中華民國第 59 屆中小學科學展覽會

作品說明書

國中組　生活與應用科學(一)科

探究精神獎

032803

解開點線面 - 用 SAT Solver 解開 Flow Free 遊戲

學校名稱：新竹縣立成功國民中學

作者：	指導老師：
國二 黃仲璿	吳宗倫
國二 林立宸	謝東育

關鍵詞：可滿足性問題、布林運算、程式

摘要

我們研究了 Flow Free 遊戲的特徵後，發現他與另一個遊戲——數獨在規則上有許多相似處。於是我們先轉而研究網路上已經有多種解法的數獨遊戲，嘗試將那些方法套用於 Flow Free 去解。經過多次嘗試，我們發現一種邏輯簡單且可行的方法 — SAT Solver。SAT Solver (Boolean Satisfiability Problem Solver 或作 Propositional Satisfiability Problem Solver)最大的特色是需要使用 Boolean Variable，也就是一切條件都必須轉換成是非題。「這一格是哪一個顏色?」變成了「這一格是不是顏色一?是不是顏色二?是不是顏色三?...」。經過研究，我們得到一個結論：如果一個問題沒有步驟性(上一步驟不影響下一步的答案)且所有條件都能以是非條件式表示，就可以用 SAT Solver 解開。

壹、研究動機

我們在 Google Play 商店中發現了一個名為 Flow Free 的遊戲。這個遊戲的規則是在不交叉的情況下把相同顏色的點點連接起來，即可過關。Flow Free 以簡單的規則引起我們的興趣。一開始單憑直覺一步一步的破關方法，隨著關卡越來越後面，版面與需連線的點逐漸增加，漸漸變得窒礙難行，幾乎沒辦法在不使用提示的情況下將遊戲解出來。這讓我們想研究這個遊戲，我們希望能透過電腦、演算法等等來找出一種可以解開所有關卡的方式。

貳、研究目的

一、研究其他非步驟性遊戲以及已知的演算法嘗試推導出可以解出 Flow Free 的方法

二、找出是否所有關卡都有解

參、研究設備及器材

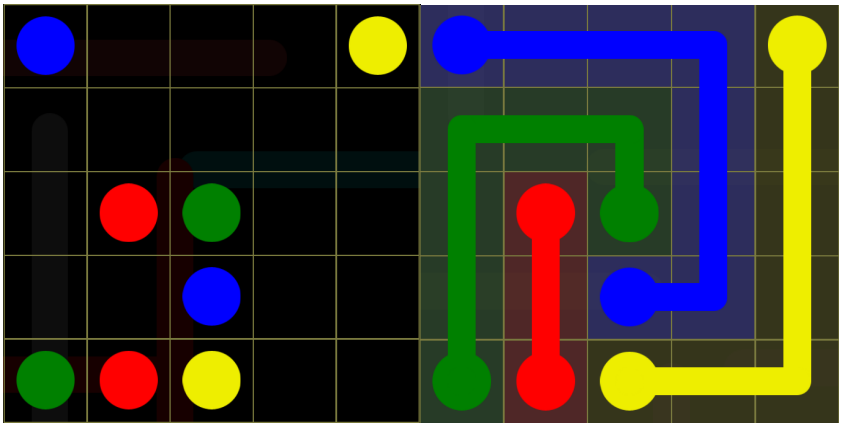
一、電腦 (Python 程式語言)

二、手機 (Flow Free 應用程式)

肆、研究過程與方法

一、Flow Free 與數獨

(一)、何謂 Flow Free



▲遊戲的初始樣貌　　▲遊戲解開後的樣貌

這是一個 Flow Free 遊戲的範例。解開 Flow Free 遊戲的方法是將所有顏色相同的點兩兩連線。Flow Free 的規則如下：

1. 連線不可超出遊戲邊界
2. 連線不可交叉重疊
3. 不同顏色的兩點不可連線
4. 連線不可通過有其他顏色的點的格子

由以上規則我們可以延伸出以下的條件：

1. 每一格至少且只能有一種顏色
2. 每個非終點格至少且只能有一個連出方向
3. 每個起始格與終點格鄰近至少且只能有一個與它相同顏色的格子
4. 每個非終點格必須與它所連向的格顏色相同
5. 兩個格子不得互相連向
6. 方向不可指出邊界外
7. 方向不可指向起始格
8. 每個非起始格至少且只能被一個格子連向

(二)、Flow Free 與數獨之相似處

在這裡我們可以發現，Flow Free 許許多多的規則，導致遊戲中的連線方法與每格的顏色固定，先進行哪一個步驟並不會影響其他步驟的進行，也就是代表 Flow Free 具有每一個步驟不會互相影響的「非步驟性」的性質。數獨中每一格可填入的答案固定，因此數獨也擁有這個特徵。

經過研究與嘗試後我們成功的將 Flow Free 中的所有條件轉換成是非條件式。例如，我們將「這一格是哪一個顏色?」轉換成「這一格是不是顏色一?是不是顏色二?是不是顏色三?...」。這代表 Flow Free 有「所有的條件都可以用是非條件式表示」的特性。數獨的條件可以表示成「這一格是不是 1?」「這一格是不是 2?」...，因此數獨也擁有這項特徵。

非步驟性和所有的條件都可以用是非條件式表示都是要用 SAT Solver 解開問題的必要條件。由此推斷能解開數獨的 SAT Solver 也能夠解開 Flow Free。

二、布林可滿足性問題

(一)、定義

布林可滿足性問題(Boolean Satisfiability Problem，簡稱 SAT)是一個用於確認布林運算式是否可以滿足所有給定條件的方法。若布林運算式內的所有布林變數均可以被賦值，且被給定的值都能夠滿足所有條件，則該問題為有解; 反之則為無解。

(二)、SAT Solver

SAT Solver 是一套用於解開可滿足性問題的演算法。此演算法會持續的將每個布林變數賦予一個值 True(是)或 False(非)，直到有一組解滿足所有給予的條件。若沒有一組解可滿足所有的條件，則該問題為無解。將用 CNF 表示的布林變數丟給 SAT Solver 運算後，即可得到該問題的解答。以下列這個 CNF 為例:

[[1, -5, 4], [-1, 5, 3, 4], [-3, -4], [2, 3]]

經過 SAT Solver 運算後，即可得到以下的結果:

[1, 2, 3, -4, -5]

三、何謂 CNF

CNF(Conjunctive Normal Form) 是一種用於表示布林變數(Boolean Variable)的方法。一個布林變數(v)只有兩種可能的值：True(T) 或者 False(F)。若 v = T，則 ¬v= F。 CNF 表示法中，有文字(literal)、子句(clause)跟 CNFs。

