Algorithms

授課老師：張景堯

Introduction to NP-Completeness

Chapter.11-Intractable Problems and Approximation Algorithms

Reporting to the Boss

- Suppose you fail to find a fast algorithm. What can you tell your boss? (當找不到快的演算法時，你怎麼說明?)
  - “I guess I’m too dumb. . . ” (dangerous confession)
  - “There is no fast algorithm!” (lower bound proof)
  - “I can’t solve it, but no one else in the world can, either. . . ” (NP-completeness reduction)

The Theory of NP-Completeness

- Several times this semester we have encountered problems for which we couldn't find efficient algorithms, such as the traveling salesman problem.
- We also couldn't prove exponential-time lower bounds for these problems.
- The theory of NP-completeness, developed by Stephen Cook and Richard Karp, provides the tools to show that all of these problems were really the same problem.

(碰到困難的問題找不到夠快的演算法，也證明不了他的lower bound一定是指數型時間複雜度，但我們現在能證明他跟其他NP-C是同一類問題。)

The Main Idea

- Suppose I gave you the following algorithm to solve the *bandersnatch* (just example is not real) problem:

「這只佔了我百分之一歲月的旅途，改變了我。」
「人類生命如此短暫，我為什麼沒試著多了解他一點？」

Bandersnatch(G)

1. Convert G to an instance of the Bo-billy problem Y.
2. Call the subroutine Bo-billy on Y to solve this instance.
3. Return the answer of Bo-billy(Y) as the answer to G.

「沿著當初走過的旅途，縱使你的身影已經不在，我相信一定能找到你留下的痕跡。」

- Such a translation from instances of one type of problem to instances of another type such that answers are preserved is called a *reduction*(歸約).

What Does this Imply?

- Now suppose my reduction translates G to Y in $O(P(n))$:

1. If my Bo-billy subroutine ran in $O(P'(n))$ I can solve the Bandersnatch problem in $O(P(n) + P'(n))$
   (Bandersnatch可用Bo-billy解，但不意味找不到更好解法)
2. If I know that $\Omega(P'(n))$ is a lower-bound to compute Bandersnatch, then $\Omega(P'(n) - P(n))$ must be a lower-bound to compute Bo-billy.
   (若Bandersnatch已證實沒有再快的解法了，即知道其lower-bound時間下限，意味Bo-billy也不會有更快解法)

- The second argument is the idea we use to prove problems hard!

What is a Problem?

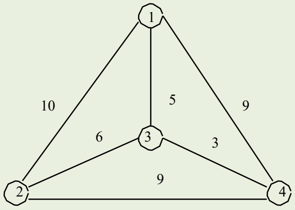
- A problem is a general question, with parameters for the input and conditions on what is a satisfactory answer or solution.
- Example: The Traveling Salesman (銷售員旅行問題)
- Problem: Given a weighted graph G, what tour $\{v_1, v_2, \dots, v_n\}$ minimizes

$$\sum_{i=1}^{n-1} d[v_i, v_{i+1}] + d[v_n, v_1] .$$

What is an Instance?

- An instance is a problem with the input parameters specified.

TSP instance: $d[v_1, v_2] = 10$, $d[v_1, v_3] = 5$, $d[v_1, v_4] = 9$, $d[v_2, v_3] = 6$, $d[v_2, v_4] = 9$, $d[v_3, v_4] = 3$



- Solution: $\{v_1, v_2, v_4, v_3, v_1\}$ cost= 27

Decision Problems (決策問題)

- A problem with answers restricted to ***yes*** and ***no*** is called a *decision problem*.
- Most interesting optimization problems can be phrased as decision problems which capture the essence of the computation.
- For convenience, from now on we will talk *only* about decision problems.

The Traveling Salesman Decision Problem

- Given a weighted graph G and integer k, does there exist a traveling salesman tour with cost $\leq k$?

(如：是否存在一個旅行推銷員巡迴路徑長度等於或小於27)

- Can we use the decision version of the problem to find the optimal TSP solution? How?
(怎麼用決策型問題即只回答**是**或**否**來找出TSP數值解k即上例的27? 甚至是整條巡迴路徑)

About Reduction

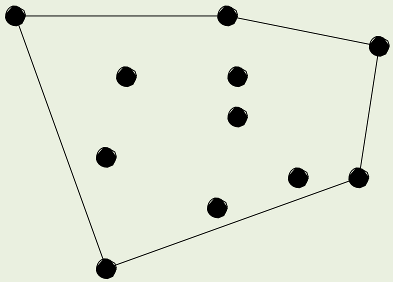
Chapter.11-Intractable Problems and Approximation Algorithms

Reductions (歸約)

- Reducing (transforming) one algorithm problem A to another problem B is an argument that <u>if you can figure out how to solve B then you can solve A</u>.
- Denote as $A \leq_m B$ or $A \propto B$. When this is true, solving A cannot be harder than solving B.
(求解A並不會比求解B更困難)
- We showed that many algorithm problems are reducible to sorting (e.g. element uniqueness, mode, etc.).
- Story: A computer scientist and an engineer wanted some tea. . .

