投稿類別：資訊類

篇名：

數獨與NP問題的探討

作者:
羅煜翔。北市大同高中。高二 15 班。
洪佑銓。北市大同高中。高二 15 班。
劉宇宸。北市大同高中。高二 15 班。

指導老師：
北市大同高中　陳瑞宜老師
北市大同高中　張繼元老師

壹●前言

一、研究動機

自從2000年美國克雷數學研究所公佈了七道世界難題之後，除了龐加萊猜想問題，其他6題仍是人們目前尚未破解的領域。某位數學家曾說：「一個好的問題勝過十個好的解答！」因為，若正確答案被求出，則這道題目就會抵達終點，唯有不斷的摸索、探究，才能稍微的嗅出高深的題目中那一絲絲的微妙。因此，當我們看到P與NP問題定理的題目時，便產生了一股強烈的慾望，想去深入瞭解，所以我們磨刀霍霍的準備開始來研究了！

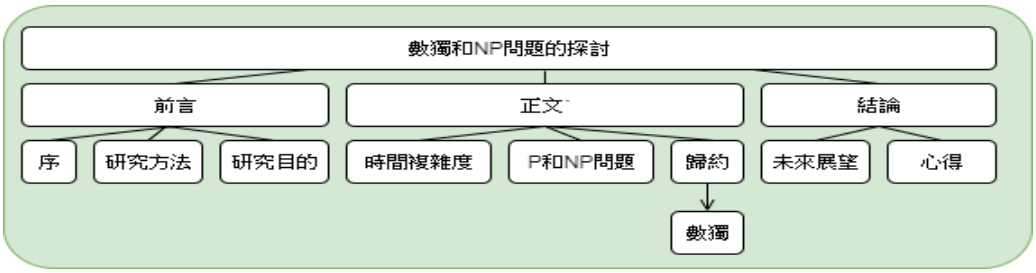
二、研究目標

(一)瞭解時間複雜度的定義。
(二)瞭解P、NP、NPC、NPH問題的定義。
(三)學習NPC問題和NPC問題的歸約。
(四)嘗試數獨問題歸約到SAT問題。

三、研究方法

我們在網路上搜尋關於時間複雜度、P、NP、NPC、NPH等文獻學習研讀(請參見引註資料的網站)，並且在$OpenDSA\ Data\ Structures\ and\ Algorithms\ Modules\ Collection$網站上學習大量相關的資訊／數學名詞定義和解釋，以明白NPC問題的歸約方式，並且請教資訊老師以及數學老師的意見，經小組成員多次討論後才完成本研究。