- 一個文字(literal) l 的形式為 v 或 ¬v
- 一個子句(clause) c 的形式為 $(l_1 \vee l_2 \vee ... \vee l_n)$
- 一個 CNFs 的形式為 $c_1 \wedge c_2 \wedge ... \wedge c_n$

(注：∨為布林運算中的「或 (or)」，∧為布林運算中的「且 (and)」，¬為布林運算中的「非 (not)」)

舉例來說，(A ∨ B) ∧ (C ∨ ¬D)是 CNF，((A ∨ B) ∧ (C ∨ ¬D)) ∧ ((¬E ∨ F) ∧ (G ∨ ¬H))是 CNFs。

布林變數中，還有一個重要的觀念——那就是¬(A ∧ B) = (¬A ∨ ¬B)。

四、以迷你數獨為例

這裡我們用一個 3×3 的迷你數獨來解釋如何用 SAT Solver 解數獨，題目如下圖：

1		
		2
	3	

(一)、定義變數

首先我們先定義格子。由左上角到右下角分別為第一格~第九格。也就是說左上角那一格為第一格、右上角那一格為第三格...以此類推。繪成圖表為：

一	二	三
四	五	六
七	八	九

我們再來定義 27 個變數，

變數 1 代表數獨中第一格的數字為 1、變數 2 代表數獨中第一格的數字為 2、變數 3 代表數獨中第一格的數字為 3；

變數 4 代表數獨中第二格的數字為 1、變數 5 代表數獨中第二格的數字為 2、變數 6 代表數獨中第二格的數字為 3… 以此類推。

(二)、依據題目取得條件

1. 題目給予的線索

```
CNF = [[1], [17], [24]]
```

[1]代表第一格為 1、[17]代表第六格為 2、[24]代表第八格為 3。

2. 每格至少有一個數字為正確答案

```
CNF = [[1,2,3], [4,5,6], [7,8,9],
      [10,11,12], [13,14,15], [16,17,18],
      [19,20,21], [22,23,24], [25,26,27]]
```

[1,2,3]代表第一格可能為 1 或 2 或 3、

[4,5,6]代表第二格可能為 1 或 2 或 3、

以此類推，

[25,26,27]代表第九格可能為 1 或 2 或 3。

3. 每格只能有一個數字為正確答案

```
CNF=[[-1,-2], [-1,-3], [-2,-3],
[-4,-5], [-4,-6], [-5,-6],
[-7,-8], [-7,-9], [-8,-9],
[-10,-11], [-10,-12], [-11,-12],
[-13,-14], [-13,-15], [-14,-15],
[-16,-17], [-16,-18], [-17,-18],
[-19,-20], [-19,-21], [-20,-21],
[-22,-23], [-22,-24], [-23,-24],
[-25,-26], [-25,-27], [-26,-27]]
```

[-1,-2]代表第一格不能同時為 1 和 2、

[-1,-3]代表第一格不能同時為 1 和 3、

[-2,-3]代表第一格不能同時為 2 和 3、

所以[-1,-2],[-1,-3],[-2,-3]代表第一格的答案只能是 1、2、3 其中一個；

[-4,-5],[-4,-6],[-5,-6]代表第二格的答案只能是1、2、3其中一個；

以此類推，

[-25,-26],[-25,-27],[-26,-27]代表第九格的答案只能是 1、2、3 其中一個。

4. 每一列至少有一個格子為1、2、3

```
CNF = [[1,4,7], [2,5,8], [3,6,9],
      [10,13,16], [11,14,17], [12,15,18],
      [19,22,25], [20,23,26], [21,24,27]]
```

[1,4,7]代表第一列至少有一格為1、

[2,5,8]代表第一列至少有一格為2、

[3,6,9]代表第一列至少有一格為3、

以此類推，

[19,22,25]代表第三列至少有一格為1、

[20,23,26]代表第三列至少有一格為2、

[21,24,27]代表第三列至少有一格為3。

5. 每一行至少有一個格子為1、2、3

```
CNF = [[1,10,19], [2,11,20], [3,12,21],
      [4,13,22], [5,14,23], [6,15,24],
      [7,16,25], [8,17,26], [9,18,27]]
```

[1,10,19]代表第一行至少有一格為1、

[2,11,20]代表第一行至少有一格為2、

[3,12,21]代表第一行至少有一格為3、

…

[7,16,25]代表第三行至少有一格為1、

[8,17,26]代表第三行至少有一格為2、

[9,18,27]代表第三行至少有一格為3。

6. 每一列只能有一個格子為1、2、3

```
CNF = [[-1,-4],[-1,-7],[-4,-7],
[-2,-5],[-2,-8],[-5,-8],
```

```
[-3,-6],[-3,-9],[-6,-9],
[-10,-13],[-10,-16],[-13,-16],
[-11,-14],[-11,-17],[-14,-17],
[-12,-15],[-12,-18],[-15,-18],
[-19,-22],[-19,-25],[-22,-25],
[-20,-23],[-20,-26],[-23,-26],
[-21,-24],[-21,-27],[-24,-27]]
```

[-1,-4]第一列的第一格和第二格不可同時為 1、

[-1,-7]第一列的第一格和第三格不可同時為 1、

[-4,-7]第一列的第二格和第三格不可同時為 1、

所以[-1,-4],[-1,-7],[-4,-7]代表第一列只能有一個格子為 1、

所以[-2,-5],[-2,-8],[-5,-8]代表第一列只能有一個格子為 2、

所以[-3,-6],[-3,-9],[-6,-9]代表第一列只能有一個格子為 3、

…

所以[-19,-22],[-19,-25],[-22,-25]代表第三列只能有一個格子為 1、

所以[-20,-23],[-20,-26],[-23,-26]代表第三列只能有一個格子為 2、

所以[-21,-24],[-21,-27],[-24,-27]代表第三列只能有一個格子為 3。

7. 每一行只能有一個格子為 1、2、3

```
CNF = [[-1,-10],[-1,-19],[-10,-19],
[-2,-11],[-2,-20],[-11,-20],
[-3,-12],[-3,-21],[-12,-21],
[-4,-13],[-4,-22],[-13,-22],
[-5,-14],[-5,-23],[-14,-23],
[-6,-15],[-6,-24],[-15,-24],
[-7,-16],[-7,-25],[-16,-25],
[-8,-17],[-8,-26],[-17,-26],
[-9,-18],[-9,-27],[-18,-27]]
```

