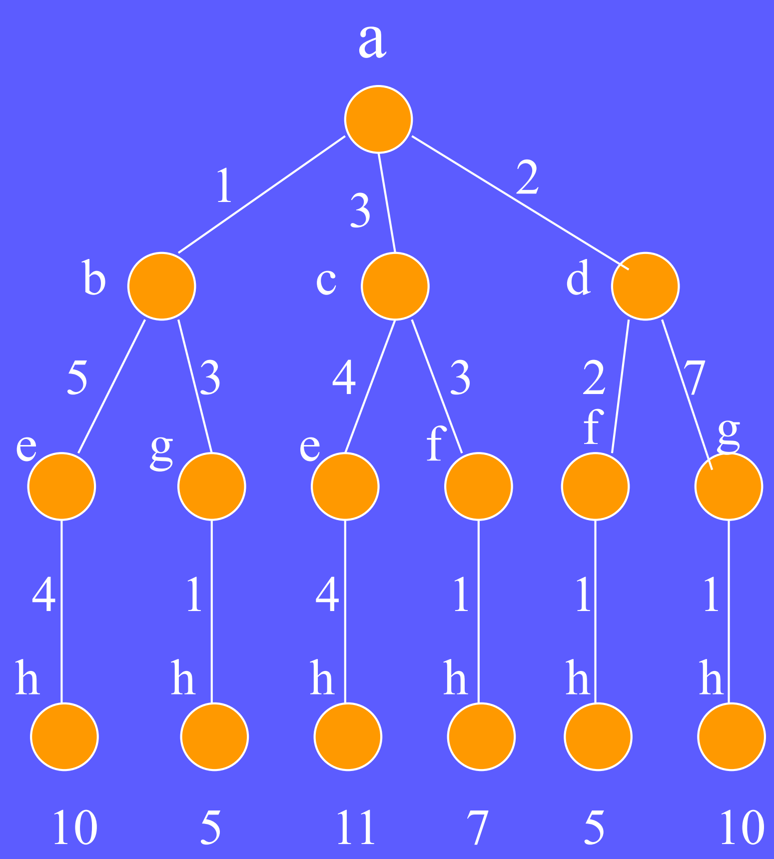
- **It seems that NP-complete problem cannot be solved precisely & completely with polynomial time algorithm**
- **Techniques to deal with NP-complete problem**
  - **approximation algorithm: not lead to optimal (precise) solution**
  - **allow algorithm for some special inputs**
    - e.g. vertex cover is NP-complete, but can be solved in polynomial time for bipartite graphs.
  - **algorithms whose running time is exponential, but work well for small inputs**
    - Backtracking
    - Branch and bound

Branch and Bound

- **Solution space represented as a tree for multi-stage shortest path problem**



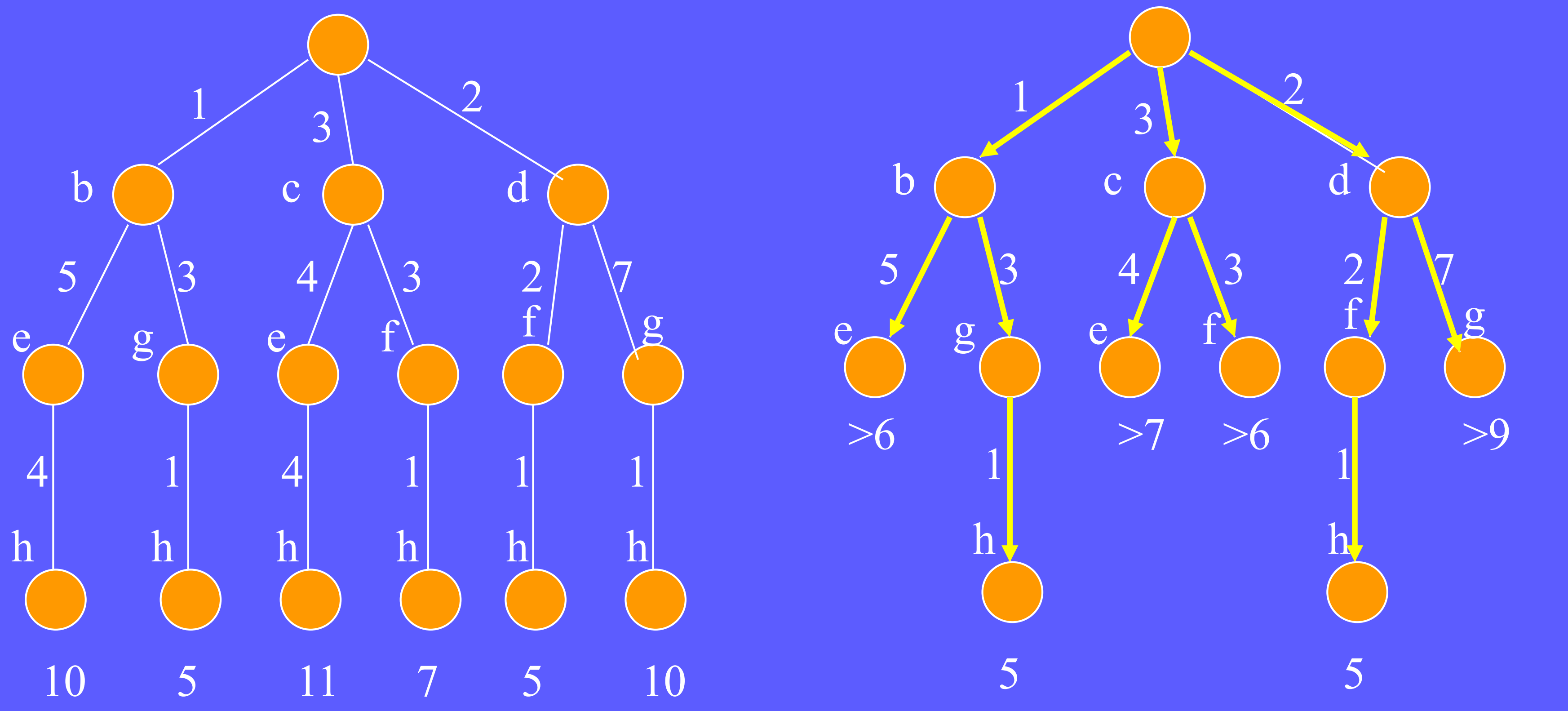
Shortest path?