四、研究架構



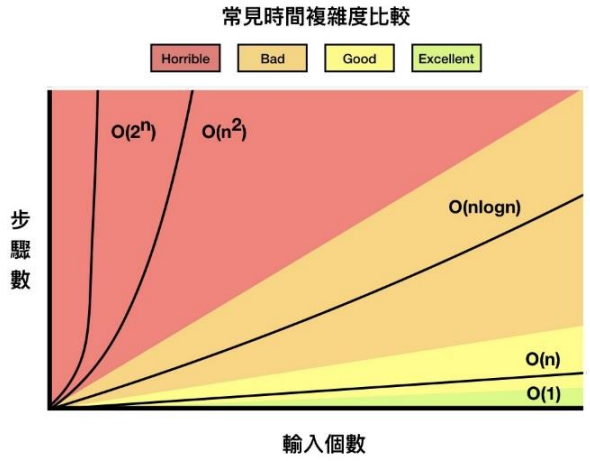
貳●正文

在瞭解P與NP問題之前，首先得先了解「時間複雜度」及「圖靈機」的假設。

一、時間複雜度與圖靈機

判斷演算法的好壞，最基本的兩個指標是：1.這個演算法有多快？2.他們用到的記憶體有

多少？而時間複雜度就是用來評斷演算法執行之快慢的指標。一般來說，時間複雜度用大O符號表示，它是描述一個演算法在輸入n個東西時，全部執行時間與n的關係。通常在設計一個演算法時，會希望能盡可能的縮減其時間複雜度。例如，當$n = 10000$時，$O(n^2)$所需步驟數將比$O(n)$步驟數多一萬倍，也因此處理大量資料或複雜運算時，好的演算法可以省下很多時間。在n相當大時，我們比較兩數大小時通常會先比較最高次方，因此，在記錄時間複雜度時，我們只記錄最高次方的那一項且盡可能化簡，譬如，若有一個程式需要$n^2 + n$個步驟，就會把它的時間複雜度簡記成$O(n^2)$ ，而且如果另外一個程式需要$4n^2 - n$個步驟時，仍然該時間複雜度也是記成$O(n^2)$。換句話說，大O符號可以視作是一個區段類別，不是一個精準的記錄單位，只要步驟數是 n的二次多項式就會全部歸類到$O(n^2)$。在電腦科學中，演算法($algorithm$)是一個計算的具體步驟，常用於計算、資料處理和自動推理，簡單來說，如果我輸入一個東西，想要得到另外一個東西(例如：輸入 1，想要得到 100)，中間的過程就是演算法。以下整理六種重要的演算法與時間複雜度比較表(請見表一)。此外，圖靈機($Turing\ Machine$)是現代電腦基礎的雛型，是英國數學家圖靈($Alan\ Turing$)在1936年所提出的一種抽象的計算機，這個計算機模型的猜想是「能夠計算所有在演算法中可以計算的問題」，又稱邱奇－圖靈問題($Church–Turing\ thesis$)，而這個猜想普遍是可讓人接受的想法。我們可以先設定圖靈機一個指令或符號，讓它的下一步動作只能有一種可能，讓圖靈機按照可以操作的規則做事情，符合這樣操做方式的圖靈機又稱作決定型圖靈機$Deterministic\ Turing\ Machine(DTM)$。理論上，圖靈機可以完成所有可以計算的題目，又可以用物理的方式製造出來，往後才有電腦的誕生。那，如果擁有一個圖靈機，是否可以解決所有可解的問題？答案是否定的，由上面的時間複雜度可知，每種的演算法有它個別的計算步驟(時間)，若已知該演算法的時間複雜度是$O(2^n)$，則當$n = 10000$時，2^{10000}也是個天文數字，即使每一步驟僅需要0.0000001秒，該問題也要超過地球年齡時間才能算完，因此，問題就算知道是可解的，也不代表可以解完，因為需要考慮解決問題的時間。所以，現實層面是只有一個數量級時間以下的問題才容易解決，這個數量級時間稱為多項式時間$polynomial\ time$，例如時間複雜度$O(1)$、$O(n)$、$O(n^2)$等等皆為多項式時間可解決之問題，如果超過多項式時間可解決的問題就是很困難的問題了！



【圖一】常用時間複雜度的比較
(圖一來源：初學者演算法—談甚麼是演算法和時間複雜度。2019 年 9 月 5 日，取自於： https://reurl.cc/qd8bmN)

【表一】時間複雜度比較表

時間複雜度	$O(1)$	$O(n)$	$O(n^2)$
代表性演算法	陣列讀取法	簡易搜尋法	泡沫排序法

<table>
<tr><td>程式碼</td><td>

```
using namespace std;
int main()
{
    int n,i,k;
    cin>>n;
    int x[n]={0};
    for(i=0;i<n;i++)
    {
        cin>>x[i];
    }
    cin>>k;
    cout<<x[k];
}
```

</td><td>

```
for(i=0;i<n;i++)
{
    if(k=x[i])
    cout<<i;
    break;
}
```

</td><td>

```
for(k=0;k<n;k++)
{
    for(i=0;i<n-1;i++)
    {
        if(x[i]>x[i+1])
        {
            j=x[i];
            x[i]=x[i+1];
            x[i+1]=j;
        }
    }
}
```

</td></tr>
<tr><td>時間複雜度</td><td>$O(logn)$</td><td>$O(n)$</td><td>$O(2^n)$</td></tr>
<tr><td>代表性演算法</td><td>二分搜尋法</td><td>費波納契數列法 1</td><td>費波納契數列法 2</td></tr>
<tr><td>程式碼</td><td>

```
a=0 b=n
while(1)
{
    mid=(a+b)/2;
    if(x[mid]>k)
    {
        b=mid;
    }
    if(x[mid]<k)
    {
        a=mid;
    }
    if(x[mid]=k)
    {
        cout<<mid;
        break;
    }
```

</td><td>

```
int main()
{
    int n;
    cin>>n;
    int a[n];
    a[0]=1;
    a[1]=1;
    for(int i=2;i<n;i++)
    {
        a[i]=a[i-1]+a[i-2];
    }
    cout<<a[n-1];
}
```

</td><td>

```
int F(int n){
    if(n==1|n==2)
        return 1;
    return F(n-1)+F(n-2);
}
int main()
{
    int n;
    cin>>n;
    cout<<F(n)<<endl;
}
```

</td></tr>
</table>

(表一來源：研究者研究自製程式碼)

二、P、NP和NPC問題

承上，我們接著思考，什麼是P問題，什麼是NP問題，什麼是NPC問題。

定義：

- P問題：若有一群演算法或問題，用DTM做計算，在多項式時間內解決，這一群演算法或問題被稱為P問題。
- NP問題：如果用演算法驗證一組解是否正確，用DTM做計算，在多項式時間內解決，這個問題就是NP問題。

由上定義，我們可以推得$P \subseteq NP$，即P問題為NP問題，但NP問題不一定為P問題。然而，「到底P問題有沒有可能等價於NP問題？」這個問題就是在西元2000年被數學家所公布的世界七大難題之一。該問題很重要的原因是，現在最普遍的加密方法為RSA加密，RSA的加密方法是將兩個很大的質數相乘，如果需要解鎖則要知道其中之一的質數才能整除，用暴力破解法會超過多項式時間相當困難，所以該加密方式是NP問題。我們認為$P \neq NP$，若有人可以找到證明NP問題是P問題的方式，那麼用現在的電腦便可以輕易破解RSA加密法，目前的加密方式大多也是NP問題，使得很多加密方式都會被破解，一定會對世界造成很大的影響！