[-1,-10]第一行的第一格和第二格不可同時為 1、

[-1,-19]第一行的第一格和第三格不可同時為 1、

[-10,-19]第一行的第二格和第三格不可同時為 1、

所以[-1,-10],[-1,-19],[-10,-19]代表第一行只能有一個格子為 1、

所以[-2,-11],[-2,-20],[-11,-20]代表第一行只能有一個格子為 2、

所以[-3,-12],[-3,-21],[-12,-21]代表第一行只能有一個格子為 3、

…

所以[-7,-16],[-7,-25],[-16,-25]代表第三行只能有一個格子為 1、

所以[-8,-17],[-8,-26],[-17,-26]代表第三行只能有一個格子為 2、

所以[-9,-18],[-9,-27],[-18,-27]代表第三行只能有一個格子為 3。

8. 將以上所有條件交由 SAT Solver 去解，得到的答案為：

```
[1, -2, -3,      -4, 5, -6,       -7, -8, 9,
-10, -11, 12,    13, -14, -15,    -16, 17, -18,
-19, 20, -21,    -22, -23, 24,    25,-26, -27]
```

也就是：

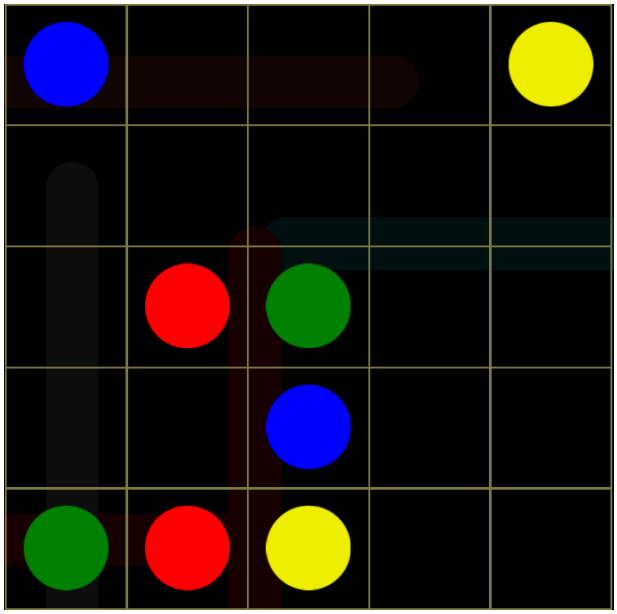
1	2	3
3	1	**2**
2	**3**	1

伍、研究結果

一、解開 Flow Free

(一)、定義 Flow Free

研究完迷你數獨後，我們用相同將 Flow Free 的格子以編號 0~m 的方式表示，每一個格子有 dColor+4 種可能性(總顏色數+4 種方向)。因此在一個完整的 Flow Free 中，總共會有`(m + 1)*(dColor + 4)`種可能性。我們以下列這個 5x5 的 Flow Free 為預設題目:



(二)、條件設置

必須要有初始顏色格，電腦再依下列條件判斷出其他格的顏色及方向。

1. 每一格至少且只能有一種顏
2. 每個非終點格至少且只能有一個連出方向
3. 每個起始格與終點格鄰近至少且只能有一個與它相同顏色的格子
4. 每個非終點格必須與它所連向的格顏色相同
5. 兩個格子不得互相連向
6. 方向不可指出邊界外
7. 方向不可指向起始格
8. 每個非起始格至少且只能被一個格子連向

(三)、演算法解析

1. 定義變數的表示方式

(1)以二維座標(x, y)的方式將條件變數輸入至 CNF 中

```
def xyVar(n, dMax, x, y, d):
      return dMax * (n * y + x) + d
```

(2)以一維座標(m)的方式將條件變數輸入至 CNF 中

```
def mVar(dMax, m, d):
      return dMax * m + d
```

2. 將 Flow Free 轉換為 CNF

(1)變數宣告

a. 函數定義

```
def getClauses(flowfree):
```

b.Flow Free 的總格子數

```
      m = len(flowfree)
```

c.Flow Free 的行/列格子數

```
      n = int(m ** 0.5)
```

d.Flow Free 的總顏色數

```
      dColor = 0
```

e.Flow Free 的總變化數

```
      dMax = 0
```

f.CNF clauses

```
      clauses = []
```

g.起始點和終點

```
      terminalPoints = []
```

h.起始點

```
      beginPoints = []
```

i.起始點的顏色

```
      valBeginPoints = []
```

j.起始點的座標

```
      idxBeginPoints = []
```

k.非終點格的座標

```
      idxNonEndPoints = []
```

(2)取得初始變數

a. 取得 Flow Free 的總顏色數

```
      dColor = max(flowfree)
```

b.取得 Flow Free 的總變化數

```
      dMax = dColor + 4
```

c.取得 Flow Free 的起始格、終點格以及非終點格的座標

```
      for i in range(m):
            if flowfree[i] != 0:
                  terminalPoints += [mVar(dMax,
      i ,flowfree[i])]
```

```
            if flowfree[i] != 0 and flowfree[i] not in
            valBeginPoints:
                     beginPoints += [mVar(dMax, i,
            flowfree[i])]
                        idxBeginPoints += [i]
                  valBeginPoints += [flowfree[i]]
                    if flowfree[i] == 0:
                        idxNonEndPoints += [i]
      idxNonEndPoints += idxBeginPoints
      idxNonEndPoints.sort()
```

d.將題目原有顏色的格子輸入至 CNF 中

```
      for i in range(m):
            if flowfree[i] != 0:
            clauses += [[mVar(dMax, i, flowfree[i])]]
```

CNF：[[1], [34], [91], [100], [137], [164], [171], [178]] (共 8 個)

(3)條件設置

a.每一格至少要有一種顏色

```
  for i in range(m):
        clause = []
        for d in range(1, dColor + 1):
            clause += [mVar(dMax, i, d)]
        clauses += [clause]
```

CNF : [[1, 2, 3, 4], [9, 10, 11, 12], …, [193, 194, 195, 196]] (共 25 個)

b.每一格只能有一種顏色

```
   for i in range(m):
         for d in range(1, dColor):
            for k in range(d + 1, dColor + 1):
                  clauses += [[-mVar(dMax, i, d), -
mVar(dMax, i, k)]]
```

CNF : [[-1, -2], [-1, -3], …, [-195, -196]] (共 150 個)

c.每一個非終點格至少要有一個連出方向

```
      for i in idxNonEndPoints:
            clause = []
            for d in range(dColor + 1, dMax + 1):
                  clause += [mVar(dMax, i, d)]
      clauses += [clause]
```