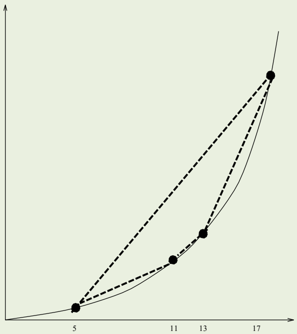
Convex Hull(凸包) and Sorting

- A nice example of a reduction goes from sorting numbers to the convex hull problem:



- We must translate each number to a point. We can map x to (x, x^2).

Why the Parabola(抛物線)?



- Each integer is mapped to a point on the parabola $y = x^2$. Since this parabola is convex, every point is on the convex hull. Further since neighboring points on the convex hull have neighboring x values, the convex hull returns the points sorted by x-coordinate, i.e. the original numbers.

Sorting to Convex Hull Reduction

```
Sort(S)
    For each i ∈ S, create point (i, i^2).
    Call subroutine convex-hull on this point set.
    From the leftmost point in the hull,
        read off the points from left to right.
```

- Recall the sorting lower bound of $\Omega(n \lg n)$. If we could do convex hull in better than $n \lg n$, we could sort faster than $\Omega(n \lg n)$ – which violates our lower bound. (如果convex hull能快過$n \lg n$，表示sort的lower bound就不應該只是$n \lg n$)
- Thus convex hull must take $\Omega(n \lg n)$ as well!!!
- Observe that any $O(n \lg n)$ convex hull algorithm also gives us a complicated but correct $O(n \lg n)$ sorting algorithm as well.

Satisfiability

Chapter.11-Intractable Problems and Approximation Algorithms

Satisfiability (滿足性)

■Consider the following logic problem:

Input: A set V of variables and a set of clauses C over V.

Problem: Does there exist a satisfying **truth** assignment for C?

■Example 1: $V = v_1, v_2$ and $C = \{\{v_1, \overline{v_2}\}, \{\overline{v_1}, v_2\}\}$
A clause is satisfied when at least one literal in it is *true*. C is satisfied when $v_1 = v_2 = true$.

■Clause is a logical expression. $C = (v_1 \; or \; \overline{v_2}) \; and \; (\overline{v_1} \; or \; v_2)$

(Clause裡面是用 or 運算，Clauses間是 and 運算)

Not Satisfiable (無法滿足的例子)

- Example 2: $V = v_1, v_2$ and $C = \{\{v_1, v_2\}, \{v_1, \overline{v_2}\}, \{\overline{v_1}\}\}$
- Although you try, and you try, and you try and you try, you can get no satisfaction.
- There is no satisfying assignment since v_1 must be false (third clause), so v_2 must be false (second clause), but then the first clause is unsatisfiable!

Satisfiability is Hard

- Satisfiability is known/assumed to be a hard problem.
- Every top-notch algorithm expert in the world has tried and failed to come up with a fast algorithm to test whether a given set of clauses is satisfiable.
- Further, many strange and impossible-to-believe things have been shown to be true if someone in fact did find a fast satisfiability algorithm.

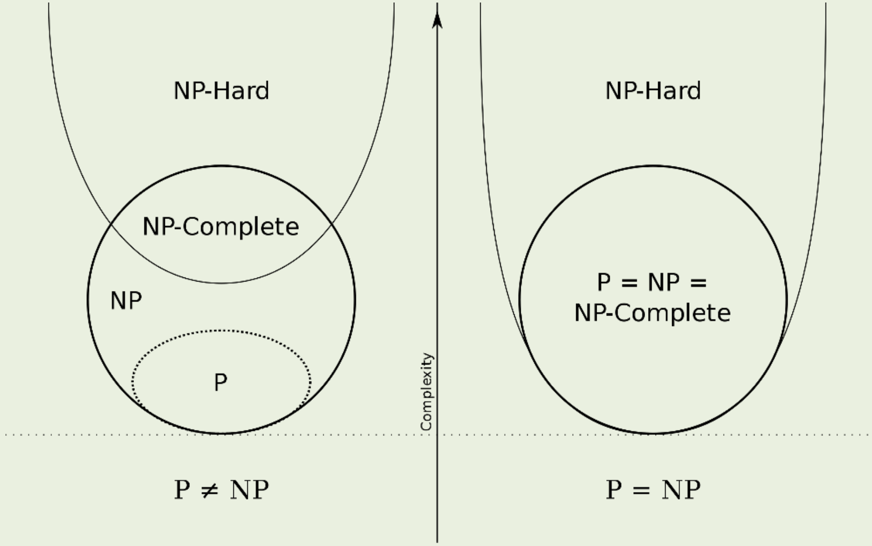
(如果有人真的找到了能快速解出滿足性問題的演算法，意味著TSP甚至一堆難解問題也就都有快速方法解決)