然而，在數學家嘗試證明$P \subseteq NP$的過程，又帶出一個重要想法「$NP-Complete$問題」

定義：

- NPC問題：只要滿足以下兩個條件的問題，都稱$NP-Complete$問題(簡稱NPC)
  - ✓ 條件一：問題本身是NP問題
  - ✓ 條件二：其他所有屬於NP問題都可以在演算法，用DTM做計算，在多項式時間內「歸約」成這個問題

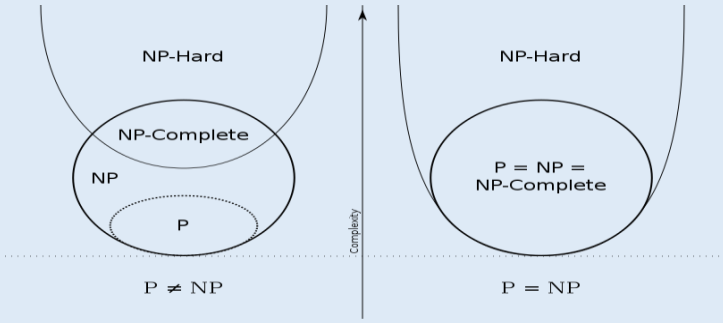
- 歸約($reduction$)：對每個問題L，總有一個多項式時間多對一變換，即一個決定性的算法可以將實例$l \subseteq L$轉化成實例$c \subseteq C$，並讓c回答Yes若且唯若l此答案也是Yes。歸約的符號一般寫作$A \leq_m B$，例如平方$\leq_m$乘法，但反之則否。通常$\leq$ 符號下標表示歸約類型，如m表對應縮小，p表多項式縮減。

為了證明某個NP問題Q是NPC問題，證明者需要自己去找到一個已知的NPC問題可以變換成Q。而在1971年，美國數學家$Stephen\ A.Cook$提出了$Cook-Levin$理論，該理論指出任何一個NP中的問題都可以在多項式時間內，將之歸約成「布林可滿足性問題(SAT)」，且SAT問題是全世界被嚴謹證明的第一個為NPC問題，這個想法非常漂亮，因為如果可以證明SAT問題是P問題，等於今天手中有任何一個NP問題，就把它在多項式時間內歸約成SAT問題，所以，所有的NP問題都變成了P問題，亦即$P = NP$。這個想法很重要，也就是NPC問題是解決$P = NP$。關鍵是，如果能夠證明NPC問題是或不是P問題，那麼重要的千禧年問題就破解了！另外有一類問題稱為$NP-Hard$問題，其定義如下：

定義：

- $NP-Hard$問題：其他所有屬於NP問題都可以在演算法，用DTM做計算，在多項式時間內「歸約」成這個問題，這個問題就是$NP-Hard$問題

從定義可知，$NP-Hard$問題包含了NPC問題，而$NP-Hard$問題本身可以不是NP問題。在我們對於所有名詞皆有定義後，我們可以圖二知道相對應的示意圖。在圖二左邊是假設$P \neq NP$的前提下，問題所分類的形式。舉例，「找中位數問題」是一個P問題，也是NP問題，但不是NPC問題。右圖是$P = NP$被證完後$P = NP = NPC$所分類的形式。



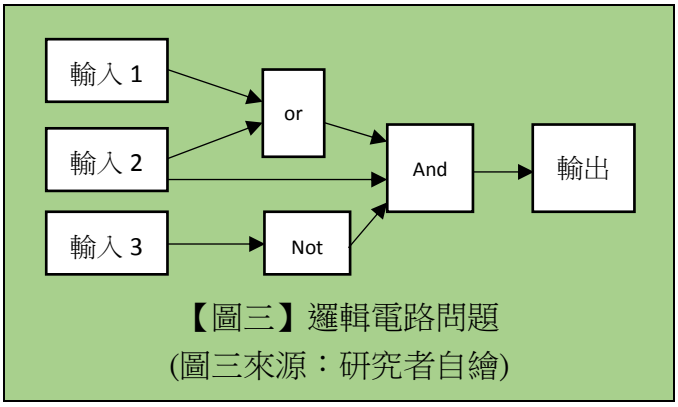
【圖二】P、NP、NPC、$NP-Hard$問題的示意圖
(圖二來源：輕鬆談演算法複雜度的分界—什麼是P,NP,NP-Complete,NP-Hard 問題。2019 年 9 月 8 日，取自於：https://reurl.cc/O1qakr

三、經典的NPC問題

在1972年，計算機理論家暨科學家*Richard Manning Karp*教授的論文「*Reducibility Among Combinatorial Problems*」中提出了21種經典的NPC問題，其牽涉的領域包涵了圖論與組合數學，以下我們舉例其中五個經典的NPC問題。