CNF : [[5, 6, 7, 8], [13, 14, 15, 16], …, [197, 198, 199, 200]] (共 21 個)

d.每一個非終點格只能有一個連出方向

```
      for i in idxNonEndPoints:
            for d in range(dColor + 1, dMax):
                  for k in range(d + 1, dMax + 1):
                        clauses += [[-mVar(dMax, i, d),
-mVar(dMax, i, k)]]
```

CNF : [[-5, -6], [-5, -7], …, [-199, -200]] (共 126 個)

e.每個起始格與終點格鄰近至少有一個與它相同顏色的格子

```
      for terminalPoint in terminalPoints:
            neighbors = []
            dCell = (terminalPoint - 1) % dMax + 1  #
cell's color number
            xCell = ((terminalPoint - 1) // dMax) % n
# cell's x coordinate
            yCell = (terminalPoint - 1) // (n * dMax)
# cell's y coordinate
            if xCell > 0:
                  neighbors += [xyVar(n, dMax, xCell -
1, yCell, dCell)]
            if xCell < n - 1:
                  neighbors += [xyVar(n, dMax, xCell +
1, yCell, dCell)]
            if yCell > 0:
                  neighbors += [xyVar(n, dMax, xCell,
yCell - 1, dCell)]
            if yCell < n - 1:
                  neighbors += [xyVar(n, dMax, xCell,
yCell + 1, dCell)]
      clauses += [neighbors]
```

CNF : [[9, 41], [26, 74], ..., [170, 186, 138]] (共 8 個)

f.每個起始格與終點格鄰近只能有一個與它相同顏色的格子

```
      for i in range(len(neighbors) - 1):
            for j in range(i + 1, len(neighbors)):
                  clauses += [[-neighbors[i], -
neighbors[j]]]
```

CNF : [[9, 41], [-9, -41], …, [-186, -138]] (共 27 個)

g.每個非終點格必須與它所連向的格顏色相同

```
      for i in idxNonEndPoints:
                  x = i % n
            y = i // n
                  for j in range(1, dColor + 1):
                        if x > 0:
                              clauses += [[-xyVar(n,
dMax, x, y, dColor + 3), -xyVar(n, dMax, x, y, j),
xyVar(n, dMax, x - 1, y, j)]]
                        if x < n - 1:
                              clauses += [[-xyVar(n,
dMax, x, y, dColor + 4), -xyVar(n, dMax, x, y, j),
xyVar(n, dMax, x + 1, y, j)]]
                        if y > 0:
                              clauses += [[-xyVar(n,
dMax, x, y, dColor + 1), -xyVar(n, dMax, x, y, j),
xyVar(n, dMax, x, y - 1, j)]]
                        if y < n - 1:
                              clauses += [[-xyVar(n,
dMax, x, y, dColor + 2), -xyVar(n, dMax, x, y, j),
xyVar(n, dMax, x, y + 1, j)]]
```

CNF : [[-6, -1, 41], [-8, -1, 9], ..., [-199, -196, 188]] (共 272 個)

h.兩個格子不得互相連向

```
  # up
    for x in range(n):
        for y in range(1, n):
            clauses += [[-xyVar(n, dMax, x, y, dColor
+ 1), -xyVar(n, dMax, x, y - 1, dColor + 2)]]
    # down
    for x in range(n):
        for y in range(n - 1):
            clauses += [[-xyVar(n, dMax, x, y, dColor
+ 2), -xyVar(n, dMax, x, y + 1, dColor + 1)]]
    # left
    for x in range(1, n):
        for y in range(n):
            clauses += [[-xyVar(n, dMax, x, y, dColor
+ 3), -xyVar(n, dMax, x - 1, y, dColor + 4)]]
    # right
    for x in range(n - 1):
        for y in range(n):
            clauses += [[-xyVar(n, dMax, x, y, dColor
+ 4), -xyVar(n, dMax, x + 1, y, dColor + 3)]]
```

CNF : [[-45, -6], [-85, -46], ..., [-192, -199]] (共 80 個)

i.連出方向不可連向邊界

```
   for i in range(n):
        clauses += [[-xyVar(n, dMax, i, 0, dColor+1)]]
        clauses += [[-xyVar(n, dMax, i,n-1, dColor+2)]]
        clauses += [[-xyVar(n, dMax, 0, i, dColor+3)]]
        clauses += [[-xyVar(n, dMax,n-1, i, dColor+4)]]
```

```
CNF : [[-5], [-166], [-7], [-40], [-13], [-
174], [-47], [-80], [-21], [-182], [-87], [-
120], [-29], [-190], [-127], [-160], [-37], [-
198], [-167], [-200]]
```
(共 20 個)

j.連出方向不可連向起始格

```
  for i in range(m):
        neighbors = []
        xCell = i % n
        yCell = i // n
        if xCell > 0:  # flow right
            neighbors += [mVar(dMax, i-1, dColor + 4)]
        if xCell < n - 1:  # flow left
            neighbors += [mVar(dMax, i + 1, dColor+3)]
        if yCell > 0:  # flow down
            neighbors += [mVar(dMax, i - n, dColor+2)]
        if yCell < n - 1:  # flow up
            neighbors += [mVar(dMax, i + n, dColor+1)]
        if i in idxBeginPoints:
            for neighbor in neighbors:
                clauses += [[-neighbor]]
```

```
CNF : [[-15], [-45], [-32], [-77], [-88], [-
103], [-54], [-133], [-96], [-111], [-62], [-
141]]
```
(共 12 個)

k.每個非起始格至少被一個格子連向

```
  else:
            clauses += [neighbors]
```

```
CNF : [[8, 23, 53], [16, 31, 61], ..., [192,
158]]
```
(共 21 個)

l.每個非起始格只能被一個格子指向

```
        for i in range(len(neighbors) - 1):
            for j in range(i + 1, len(neighbors)):
                clauses += [[-neighbors[i],
-neighbors[j]]]
        return clauses, n, dMax
```

CNF : [[-8, -23], [-8, -53], ..., [-192,-158]]
(共 80 個)