P, NP, NP-hard

- P: A set of decision problem whose worst case can be solved by deterministic algorithm in polynomial time.
(能在多項式時間內用確定性演算法解決)
- NP: (Non-deterministic, Polynomial time) A set of decision problem whose non-deterministic answer can be verified in polynomial time.
(以非確定性方式提出解答後能在多項式時間內驗證)
  - $P \subseteq NP$ (✓)
  - $P = NP$ (?, **Unsolved problem** in computer science)
- NP-hard: Class of decision problems which are at least as hard as the hardest problems in NP. Problems that are NP-hard do not have to be elements of NP, such as halting problem.

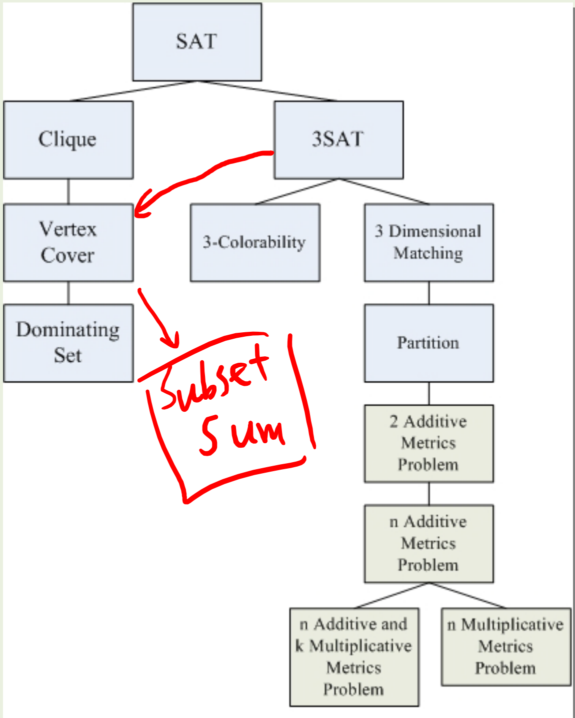
NP-complete

- A decision problem C is NP-complete if:
  - C is in NP, and
  - Every problem in NP is reducible to C in polynomial time.



Satisfiability is NP-Complete

- Cook's Theorem states that the Boolean satisfiability problem is NP-complete. (證明了所有的指令敘述都可以轉換成滿足性問題的格式)
- It was the first NPC problem has been proven in the world.
- So called the mother of all NP-complete problems.



Excerpt from "Introduction to Algorithms, UDI MANBER" Figure 11.1

3-Satisfiability

Chapter.11-Intractable Problems and Approximation Algorithms

3-Satisfiability (3元可滿足性)

- Instance: A collection of clause C where each clause contains *exactly 3* literals, boolean variable v.
- Question: Is there a truth assignment to v so that each clause is satisfied?
- Note that this is a more restricted problem than SAT. If 3-SAT is NP-complete, it implies SAT is NP-complete but not visa-versa, perhaps long clauses are what makes SAT difficult?!
- After all, 1-Sat is trivial!

3-SAT is NP-Complete

- First, we prove $Q \in NP$ (can guess an answer, and check it in polynomial time) (先證明屬於NP)
- Then, we prove it is complete, we give a reduction from $Sat \propto 3 - Sat$. We will transform each clause independently based on its *length*. (看看能否從SAT歸約到3-SAT)
- Suppose the clause C_i contains k literals.
- If $k = 1$, meaning $C_i = \{z_1\}$, create **two new** variables v_1, v_2 and four new 3-literal clauses: $\{v_1, v_2, z_1\}, \{v_1, \overline{v_2}, z_1\}, \{\overline{v_1}, v_2, z_1\}, \{\overline{v_1}, \overline{v_2}, z_1\}$.
- Note that the only way all four of these can be satisfied is if z is true.

3-SAT is NP-Complete (Cont.)