(一) SAT問題

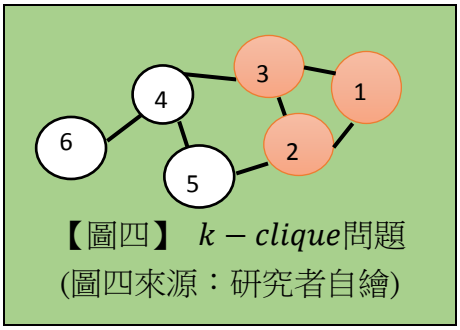
SAT問題是*Boolean Satisfiability*問題的簡寫，簡單的說就是判斷一組給定的布林函數，是否能夠找出一組變數的值使其為真。SAT問題是研究史中的第一個NPC問題，已經有許多的演算法跟技巧被發明來解決它。SAT問題的應用很廣泛，如邏輯電路問題，它是指：給定一個邏輯電路，經過中間的邏輯門，問是否存在一種輸入使輸出為$True$。如圖三，當輸入1、輸入2、輸入3分別為$True$、$True$、$False$或$False$、$True$、$False$時，則輸出為$True$。而$3-SAT$問題的定義是每一個布林函數表示法括弧內的文字就剛好只有三個，例如：$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee \overline{x_2} \vee \overline{x_1}) \wedge (x_3 \vee \overline{x_2} \vee \overline{x_1}) \wedge (x_3 \vee \overline{x_4} \vee x_1)$。



【圖三】邏輯電路問題
(圖三來源：研究者自繪)

(二)$k-clique$問題

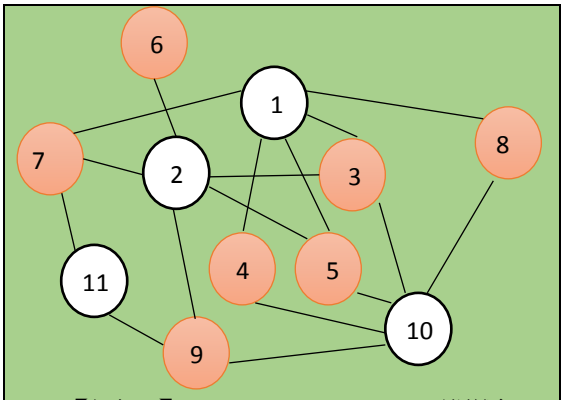
$k-clique$問題也稱分團問題，在圖論中是一個NPC問題。如圖四中的{1,2,3}的三個點，在其中任一個點與其他所有點相連接，就是大小為3的$clique$ (團)。分團問題是問一個圖中有沒有大小是k以上的團，任意挑出k個點，因為我們可以判斷出這k個點是否為一個團，所以這是NP問題。



【圖四】 $k-clique$問題
(圖四來源：研究者自繪)

(三) *Independent Set*問題

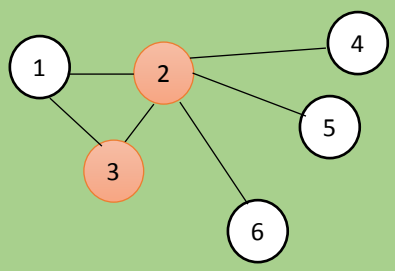
*Independent Set*問題也稱獨立集問題，是圖論中的一個概念，一個獨立集的意思是一個圖中兩兩不相鄰的頂點所成的集合。令這些點所構成的集合為S，S中的所有任兩個點沒有邊連接。最大獨立集(*Maximum Independent Set*)問題也是NPC問題，目前尚未找到一個有效率的演算法求圖中的最大獨立集。如圖五，{3,4,5,6,7,8,9}共七個點為該圖的一個獨立集。



【圖五】 *Independent Set*問題
(圖五來源：研究者自繪)

(四)$Vertex\ cover$問題

$Vertex\ cover$問題也稱點覆蓋問題，在一群點中，挑選數個點，碰觸到所有的邊，這些點就稱「點覆蓋」，可能有許多種，換句話說，每一條邊，都會觸碰到大於等於1個的選定點，點覆蓋就像紙鎮，壓住所有邊讓邊不被吹走，最少點覆蓋問題($Minimun\ Vertex\ Cover\ Problem$)也是經典的$NPC$問題。如圖六，{2,3}共兩個點為該圖的一個最少點覆蓋。一個有趣的小觀察，圖中沒被選到的點集合{1,4,5,6}就是一個獨立集。



【圖六】 $Vertex\ cover$問題
(圖片來源：研究者自繪)

四、NPC問題的相互歸約

(一) SAT問題歸約到$3-SAT$問題

考慮一組or的運算之變數個數有以下四種情形：

1. 1個變數的or

A：(x)

B：$(x \vee y_1 \vee y_2) \wedge (x \vee \overline{y_1} \vee y_2) \wedge (x \vee y_1 \vee \overline{y_2}) \wedge (x \vee \overline{y_1} \vee \overline{y_2})$，其中$y_1$、$y_2$為新增的變數。透過$x$、$y_1$、$y_2$三個變數共8種的真值表如表二可得知$A$、$B$會同時為$T$或同時為$F$。