3.轉換/輸出函式

```
def printFlowLine(flowfree, solved, dMax, n):
    letters = ['-', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
'W', 'X', 'Y', 'Z']
    # 1:up, 2:down, 3:left, 4:right
    dirList = []
    ascList = []
    for y in range(n):
        for x in range(n):
            ascList += [0]
            dir = 0
            for d in range(dMax - 4, dMax):
                if solved[dMax * (n * y + x) + d] > 0:
                    mod = solved[dMax * (n * y + x) + d] % dMax
                    if mod == 0:
                        mod = dMax
                    dir = 4 - (dMax - mod)
                    dirList += [dir]
            if dir == 0:
                dirList += [dir]

    dirs = [u'\u2550', u'\u2551', u'\u2554', u'\u2557', u'\u255a',
u'\u255d']

    for y in range(n - 1):
        for x in range(n):
            if dirList[n * y + x] == 2:  # down
                if dirList[n * (y + 1) + x] == 3:  # left
                    ascList[n * (y + 1) + x] = 5
                if dirList[n * (y + 1) + x] == 4:  # right
                    ascList[n * (y + 1) + x] = 4
                if dirList[n * (y + 1) + x] == 2:  # down
                    ascList[n * (y + 1) + x] = 1
    for y in range(1, n):
        for x in range(n):
            if dirList[n * y + x] == 1:  # up
                if dirList[n * (y - 1) + x] == 3:  # left
                    ascList[n * (y - 1) + x] = 3
                if dirList[n * (y - 1) + x] == 4:  # right
                    ascList[n * (y - 1) + x] = 2
                if dirList[n * (y - 1) + x] == 1:  # up
                    ascList[n * (y - 1) + x] = 1
```

```
    for y in range(n):
        for x in range(n - 1):
            if dirList[n * y + x] == 4:  # right
                if dirList[n * y + x + 1] == 1:  # up
                    ascList[n * y + x + 1] = 5
                if dirList[n * y + x + 1] == 2:  # down
                    ascList[n * y + x + 1] = 3
                if dirList[n * y + x + 1] == 4:  # right
                    ascList[n * y + x + 1] = 0
    for y in range(n):
        for x in range(1, n):
            if dirList[n * y + x] == 3:  # left
                if dirList[n * y + x - 1] == 1:  # up
                    ascList[n * y + x - 1] = 4
                if dirList[n * y + x - 1] == 2:  # down
                    ascList[n * y + x - 1] = 2
                if dirList[n * y + x - 1] == 3:  # left
                    ascList[n * y + x - 1] = 0

    for y in range(n):
        rowLine = ''
        for x in range(n):
            if flowfree[n * y + x] != 0:
                rowLine += letters[flowfree[n * y + x]]
            else:
                rowLine += dirs[ascList[n * y + x]]
        print(rowLine)
```

4.程式主體

(1)輸入要解開的題目

```
flowfree = [1, 0, 0, 0, 2, \
            0, 0, 0, 0, 0, \
            0, 3, 4, 0, 0, \
            0, 0, 1, 0, 0, \
            4, 3, 2, 0, 0]
```

(2)將題目丟至函數內並輸出

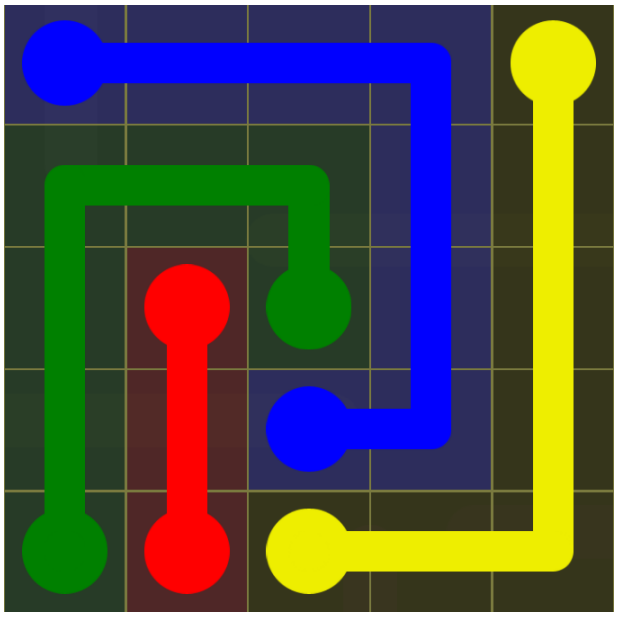
```
clauses, n, dMax = getClauses(flowfree)
solvedFlowfree = pycosat.solve(clauses)
if solvedFlowfree == 'UNSAT':
     print("No Solution")
else:
     printFlowLine(flowfree, solvedFlowfree, dMax, n)
```

(四)、結果

將我們設計的演算法得出的 CNF 丟給 SAT Solver 運算後，得出的結果如下:

[1, -2, -3, -4, -5, -6, -7, 8, 9, -10, -11, -12, -13, -14, -15, 16, 17, -18, -19, -20, -21, -22, -23, 24, 25, -26, -27, -28, -29, 30, -31, -32, -33, 34, -35, -36, -37, 38, -39, -40, -41, -42, -43, 44, -45, 46, -47, -48, -49, -50, -51, 52, -53, -54, 55, -56, -57, -58, -59, 60, -61, -62, 63, -64, 65, -66, -67, -68, -69, 70, -71, -72, -73, 74, -75, -76, -77, 78, -79, -80, -81, -82, -83, 84, -85, 86, -87, -88, -89, -90, 91, -92, -93, 94,-95, -96, -97, -98, -99, 100, 101, -102, -103, -104, 105, -106, -107, -108, -109, 110, -111, -112, -113, 114, -115, -116, -117, 118, -119, -120, -121, -122, -123, 124, -125, 126, -127, -128, -129, -130, 131, -132, -133, 134, -135, -136, 137, -138, -139, -140, -141, -142, -143, -144, 145, -146, -147, -148, -149, -150, 151, -152, -153, 154, -155, -156, -157, 158, -159, -160, -161, -162, -163, 164, -165, -166, -167, -168, -169, -170, 171, -172, -173, -174, -175, -176, -177, 178, -179, -180, -181, -182, -183,-184, -185, 186, -187, -188, -189, -190, 191, -192, -193, 194, -195, -196, -197, -198, 199, -200]

再經過轉換輸出函式後，即可得到以下的結果:



二、運算結果

我們從 5x5 到 14x14 的題目中各自隨機挑選了一個來做測試，結果如下。

(一)、 5x5

1. 題目:

```
1, 0, 0, 0, 2,
0, 0, 0, 0, 0,
0, 3, 4, 0, 0,
0, 0, 1, 0, 0,
4, 3, 2, 0, 0
```

2. 輸出:

3. CNF 數: 850

(二)、 6x6

1. 題目:

```
1, 0, 2, 0, 0, 0,
3, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 2,
0, 0, 0, 3, 4, 5,
0, 5, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0
```

2. 輸出:

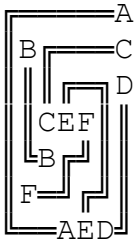
3. CNF 數: 1508

(三)、 7x7

1. 題目:

```
0, 0, 0, 0, 0, 0, 1,
0, 2, 0, 0, 0, 0, 3,
0, 0, 0, 0, 0, 0, 4,
0, 0, 3, 5, 6, 0, 0,
0, 0, 2, 0, 0, 0, 0,
0, 6, 0, 0, 0, 0, 0,
0, 0, 0, 1, 5, 4, 0
```