- If $k = 2$, meaning $C_i = \{z_1, z_2\}$, create **one new** variable v_1 and two new clauses: $\{v_1, z_1, z_2\}, \{\overline{v_1}, z_1, z_2\}$
- If $k = 3$, meaning $\{z_1, z_2, z_3\}$, copy into the 3-SAT instance as it is.
- If $k > 3$, meaning $\{z_1, z_2, \dots, z_n\}$, create $n - 3$ new variables and $n - 2$ new clauses in a **chain**:

$$\{z_1, z_2, v_1\}, \{\overline{v_1}, z_3, v_2\}, \{\overline{v_2}, z_4, v_3\}, \dots,$$
$$\{\overline{v_{n-4}}, z_{n-2}, v_{n-3}\}, \{\overline{v_{n-3}}, z_{n-1}, z_n\}$$

Why does the Chain Work?

$$\{z_1, z_2, v_1\}, \{\overline{v_1}, z_3, v_2\}, \{\overline{v_2}, z_4, v_3\}, \ldots,$$
$$\{\overline{v_{n-4}}, z_{n-2}, v_{n-3}\}, \{\overline{v_{n-3}}, z_{n-1}, z_n\}$$

- If none of the original variables in a clause are true, there is no way to satisfy all of them using the additional variable:

$$(F, F, T), (F, F, T), \ldots, (\boldsymbol{F}, \boldsymbol{F}, \boldsymbol{F})$$

- But if any literal is true, we have $n-3$ free variables and $n-3$ remaining 3-clauses, so we can satisfy all clauses.

$$(F, F, \boldsymbol{T}), (F, F, \boldsymbol{T}), \ldots, (F, \boldsymbol{T}, F), \ldots, (\boldsymbol{T}, F, F), (\boldsymbol{T}, F, F)$$

(只要有一個原變數是true，新增的$n-3$個變數只要分別指派一個true在剩下$n-3$ clauses裡就行，即便其他原變數皆不為true)

- Any SAT solution will also satisfy the 3-SAT instance and any 3-SAT solution sets variables giving a SAT solution, so the problems are **equivalent**.

SAT ∝ 3-SAT example

AN INSTANCE E IN SAT :

$\{\{z_1, z_2\}, \{\overline{z_3}\}, \{z_1, \overline{z_2}, z_3, \overline{z_4}, z_5, z_6\}\}$

THE INSTANCE E′ IN 3-SAT :

$\{z_1, z_2\} \dashrightarrow \begin{cases} \{z_1, z_2, v_1\} \\ \{z_1, z_2, \overline{v_1}\} \end{cases}$

$\{\overline{z_3}\} \dashrightarrow \begin{cases} \{\overline{z_3}, v_2, v_3\} \\ \{\overline{z_3}, v_2, \overline{v_3}\} \\ \{\overline{z_3}, \overline{v_2}, v_3\} \\ \{\overline{z_3}, \overline{v_2}, \overline{v_3}\} \end{cases}$

$\{z_1, \overline{z_2}, z_3, \overline{z_4}, z_5, z_6\} \dashrightarrow \begin{cases} \{z_1, \overline{z_2}, v_4\} \\ \{\overline{v_4}, z_3, v_5\} \\ \{\overline{v_5}, \overline{z_4}, v_6\} \\ \{\overline{v_6}, z_5, z_6\} \end{cases}$

SAT E

f(x)

if and only if

3-SAT E′

補充說明

■証明 "有一組解可使SAT問題的布林函數 E 為True ↔ 有一組解可使 3-SAT 問題的布林函數 E′ 為True"

① E is satisfiable ⇒ E′ is satisfiable

E is True
- 若要讓一個SAT問題的布林函數 E 為True，則每一個括號均要為True。
- 若要讓一個括號為True，則此括號中至少要有一個變數為True。

⇩

E' is True
- 若有一組可讓原SAT問題之布林函數 E 為True的解，也能使轉換後的3-SAT問題之布林函數 E′ 為True，則該組解能讓 E′ 中的每一個括號均能夠為True。
- 將原本在SAT問題內為True之 z_i 變數所在的括號內的 v_i 變數設成False；原本在SAT問題內為False之變數所在的括號內的 $\overline{v_i}$ 變數設成True，則此組解可使 E′為True

補充說明

② E' is satisfiable ⇒ E is satisfiable

E' is True
- ◘若要讓一個3-SAT問題的布林函數 E′ 為True，則每一個括號均要為True。
- ◘若要讓一個括號為True，則去除掉 v_i 變數不看，此括號中至少要有一個 x_i 變數為True。

⇩

E is True
- ◘若有一組可讓原3-SAT問題之布林函數 E′ 為True的解，也必能使SAT問題之布林函數 E 為True。

■由 ① 與 ② 得證：“有一組解可使SAT問題的布林函數 E 滿足 ↔ 有一組解可使 3-SAT 問題的布林函數 E′ 滿足”

4-Sat and 2-Sat

- A slight modification to this construction would prove 4-SAT, or 5-SAT,... also NP-complete.
- However, it breaks down when we try to use it for 2-SAT, since there is no way to stuff anything into the chain of clauses.