【表二】真值表

1個變數的or					2個變數的 or				
x	y_1	y_2	A	B	x_1	x_2	y	A	B
T	T	T	T	T	T	T	T	T	T
T	T	F	T	T	T	T	F	T	T
T	F	T	T	T	T	F	T	T	T
T	F	F	T	T	T	F	F	T	T
F	T	T	F	F	F	T	T	T	T
F	T	F	F	F	F	T	F	T	T
F	F	T	F	F	F	F	T	F	F
F	F	F	F	F	F	F	F	F	F

(表格來源：研究者討論整理)

2. 2個變數的or

A：$(x_1 \vee x_2)$

B：$(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \bar{y})$，其中$y$為新增的變數。透過$x_1$、$x_2$、$y$三個變數共8種的真值表如表二檢驗可以得知$A$、$B$會同時為$T$或同時為$F$。

3. n個變數的or（$n \geq 4$）

A：$(x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n)$

B：$(x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee y_3) \wedge \ldots \wedge (\overline{y_{n-3}} \vee x_{n-1} \vee x_n)$，其中$y_1$、$y_2$、$\cdots$、$y_{n-3}$為新增的變數。分成以下四類情況討論：

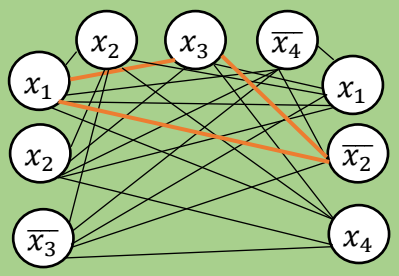
	情形	新增的變數y_i的配置	結果
第一類	當x_1、x_2其中之一為T。	設定y_1、y_2、$\cdots$、y_{n-3}全為F。	會導致每個括弧內都有T，即A、B會同時為T。
第二類	當x_{n-1}、x_n其中之一為T。	設定y_1、y_2、$\cdots$、y_{n-3}全為T。	會導致每個括弧內都有T，即A、B會同時為T。
第三類	當x_3、x_4、x_5 … 、x_{n-2}其中之一為T。此類的$n \geq 5$。 假若x_i為T，$i \in \{3,4,\ldots,n-2\}$	設定y_1、y_2、$\cdots y_{i-2}$皆為T；設定y_{i-1}、y_i、$\cdots$、y_{n-3}皆為F。	會導致第i個括弧前、第i個括弧、第i個括弧後每個括弧內都有T，即A、B會同時為T。

第四類	x_1、x_2、⋯、x_n全為F。	設定y_1、y_2、⋯、y_{n-3}全為F。	會導致除了第一個括弧為 F，其餘的括弧皆為T，即A、B會同時為F

(以上y的配置不唯一，但會使 A、B 等價)

(二)$3-SAT$問題歸約到$k-clique$問題

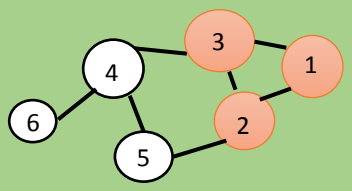
我們舉例$3-SAT$問題歸約到$k-clique$問題。假設給定的一組$3-SAT$為：$A=(x_1\vee x_2\vee \overline{x_3})\wedge(x_2\vee x_3\vee \overline{x_4})\wedge(x_1\vee \overline{x_2}\vee x_4)$。不難找出當$x_1$和$x_3$為$T$、$x_2$和$x_4$為$F$可使$A$為$T$。將問題中所有布林變數寫成左邊、上面和右邊，並將它們彼此合理的相互連接，故將A畫成如圖八所示。透過找$clique$的方式找圖八中存在的$k-clique$，如橘色三角形x_1、x_3、$\overline{x_2}$所連成的$3-clique$，即為A之解。因為每一條連線代表允許線的兩端點T，也就是說，橘色三角形的三端點x_1、x_3、$\overline{x_2}$為T時，就可以讓A為T。



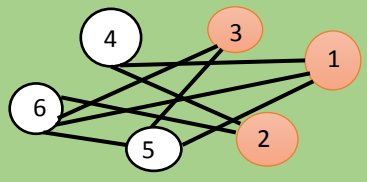
【圖八】$3-SAT$ 問題歸約到$3-clique$問題
(圖片來源：研究者自繪)

(三)$k-clique$問題歸約到$Independent\ Set$問題

我們舉例在圖四中的$3-clique$問題。將圖四中的任兩點之連線從有變無、從無變有，重新繪製如右圖九所示（也稱圖九和圖四互為補圖）。我們觀察，圖四中的{1,2,3}的三個點，就是大小為3的 clique(團)，而圖九中的{1,2,3}的三個點，就是一組圖九的獨立集。所以這兩個問題可以互相歸約。



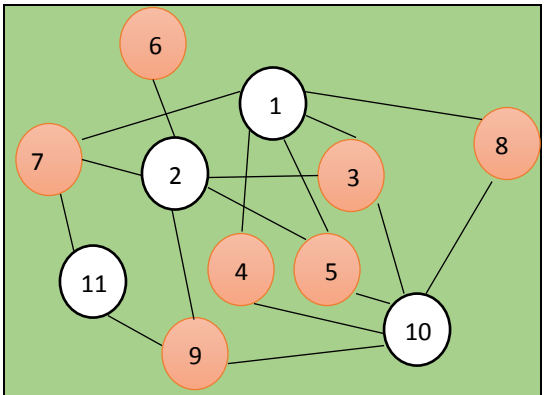
【圖四】 $3-clique$e 問題
(圖片來源：研究者自繪)