2. 輸出



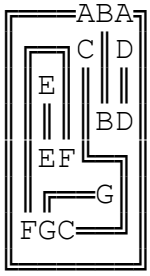
3. CNF 數: 2491

(四)、 8x8

1. 題目:

```
0, 0, 0, 0, 1, 2, 1, 0,
0, 0, 0, 0, 3, 0, 4, 0,
0, 0, 5, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 2, 4, 0,
0, 0, 5, 6, 0, 0, 0, 0,
0, 0, 0, 0, 0, 7, 0, 0,
0, 6, 7, 3, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0
```

2. 輸出:



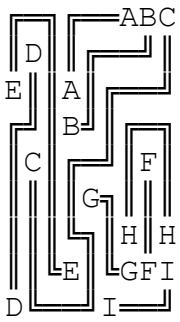
3. CNF 數:3884

(五)、9x9

1. 題目:

```
0, 0, 0, 0, 0, 0, 1, 2, 3,
0, 4, 0, 0, 0, 0, 0, 0, 0,
5, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 2, 0, 0, 0, 0, 0,
0, 3, 0, 0, 0, 0, 0, 6, 0,
0, 0, 0, 0, 7, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 8, 0, 8,
0, 0, 0, 5, 0, 0, 7, 6, 9,
4, 0, 0, 0, 0, 9, 0, 0, 0
```

2. 輸出:



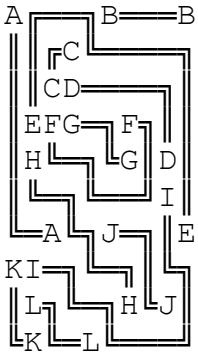
3. CNF 數: 6692

(六)、10x10

1. 題目:

```
1, 0, 0, 0, 0, 2, 0, 0, 0, 2,
0, 0, 0, 3, 0, 0, 0, 0, 0, 0,
0, 0, 3, 4, 0, 0, 0, 0, 0, 0,
0, 5, 6, 7, 0, 0, 6, 0, 0, 0,
0, 8, 0, 0, 0, 0, 7, 0, 4, 0,
0, 0, 0, 0, 0, 0, 0, 0, 9, 0,
0, 0, 1, 0, 0,10, 0, 0, 0, 5,
11,9, 0, 0, 0, 0, 0, 0, 0, 0,
0,12, 0, 0, 0, 0, 8, 0,10, 0,
0,11, 0, 0,12, 0, 0, 0, 0, 0,
```

2. 輸出:



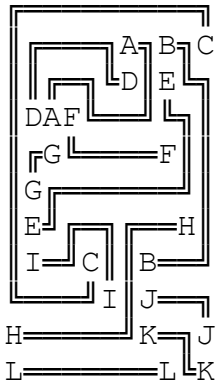
3. CNF 數: 12242

(七)、11x11

1. 題目:

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 3,
0, 0, 0, 0, 0, 0, 4, 0, 5, 0, 0,
0, 4, 1, 6, 0, 0, 0, 0, 0, 0, 0,
0, 0, 7, 0, 0, 0, 0, 0, 6, 0, 0,
0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 5, 0, 0, 0, 0, 0, 0, 0, 8, 0,
0, 9, 0, 0, 3, 0, 0, 2, 0, 0, 0,
0, 0, 0, 0, 0, 9, 0,10, 0, 0, 0,
8, 0, 0, 0, 0, 0, 0,11, 0, 0,10,
12,0, 0, 0, 0, 0, 0, 0,12, 0,11
```

2. 輸出:



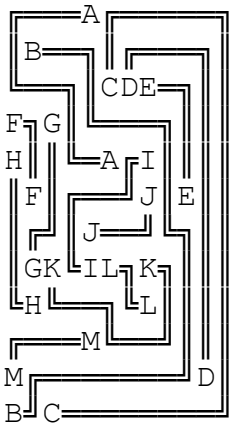
3. CNF 數: 14976

(八)、12x12

1. 題目:

```
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 3, 4, 5, 0, 0, 0, 0,
6, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0,
8, 0, 0, 0, 0, 1, 0, 9, 0, 0, 0, 0,
0, 6, 0, 0, 0, 0, 0,10, 0, 5, 0, 0,
0, 0, 0, 0,10, 0, 0, 0, 0, 0, 0, 0,
0, 7,11, 0, 9,12, 0,11, 0, 0, 0, 0,
0, 8, 0, 0, 0, 0, 0,12, 0, 0, 0, 0,
0, 0, 0, 0,13, 0, 0, 0, 0, 0, 0, 0,
13,0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0,
2, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

2. 輸出:



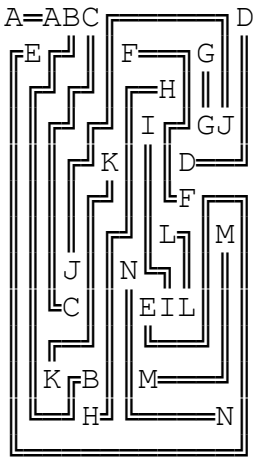
3. CNF 數: 20132

(九)、13x13

1. 題目:

```
1, 0, 1, 2, 3, 0, 0, 0, 0, 0, 0, 0, 4,
0, 5, 0, 0, 0, 0, 6, 0, 0, 0, 7, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 7,10, 0,
0, 0, 0, 0, 0,11, 0, 0, 0, 4, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,12, 0, 0,13, 0,
0, 0, 0,10, 0, 0,14, 0, 0, 0, 0, 0, 0,
0, 0, 0, 3, 0, 0, 0, 5, 9,12, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,11, 0, 2, 0, 0,13, 0, 0, 0, 0, 0,
0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0,14, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

2. 輸出:



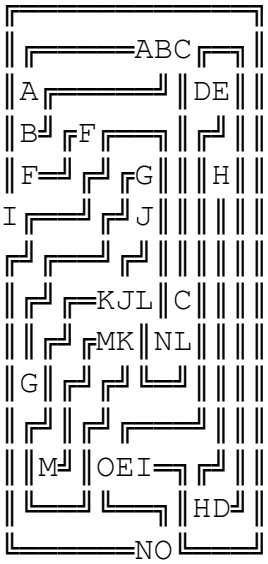
3. CNF 數: 26479

(十)、14x14

1. 題目:

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 4, 5, 0, 0,
0, 2, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 6, 0, 0, 0, 0, 0, 7, 0, 0, 0, 8, 0, 0,
9, 0, 0, 0, 0, 0, 0,10, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,11,10,12, 0, 3, 0, 0, 0, 0,
0, 0, 0, 0, 0,13,11, 0,14,12, 0, 0, 0, 0,
0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,13, 0, 0,15, 5, 9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 4, 0, 0,
0, 0, 0, 0, 0, 0, 0,14,15, 0, 0, 0, 0, 0
```