The Power of 3-SAT

- Now that we have shown 3-SAT is NP-complete, we may use it for further reductions. Since the set of 3-SAT instances is smaller and more regular than the SAT instances, it will be easier to use 3-SAT for future reductions.
(3元可滿足性問題比較固定不會太發散因此比SAT好做歸約)
- Remember the direction of the reduction!

$$\mathrm{Sat} \propto 3 - \mathrm{Sat} \propto X$$

A Perpetual Point of Confusion

- Note carefully the direction of the reduction.
(歸約方向很重要，常常容易弄反)
- We must transform *every* instance of a known NP-complete problem to an instance of the problem X we are interested in.

$$\text{Known NP} - \text{Complete} \ \propto \ X$$

- If we do the reduction the other way, all we get is a slow way to solve X, by using a subroutine which probably will take exponential time.

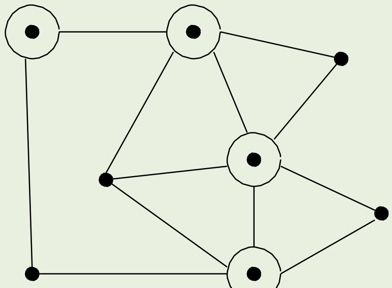
$$\cancel{X \ \propto \text{Known NP} - \text{Complete(Slow Algorithms)}}$$

Creative Reductions from SAT

Chapter.11-Intractable Problems and Approximation Algorithms

Vertex Cover 頂點涵蓋

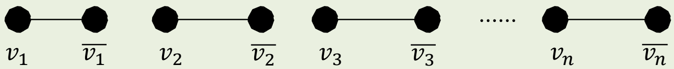
- Instance: A graph $G = (V, E)$, and integer $k \leq |V|$
- Question: Is there a subset of at most k vertices such that every $e \in E$ has at least one vertex in the subset? (這k個頂點的邊能涵蓋所有邊)



- Here, four of the eight vertices suffice to cover.
- It is easy to find a vertex cover of a graph: just take all the vertices. The hard part is to cover with *as **small** a set as* possible.

Vertex cover is NP-complete

- To prove completeness, we show reduce 3-SAT to VC. From a 3-SAT instance with n variables and c clauses, we construct a graph with $2n + 3c$ vertices.
- For each variable, we create two vertices connected by an edge:

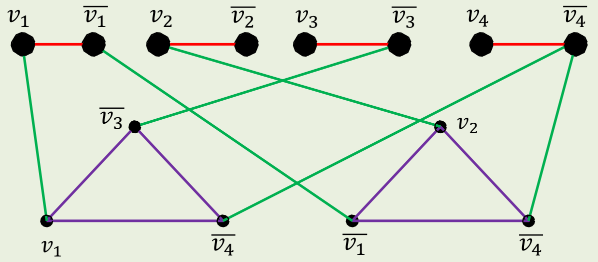


- To cover each of these edges, at least $\boldsymbol{n}$ vertices must be in the cover, one for each pair.

(這裡頭每一對不管選哪個vertices都能涵蓋所有邊)

Clause Gadgets (處理子句的歸約)

- For each clause, we create three new vertices, one for each literal in each clause. Connect these in a triangle.
- At least **two** vertices per triangle must be in the cover to take care of edges in the triangle, for a total of at least **2c** vertices.
- Finally, we will connect each literal in the flat structure to the corresponding vertices in the triangles which share the same literal.
- Ex: Reducing satisfiability instance $\{\{v_1, \overline{v_3}, \overline{v_4}\}, \{\overline{v_1}, v_2, \overline{v_4}\}\}$ to vertex cover.



Claim: G has a VC of size n + 2c iff S is Satisfiable

- Any cover of G must have at least $n + 2c$ vertices. To show that our reduction is correct, we must show that:
- Every satisfying truth assignment gives a cover.
- Select the n vertices corresponding to the true literals to be in the cover.
- Since it is a satisfying truth assignment, at least one of the three cross edges associated with each clause must already be covered - pick the other two vertices to complete the cover.■

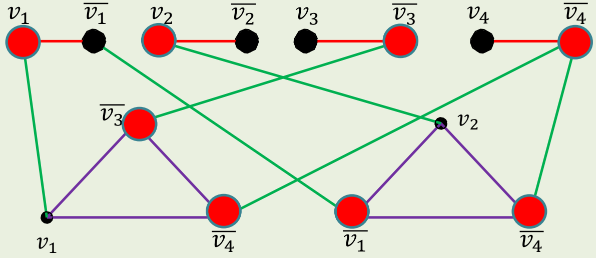
(每個子句再去挑其他兩個非主要true的頂點，就能涵蓋所有邊)

Every vertex cover gives a satisfying truth assignment

- Every vertex cover must contain n first stage vertices and $2c$ second stage vertices. Let the first stage vertices define the truth assignment.
- To give the cover, at least one cross-edge must be covered, so the truth assignment satisfies.