【圖九】 $Independent\ Set$問題
(圖片來源：研究者自繪)

(四)$Independent\ Set$問題歸約到)$Vertex\ cover$問題

如右圖五，{3,4,5,6,7,8,9}共七個點(紅色)為該圖的一個獨立集，而剩下的{1,2,10,11}共四個點(白色)就是$Vertex\ cover$點覆蓋。圖五中的每一個邊的兩端點，有可能為**紅紅**、**紅白**和**白白**共三種，而屬於**紅紅**這一種的邊是0條（符合$Independent\ Set$的定義），因此，所有的邊將會屬於**紅白**和**白白**這兩種，也就是，白色點的集合會符合)$Vertex\ cover$的定義，故兩種問題可以互相歸約。



【圖五】 $Independent\ Set$問題
(圖片來源：研究者自繪)

五、「數獨問題」歸約到「SAT問題」

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

【圖十】數獨棋盤(圖片來源：研究者自繪)

在 https://willyc20.github.io 網站中研究到將四色問題(註：四色問題是如果地圖上相鄰的兩個區域不能畫相同的顏色，那不管這張地圖多複雜，最少都可以用四種顏色畫出它。)寫成布林函數問題時的做法，真的讓我們感到十分驚艷！原來，利用布林函數的寫法能夠表達出四色問題背後的意義，雖然一個變數只有T、F兩種型式，但若配合$De\ Morgan's\ laws$，就能將兩個變數表達出多種變數，利用$\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$使$p$、$q$這兩個變數至多一個為真(等價不超過兩個)的概念，再用此概念完成數獨規則中每列、行和3×3區塊中只能填入1~9的一個數字(等價不超過兩個)，請見以下是我們原創的數獨問題歸約到SAT問題之詳細證明。圖十為9×9的數獨遊戲棋盤，首先將每個格子編碼，請見以下定義。

定義：

- x^n 代表如上圖數獨中棋盤的格子。其中$n = 1,2,\dots,81$。
- x_k^n 表示x^n 這一格的所代表的唯一數字k 的布林變數，即若x_k^n該格的數字為k，則x_k^n為T，x_k^n該格的數字為非k，則x_k^n為F。其中$n = 1,2,\dots,81$，$k = 1,2,\dots,9$。

舉例：若x^{53}該格的數字為9，x_9^{53}為T。若x^{22}該格的數字為6，x_4^{22}為F。因此，藉由數獨遊戲規則「同一列(行、3×3區塊)數字只能出現一次的1、2、3 …」，我們可知：x_1^1、x_1^2、x_1^3、x_1^4、x_1^5、x_1^6、x_1^7、x_1^8、x_1^9只有一個會是T，其餘八個為F。接著，我們思考八十一個格子中的x^1，到底**此格**要填甚麼數字呢？請見以下步驟a、b、c：

步驟a：$(\overline{x_1^1} \vee \overline{x_2^1}) \wedge (\overline{x_1^1} \vee \overline{x_3^1}) \wedge \dots \wedge (\overline{x_8^1} \vee \overline{x_9^1})$，其中任一的$(\overline{x_r^n} \vee \overline{x_s^n})$的$r \neq s$。
步驟b：$(x_1^1 \vee x_2^1 \vee x_3^1 \vee x_4^1 \vee x_5^1 \vee x_6^1 \vee x_7^1 \vee x_8^1 \vee x_9^1)$。
步驟c：將步驟a和步驟b的結果取交集，即

$$(\overline{x_1^1} \vee \overline{x_2^1}) \wedge (\overline{x_1^1} \vee \overline{x_3^1}) \wedge \dots \wedge (\overline{x_8^1} \vee \overline{x_7^1}) \wedge (\overline{x_8^1} \vee \overline{x_9^1}) \wedge (x_1^1 \vee x_2^1 \vee x_3^1 \vee x_4^1 \vee x_5^1 \vee x_6^1 \vee x_7^1 \vee x_8^1 \vee x_9^1)$$

*(格檢驗，檢查該格只能出現*1、2、3、 …、9*其中之一的數字)*

在步驟a中，共有$C_2^9 = 36$個步驟$(\overline{x_r^n} \vee \overline{x_s^n})$的交集，其目的是使$x_1^1$、$x_2^1$、$x_3^1$、$x_4^1$、$x_5^1$、$x_6^1$、$x_7^1$、$x_8^1$、$x_9^1$僅有零個或一個為$T$，即$(\overline{x_1^1} \vee \overline{x_2^1}) \wedge (\overline{x_1^1} \vee \overline{x_3^1}) \wedge \dots \wedge (\overline{x_8^1} \vee \overline{x_9^1})$結果必為$T$。如果$x_1^1$、$x_2^1$、$x_3^1$、$x_4^1$、$x_5^1$、$x_6^1$、$x_7^1$、$x_8^1$、$x_9^1$中有兩個以上為$T$，例如$x_1^1$、$x_2^1$皆為$T$，則會使得$(\overline{x_1^1} \vee \overline{x_2^1})$為 F，但這個情況是不合理的，由數獨遊戲規則可知，x_1^1、x_2^1、x_3^1、x_4^1、x_5^1、x_6^1、x_7^1、x_8^1、x_9^1