2. 輸出:



3. CNF 數: 34723

三、無解之情形

(一)、條件不完整

當一個 Flow Free 遊戲所提供的條件不足(不完整)時，就無法被解開。以下列這個 Flow Free 為例：

```
0, 0, 1, 0
0, 1, 0, 0
0, 3, 2, 0
0, 0, 0, 2
```

由於這個 Flow Free 所提供的起始與終點格並沒有全部兩兩成對(標號 3 的格子)，因此 CNF 的條件會不完整，造成 SAT Solver 無法對 CNF 進行判斷。

(二)、條件互相矛盾

當一個問題所提供的條件彼此互相矛盾時，會出現無解的狀況。

以下列這個 Flow Free 為例：

```
3, 0, 1, 2
0, 0, 2, 1
0, 0, 0, 0
0, 0, 0, 3
```

此 Flow Free 遊戲中標號 2 的格子無法進行連線，條件出現互相矛盾，因此這個 Flow Free 遊戲無解。

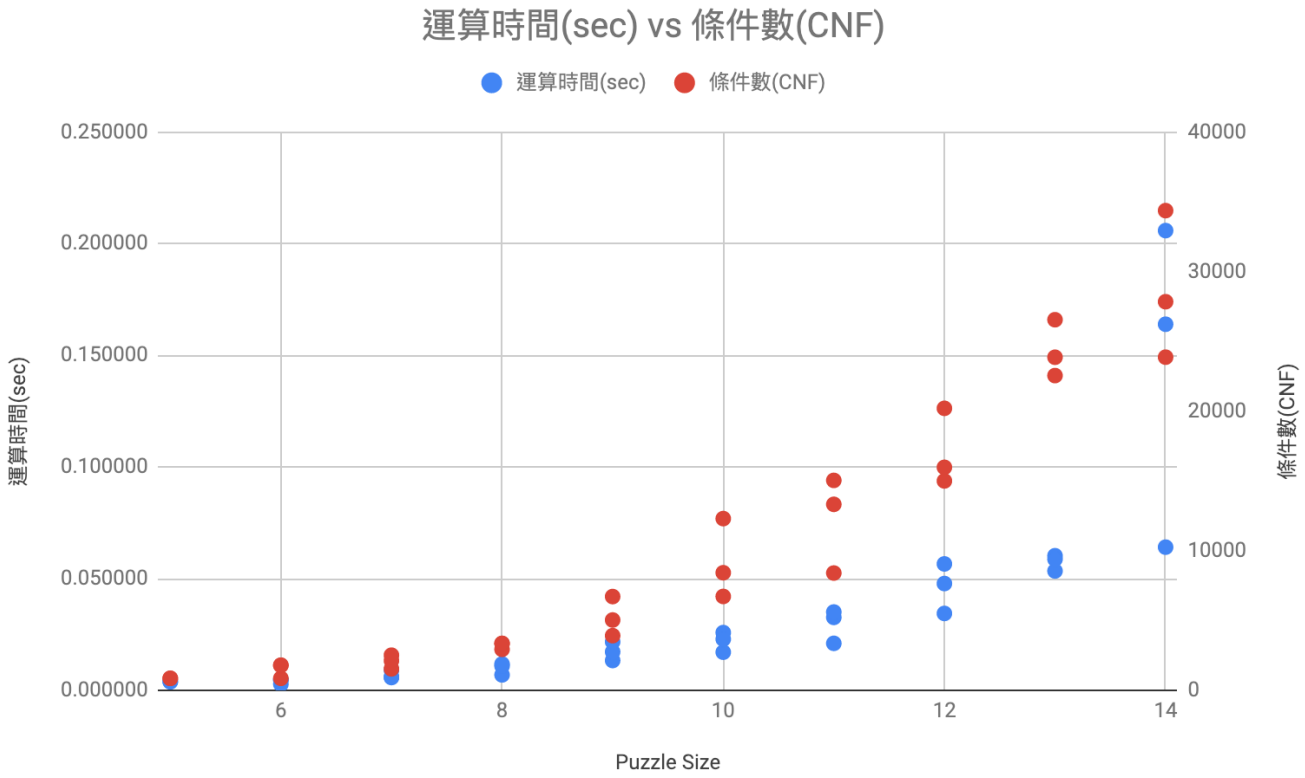
陸、討論

一、運算時間

(一)、5x5~14x14 的結果

我們在 Flow Free 的程式中選了 5x5 到 14x14 各 3 個關卡去解，將結果記錄並做了整理。

Puzzle Size	運算時間(秒)	條件數(CNF)
5	0.003900	864
5	0.003876	863
5	0.002755	863
6	0.004590	1791
6	0.004829	1805
6	0.005747	1541
7	0.009029	2521
7	0.006078	2127
7	0.006930	2937
8	0.010902	3354
8	0.011800	3368
8	0.013377	3920
9	0.017249	6719
9	0.021728	5040
9	0.017088	6729
10	0.025868	12303
10	0.022942	8425
10	0.021053	8404
11	0.032673	15040
11	0.035083	13324
11	0.034429	15002
12	0.047828	20201
12	0.056616	15976
12	0.053452	22547
13	0.060272	26547
13	0.058736	23860
13	0.064113	23866
14	0.163909	34361
14	0.205814	27836
14	0.088295	34360



從上面幾個表格我們可以發現用 SAT Solver 在解 5x5 到 14x14 的 Flow Free 時，不論題目的大小、顏色數有怎麼樣的改變，運算速度都可以維持在 1 秒以內。