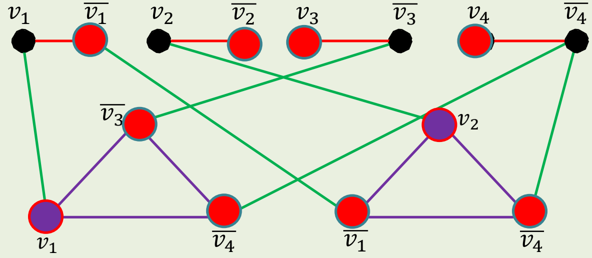
Example Reduction

- Every 3-SAT defines a vertex cover, and every cover truth values for the 3-SAT!
- Example: $v_1 = v_2 = true, \;\; v_3 = v_4 = false$. for $\{\{v_1, \overline{v_3}, \overline{v_4}\}, \{\overline{v_1}, v_2, \overline{v_4}\}\}$
- $n + 2C = 4 + 2 * 2 = 8$ vertices cover the graph



Try Another Instance

- Example: $v_1 = v_2 = false, \;\; v_3 = v_4 = true$. for $\{\{v_1, \overline{v_3}, \overline{v_4}\}, \{\overline{v_1}, v_2, \overline{v_4}\}\}$
- you should use more than 8 vertices to cover the graph



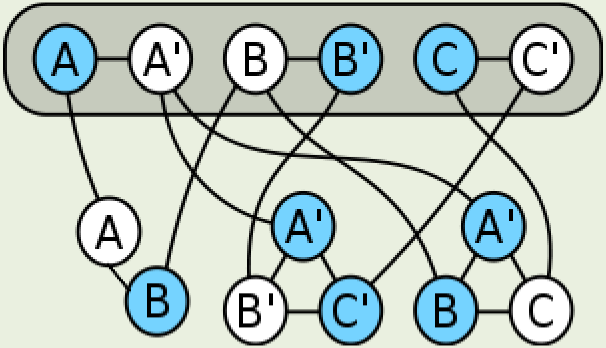
Starting from the Right Problem

- As you can see, the reductions can be very clever and very complicated. While theoretically any *NP-complete* problem can be reduced to any other one, choosing the correct one makes finding a reduction much easier.
(如果要做歸約，選對NP-C問題很重要，能讓歸約簡單點)

$$3 - Sat \propto VC$$

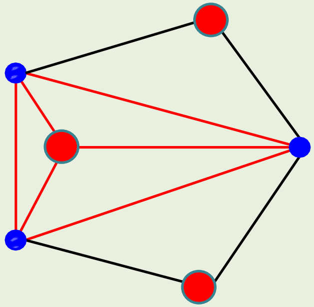
$Sat \propto VC$

- Every SAT defines a vertex cover, and every cover truth values for the SAT!
- Example from wiki : $A = C = true,\ B = false$. for $\left\{\{A, B\}, \{\bar{A}, \bar{B}, \bar{C}\}, \{\bar{A}, B, C\}\right\}$



Maximum Independent Set 最大獨立集合

- Instance: A graph $G = (V, E)$ and integer $j \leq v$.
- Question: Does the graph contain an independent of j vertices, i.e. is there a subset of v of size j such that no pair of vertices in the subset defines an edge of G? (在這頂點子集合裡各個頂點都不相連)



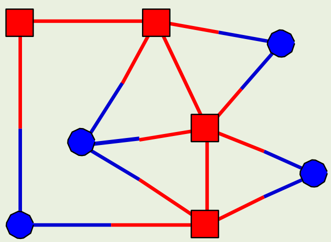
- Example: this graph contains an independent set of size 3. Recall that the movie star scheduling problem was a specific version of maximum independent set.

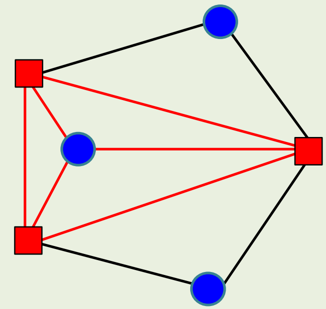
Proving Graph Problems Hard

- When talking about graph problems, it is most natural to work from a graph problem - the only NP-complete one we have is vertex cover!
- If you take a graph and find its vertex cover, the remaining vertices form an independent set, meaning there are <u>no edges between any two vertices in the independent set</u>.
- Why? If there were such an edge the rest of the vertices could not have been a vertex cover.