Tree representation of Solution space

Branch and Bound

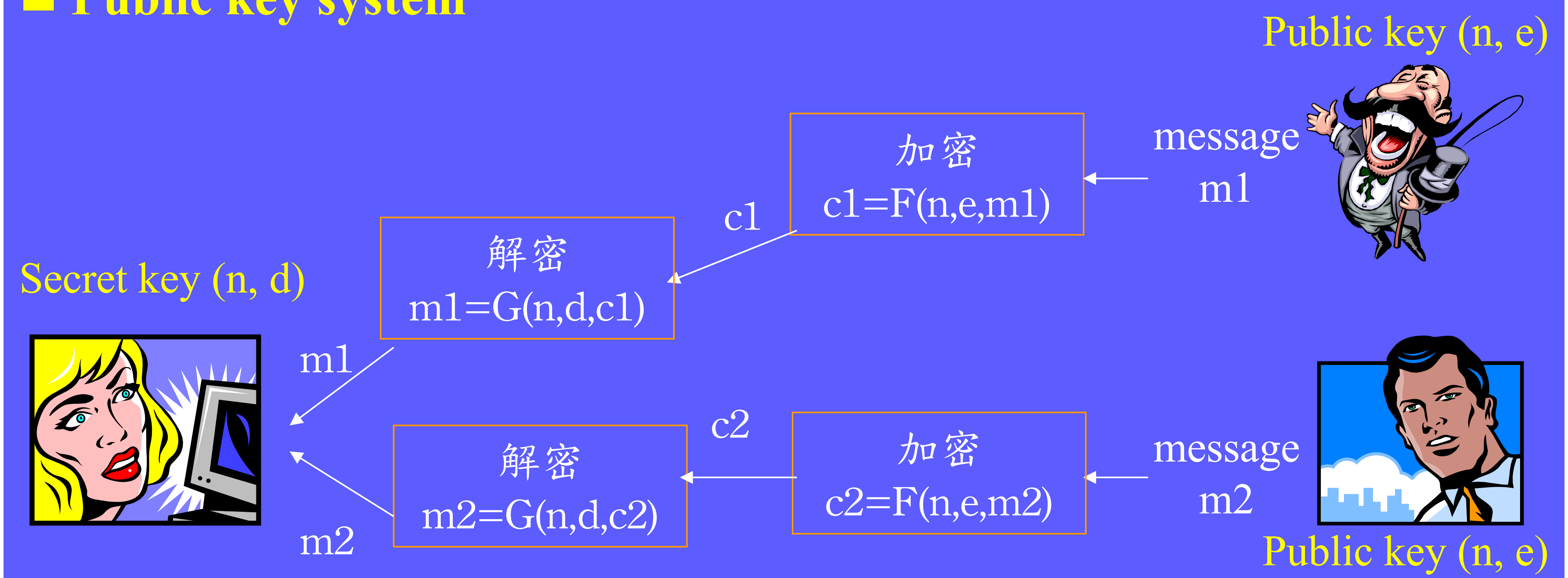
- bound the search space to avoid exhaustive searching solution space



Applications of NP-Complete Problems

Application of NP-Completeness

- Security
  - 沒有永遠無法破解的密碼，但有要花很多時間才能破解的密碼
  - 很多時間才能破解 ⇒ 破解需exponential time
  - RSA利用質因數分解是NP-complete的特性
- Public key system



RSA

- **Invented by Rivest, Shamir, & Adleman at MIT in 1978**
  - ☐ **choose two prime number, *p*, *q***
  - ☐ **compute n=p*q, z=(p-1)*(q-1)**
  - ☐ **choose a number *d* relative prime(互質) to *z***
  - ☐ **find *e* such that (*e***d*) mod *z* =1**
  - ☐ **加密 , given message *m*, cipher message $c=m^e$ mod *n***
  - ☐ **解密, given cipher message *c*, decipher message $m=c^d$ mod *n***
  - ☐ *n* & *e*公開，但是*d*不公開，解密需要知道*d*, 由*n*倒求出*d*需要exponential time （因為質因數分解是NP-complete）
- ***Example***
  - ☐ **choose two prime number, 5, 7**
  - ☐ **compute n=5*7=35, z=(5-1)*(7-1)=24**
  - ☐ **choose a number *d* =11 relative prime(互質) to 24**
  - ☐ **find *e*=11 such that (*e***d*) mod *z* =1**
  - ☐ **加密 , given message *m*=2, cipher message $c=2^{11}$ mod 35= 18**
  - ☐ **解密, given cipher message *c*, decipher message $m=18^{11}$ mod 35 = 2**

Conclusions

- Problem difficulty
- Computer Model: Turing Machine
  - Non-deterministic Turing Machine
- Problem
  - optimization problem
  - decision problem
- Deterministic vs. Non-deterministic Algorithm
- P, NP, NP-Hard, NP-Complete, Undecisible Problems
- P = NP ?
- Proof of NP-Complete by Reduction