不可能有兩個以上為T。

在步驟b中，其目的是使x_1^1、x_2^1、x_3^1、x_4^1、x_5^1、x_6^1、x_7^1、x_8^1、x_9^1有一個以上為T，即$(x_1^1 \vee x_2^1 \vee x_3^1 \vee x_4^1 \vee x_5^1 \vee x_6^1 \vee x_7^1 \vee x_8^1 \vee x_9^1)$結果必為$T$。如果$x_1^1$、$x_2^1$、$x_3^1 \cdots$、$x_9^1$有零個為$T$即皆為$F$，那會使得$(x_1^1 \vee x_2^1 \vee x_3^1 \vee x_4^1 \vee x_5^1 \vee x_6^1 \vee x_7^1 \vee x_8^1 \vee x_9^1)$結果為$F$，但這個情況不合理，由數獨遊戲規則可知，$x_1^1$、$x_2^1$、$x_3^1$、$x_4^1$、$x_5^1$、$x_6^1$、$x_7^1$、$x_8^1$、$x_9^1$只能有一個為$T$。

在步驟c中，由於是決定於步驟a和b之交集，故會使步驟c之結果必為T，因此x_1^1、x_2^1、x_3^1、x_4^1、x_5^1、x_6^1、x_7^1、x_8^1、x_9^1**必然只有一個為**T。換句話說，透過步驟c，我們可以確認x^1該格只會出現1、2、3、…、9其中某一個數字。我們可以重複此動作，一直討論到x^{81}格的數字。

接著思考**每列**的數字該如何放，我們舉第一列數字x^1、x^2、x^3、x^4、x^5、x^6、x^7、x^8、x^9為例。

步驟d：$(\overline{x_1^1} \vee \overline{x_1^2}) \wedge (\overline{x_1^1} \vee \overline{x_1^3}) \wedge \ldots \wedge (\overline{x_1^8} \vee \overline{x_1^9})$，其中任一的$(\overline{x_n^r} \vee \overline{x_n^s})$的$r \neq s$。

*(列檢驗，檢查該列是否為*1、2、3、…、9*各*0*或*1*次)*

在步驟d中，共有$C_2^9 = 36$個步驟$(\overline{x_n^r} \vee \overline{x_n^s})$的交集，其目的是使$x_1^1$、$x_1^2$、$x_1^3 \cdots$、$x_1^9$僅有0或1個為$T$，即第一列的九個格子$x^1$、$x^2$、$x^3$、$x^4$、$x^5$、$x^6$、$x^7$、$x^8$、$x^9$中只能出現零次或一次的1。如果$x_1^1$、$x_1^2$、$x_1^3$、$x_1^4$、$x_1^5$、$x_1^6$、$x_1^7$、$x_1^8$、$x_1^9$中有2個以上為$T$，例如$x_1^1$、$x_1^2$皆為$T$，則會使得$(\overline{x_1^1} \vee \overline{x_1^2})$為$F$，但這個情況是不合理的，由數獨遊戲規則可知，$x_1^1$、$x_1^2$、$x_1^3 \cdots$、$x_1^9$不可能有兩個以上為$T$。另外一個情況是，如果$x_1^1$、$x_1^2$、$x_1^3$、$x_1^4$、$x_1^5$、$x_1^6$、$x_1^7$、$x_1^8$、$x_1^9$中有0個為$T$，會使該步驟結果為$T$，但也不合理，因為會使得$x^1$、$x^2$、$x^3 \cdots$、$x^9$中出現兩個重複的數字，而違反數獨遊戲規則。

同理，由步驟d知，我們可以用$(\overline{x_2^1} \vee \overline{x_2^2}) \wedge (\overline{x_2^1} \vee \overline{x_2^3}) \wedge \ldots \wedge (\overline{x_2^8} \vee \overline{x_2^9})$確認，其目的是使$x_2^1$、$x_2^2$、$x_2^3 \cdots$、$x_2^9$僅有0或1個為$T$，即第一列的九個格子$x^1$、$x^2$、$x^3 \cdots$、$x^9$中只能出現0或1次的2。所以，第一列的數字依照此步驟之檢查可以完成1、2、3、…、9的放置。故我們完成數獨遊戲中的九列數字的檢驗。

檢驗完每列，那**每行**該怎麼檢驗？其實跟步驟d一樣，底下我們舉第一行數字x^1、x^{10}、x^{19}、x^{28}、x^{37}、x^{46}、x^{55}、x^{64}、x^{73}為例。