(把頂點涵蓋問題的解撇除掉這些頂點，是不是就得到最大獨立集合？這樣就能歸約給最大獨立集合求解有沒有等於$|V| - k$頂點數的答案，k是最小涵蓋頂點數)

Maximum Independent Set is NP-Complete





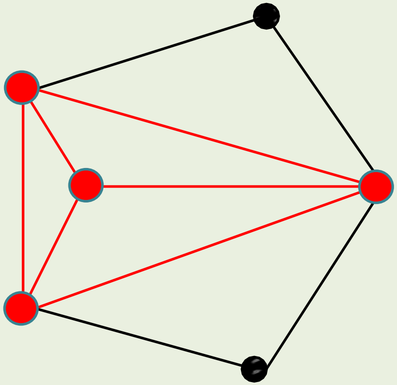
■ vertex in cover

● vertex in independent set

- The smallest vertex cover gives the biggest independent set, and so the problems are equivalent: **delete** the subset of vertices in one from V to get the other!
- Thus finding the maximum independent set is NP-complete!

Maximum Clique 最大集團

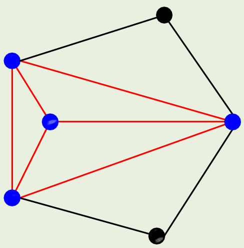
- Instance: A graph $G = (V, E)$ and integer $j \leq v$.
- Question: Does the graph contain a clique of j vertices, i.e. is there a subset of v of size j such that every pair of vertices in the subset defines an edge of G? (在這頂點子集合裡各個頂點都互相連接)



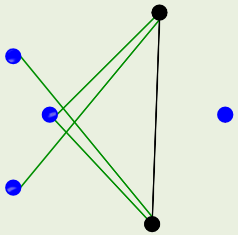
- Example: this graph contains a clique of size 4.

From Independent Set

- In an independent set, there are no edges between two vertices. In a clique, there are always a edge between two vertices. Thus if we complement a graph (have an edge iff there was no edge in the original graph), a clique becomes an independent set and an independent set becomes a clique!



Max Clique = 4
Max IS = 3

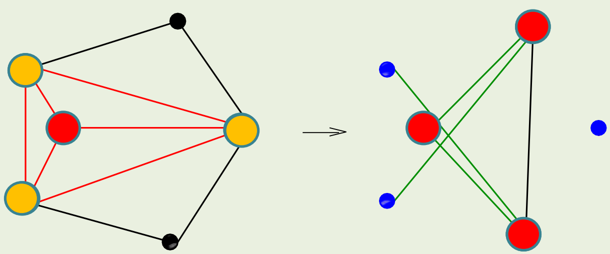


Max Clique = 3
Max IS = 4

(利用互補圖找出跟最大獨立集合的關聯性)

Punch Line (傑出的一手)

- Thus finding the largest clique is NP-complete:
  - If C is a clique in G, $V - C$ is a vertex cover in $\overline{G}$.
  - If VC is a vertex cover in G, then $V - VC$ is a clique in $\overline{G}$.



G $\qquad\qquad$ $\overline{G}$

Textbook Author's Most Profound Tweet

- An NP-completeness proof ensures that a dumb algorithm that is slow isn’t a slow algorithm that is dumb.

(NPC證明是證實一個看起來憨笨的演算法它只是慢而已，而不是一個明明有其他好解法卻還用又蠢又慢的演算法)

- Just because you use backtracking doesn’t mean your problem has no fast algorithm.

Integer Partition

Chapter.11-Intractable Problems and Approximation Algorithms

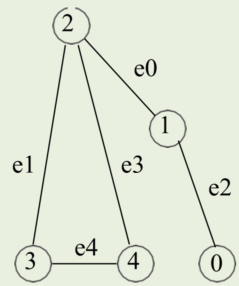
Integer Partition (Subset Sum)

- Instance: A set of integers S and a target integer T.
- Problem: Is there a subset of S which adds up exactly to T?
- Example: $S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$ and $T = 3754$
- Answer: $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = T$
- Observe that integer partition is a number problem, as opposed to the graph and logic problems we have seen to date.

Integer Partition is NP-complete

- To prove completeness, we show that vertex cover ∝ integer partition. We use a data structure called an incidence matrix(關聯矩陣) to represent the graph G.

	e4	e3	e2	e1	e0
v0	0	0	1	0	0
v1	0	0	1	0	1
v2	0	1	0	1	1
v3	1	0	0	1	0
v4	1	1	0	0	0



- How many **1**'s are there in each column? Exactly two. How many **1**'s in a row? Depends on the vertex degree.

Using the Incidence Matrix

- The reduction from vertex cover creates $|V| + |E|$ numbers from G.
  - Each "vertex" number will be a base-4 realization of the incidence matrix row, plus a high order digit: (4進位+1最高位元)

$$x_i = 4^{|E|} + \sum_{j=0}^{|E|-1} M[i,j] \times 4^j$$

    so $V_2 = 101011$ becomes $4^5 + (4^3 + 4^1 + 4^0) = 1093$.
  - Each edge(column) will also get a number: $y_i = 4^i$(也是4進位數).
  - The target integer will be

$$T = k \times 4^{|E|} + \sum_{j=0}^{|E|-1} 2 \times 4^j$$

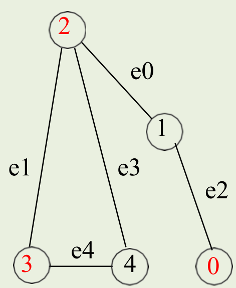
e0,e1,e2,e3,e4=1, 4, 16, 64, 256

v0,v1,v2,v3,v4=1040(100100), 1041(100101), 1093(101011), 1284(110010), 1344(111000)

How?

- Each column (digit) represents an edge. We want a subset of vertices which covers each edge.
- We can only use $k \times$ "$\textit{vertex numbers}$", because of the high order digit of the target, here $T = 322222 = 3754$

	e4	e3	e2	e1	e0
v0	0	0	1	0	0
v1	0	0	1	0	1
v2	0	1	0	1	1
v3	1	0	0	1	0
v4	1	1	0	0	0



v_0	100100	1040
v_2	101011	1093
v_3	110010	1284
e_0	000001	1
e_2	000100	16
e_3	001000	64
e_4	010000	256
T	322222	3754

$S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$ and $T = 3754$

Why?

- Because there are at exactly two 1's per column, no sum of them can carry over to the next column (in base-4).

(每欄最多就2個1，用四進位就沒有進位問題)

- In any vertex cover, edge e_i can be covered either once or twice, but with the option of adding number y_i we can always cover it twice without adding vertex numbers.

(任何頂點涵蓋的解，每條edge必定會在關聯矩陣出現一或兩次數值1，只須再針對出現一次的補上一個相應edge數字就能達成全部都是四進位2的值)

V C in G → Integer Partition in S

- Given k vertices covering G, pick the k corresponding "vertex numbers". Each edge in G is incident on one or two cover vertices. If it is one, includes the corresponding "edge number" to give two per column.

(要回答某圖是否有k個頂點能涵蓋所有edge，轉成整數分割問題後就是判斷是否能有k個頂點數值再搭配只被涵蓋1次的edge數值，能否達成一個目標整數值是$k22\ldots2$的四進位數)

Integer Partition in S → V C in G

- Any solution to S must contain exactly k "vertex numbers". The target in that digit is k, so not more, and because there are no carries not less.
- This subset of k "vertex numbers" must contain at least one edge e_i per column i. We can always pick up the second **1** to match the target using y_i.

- Neat, sweet, and NP-complete!

(當從集合S得到整數分割解後，意味著該解必含有k個頂點數值，因為該k值無法從其他位數進位獲得，要湊成目標數值，除了頂點數值外，還會有edge數值補足只被頂點涵蓋到一次的情況)

Exercises

Problems of the Day - TSP decision

- Q1: Suppose we are given a subroutine which can solve the traveling salesman decision problem in, say, linear time. Give an efficient algorithm to find the actual TSP tour by making a polynomial number of calls to this subroutine.

- Hint: upper bound, binary search, deal with useless edges

Problems of the Day (Cont.)

- Q2:Show that the *dense subgraph* problem is NP-complete:
- Input: A graph G, and integers k and y.
- Question: Does G contain a subgraph with exactly k vertices and at least y edges? (G能否找到一個子圖剛好有k個頂點並至少有y條邊)
- Hint: Clique

Problems of the Day (Cont.)

- Q3:Show that the *Hitting Set* problem is NP-complete:
- Input: A collection C of subsets of a set S, positive integer k.
- Question: Does S contain a subset S' such that $|S'| \leq k$ and each subset in C contains at least one element from S'?

(C是一群從集合S產生的子集群組，能否找到一個大小等於小於k的子集S'讓C群組的所有子集合內元素至少都有一個在S'裡)

(如：S-自助餐菜色、C-每個顧客餐盤、S'-人氣菜色有k種，每個顧客餐盤至少都會有一樣人氣菜色)

- Hint: Vertex Cover