步驟e：$(\overline{x_1^1} \vee \overline{x_1^{10}}) \wedge (\overline{x_1^1} \vee \overline{x_1^{19}}) \wedge (\overline{x_1^1} \vee \overline{x_1^{28}}) \wedge \ldots \wedge (\overline{x_1^{64}} \vee \overline{x_1^{73}})$，其中任一的$(\overline{x_n^r} \vee \overline{x_n^s})$的$r \neq s$。

*(行檢驗，檢查該行是否為*1、2、3、…、9*各*0*或*1*次)*

承步驟d的解釋，可以檢查第一行的九個格子x^1、x^{10}、x^{19}、x^{28}、x^{37}、x^{46}、x^{55}、x^{64}、x^{73}中只能出現0或1次的1，再檢查2、……類推至9，進而完成數獨遊戲中的九行數字的檢驗。

最後，數獨遊戲還要檢查**3 × 3區塊**中，是否放置一次的1、2、3、 …、9，因此，我們再以同樣邏輯檢驗，以下我們舉左上角區塊x^1 、x^2 、x^3 、x^{10}、x^{11}、x^{12}、x^{19}、x^{20}、x^{21}為例。

步驟f：$(\overline{x_1^1} \vee \overline{x_1^2}) \wedge (\overline{x_1^1} \vee \overline{x_1^3}) \wedge (\overline{x_1^1} \vee \overline{x_1^{10}}) \wedge \ldots \wedge (\overline{x_1^{20}} \vee \overline{x_1^{21}})$，其中任一塊的$(\overline{x_n^r} \vee \overline{x_n^s})$的$r \neq s$。

*(塊檢驗，檢查該塊是否為*1、2、3、 …、9*各*0*或*1*次))*

同前，可以檢查該塊的九個格子x^1 、x^2 、x^3 、x^{10}、x^{11}、x^{12}、x^{19}、x^{20}、x^{21}中只能出現零次或一次的1，再檢查2、……類推至9，進而完成數獨遊戲中的九塊數字的檢驗。當我們完成以上所有的格檢驗、列檢驗、行檢驗、塊檢驗的步驟時，即完成9×9的數獨遊戲檢查。

參●結論

藉由本次研究我們瞭解時間複雜度、P、NP、NPC、NPH等定義，也瞭解了經典的NPC問題和歸約，並在本文最後完成原創的9×9的數獨問題歸約到SAT問題之詳細證明，我們相信同理可證$n \times n$的數獨問題歸約。在數獨的歸約過程，若將步驟c、d 、e 、f的步驟數全部寫出則會有11745個步驟數，若考慮x_1^1到x_9^{81}共729個變數的窮舉法則會產生出2^{729}個變數，太多變數目前連圖靈機也不太可能全部列舉，況且還有推廣至$n \times n$的數獨。數獨其實後來有很多衍生出很多有趣的變形規則，且數獨遊戲起初每提供一個格子的數字的提示等價於提供28個的變數(舉例：若x_3^1是3為T，則x_1^1、x_2^1、x_4^1、x_5^1、x_6^1、x_7^1、x_8^1、x_9^1、x_3^2、x_3^3、x_3^4、x_3^5、x_3^6、x_3^7、x_3^8、x_3^9、x_3^{10}、x_3^{19}、x_3^{28}、x_3^{37}、x_3^{46}、x_3^{55}、x_3^{64}、x_3^{73}、x_3^{11}、x_3^{12}、x_3^{20}、x_3^{21}共28個變數皆為F)，以我們的能力還沒有辦法研究到此，未來希望能努力學習數論、離散數學和程式設計的知識，近程目標是強化我們的資訊能力，更深入學習計算複雜性理論，遠程目標是期待有天能夠解出$P \neq NP$的證明(至少我們目前是這樣的認為的！)。

肆●引註資料

1.Stanley B. Lippman 等、黃銘偉(2019)。**C++ Primer 中文版**。臺北市：碁峰資訊。

2.楊尊一(譯)(2018)。**Think Complexity： 複雜性科學與計算模型設計(第二版)**。臺灣：歐萊禮。

3.OpenDSA Data Structures and Algorithms Modules Collection。2019 年 9 月 5 日，取自於：https://reurl.cc/GVrqry

4.初學者演算法—談甚麼是演算法和時間複雜度。2019 年 9 月 5 日，取自於：https://reurl.cc/qd8bmN

5.如何衡量程式的效率？—論時間複雜度。2019 年 9 月 5 日，取自於：https://reurl.cc/vD1b5o

6.計算機的始祖：圖靈機。2019 年 10 月 6 日，取自於：https://reurl.cc/5lqLrz

7.P/NP 問題—維基百科。2019 年 10 月 6 日，取自於：https://reurl.cc/3DLAa8

8.輕鬆談演算法複雜度的分界—什麼是 P,NPNP-Complete,NP-Hard 問題。2019 年 9 月 8 日，取自於：https://reurl.cc/O1qakr

9.RSA 加密演算法—維基百科。2019 年 12 月 7 日，取自於：https://reurl.cc/4RmD4X

10.SAT – problem 的介紹。2019 年 12 月 13 日，取自於：https://ppt.cc/fnNyhx