二、延伸應用

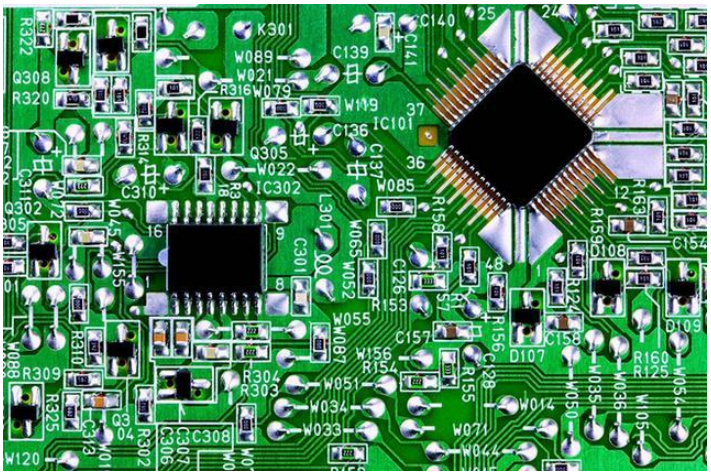
(一)、自動倉儲

由上述的研究我們可以得知 SAT Solver 的運算十分快速，使程式可以運用在即時運算上面，例如：自動倉儲管理的無人車路線規劃。現在網路購物與郵件寄送越來越進步，隨著使用人數的增加，物流公司每天都必須處理成千上萬的貨物。許多國際知名的物流公司都已經在使用「自動倉儲系統」來增加效率以及減少人為失誤。這種系統需要及時運算出每一台機器人的路線規劃，而起點終點的連線、即時運算、以及路線不衝突這三個重要的特性，正好符合 SAT Solver 的特點。

▲阿里巴巴-菜鳥物流的自動倉儲機器人　　圖片來源：https://itw01.com/2TVRNEG.html

(二)、繪製電路板

所有電子產品中都少不了一個重要元素一電路板。電路板上精細的電線不能交叉，否則便會造成短路。現在的社會充斥著各種運用到電路板的電子產品，能夠有效率地進行大量生產是各大電子廠商的主要目標之一。電路板上的電線佈局有點與點的連線和不交錯這兩個特性，與 Flow Free 非常相似，因此我們認為可以用 SAT Solver 去解開「如何繪製出電路板」的問題。



▲電路板示意圖　　圖片來源：https://itw01.com/XZS2EOO.html

柒、結論

一、經過研究與比較，我們發現數獨與 Flow Free 共同擁有的特徵有：

(一)無步驟性

(二)所有線索都能以多個是非條件式表示

從以上結果我們可以推斷，當一個問題具備上述的特徵時便能夠使用 CNF 表示法(Conjunctive Normal Form)將所有條件與線索列出，進而可以用 SAT Solver (Boolean Satisfiability Problem Solver 或作 Propositional Satisfiability Problem Solver)解開。

二、無解

當一個 Flow Free 遊戲的條件不完整(例如缺少某些格子的答案)或彼此會相互矛盾(例如得到的線索結果為第一格為 1、第一格為 2 時)，會導致程式判斷錯誤，產生無解情況。

捌、參考資料

1. A SAT-based Sudoku Solver by Tjark Weber
2. THE COMPLEXITY OF SATISFIABILITY PROBLEMS by Thomas J. Schaefer
3. Introduction to Boolean Algebra
https://www.allaboutcircuits.com/textbook/digital/chpt-7/introduction-boolean-algebra
4. Understanding SAT by Implementing a Simple SAT Solver in Python
https://sahandsaba.com/understanding-sat-by-implementing-a-simple-sat-solver-in-python.html
5. SAT problem 介紹
https://willyc20.github.io/2016/12/17/sat-problem-1
6. NP 問題
https://www.csie.ntu.edu.tw/~sprout/algo2016/homework/week10.pdf
7. Carnegie Melon University : DPLL-based SAT solvers
8. Flow Free redux: eating SAT-flavored crow
https://mzucker.github.io/2016/09/02/eating-sat-flavored-crow.html

【評語】032803

1. SAT 相關研究，是資訊科學非常基礎的理論研究。國中生瞭解 SAT (Satisfiability)，並進行相關鑽研，殊屬不易，值得鼓勵。

2. 作品說明書前半段，解釋以 SAT solver 解決數獨的方法，將變數的物理意義解釋得很清楚。但是，後半段，解釋以 SAT solver 解決 Flow Free 的方法，則不像數獨解釋的那麼清楚，較為可惜。而且中間參雜程式碼，較不易閱讀。

3. 作品說明書有說明，此項研究可應用於自動倉儲與電路板設計，若能點出遊戲與實際應用之異同，則更為完善。

作品海報

壹、研究動機

我們在Google Play 商店中發現了一個名為Flow Free的遊戲。這個遊戲的規則是在不交叉的情況下把相同顏色的點點連接起來，即可過關。Flow Free以簡單的規則引起我們的興趣。一開始單憑直覺一步一步的破關方法，隨著關卡越來越後面，版面與需連線的點逐漸增加，漸漸變得窒礙難行，幾乎沒辦法在不使用提示的情況下將遊戲解出來。這讓我們想研究這個遊戲，我們希望能透過電腦、演算法等等來找出一種可以解開所有關卡的方式。

貳、研究目的

一、研究其他非步驟性遊戲以及已知的演算法嘗試推導出可以解出Flow Free的方法

二、找出是否所有關卡都有解

參、研究設備及器材

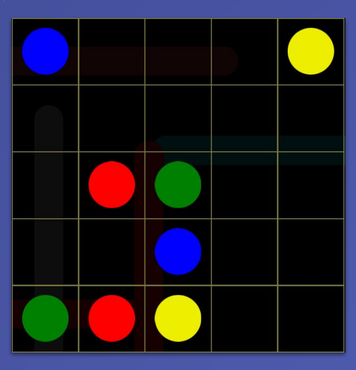
一、電腦 (Python 程式語言)

二、手機 (Flow Free 應用程式)

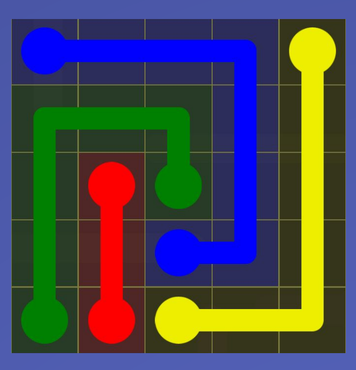
肆、研究過程與方法

一、Flow Free與數獨

(一)、何謂Flow Free



▲遊戲的初始樣貌



▲遊戲解開後的樣貌

這是一個Flow Free遊戲的範例。解開Flow Free遊戲的方法是將所有顏色相同的點兩兩連線。Flow Free的規則如下：

1. 連線不可超出遊戲邊界
2. 連線不可交叉重疊
3. 不同顏色的兩點不可連線
4. 連線不可通過有其他顏色的點的格子

由以上規則可以延伸出許多的條件，以供演算法進行判斷。

(二)、Flow Free與數獨之相似處

在這裡我們可以發現，Flow Free許許多多的規則，導致遊戲中的連線方法與每格的顏色固定，先進行哪一個步驟並不會影響其他步驟的進行，也就是代表Flow Free具有每一個步驟不會互相影響的「非步驟性」的性質。數獨中每一格可填入的答案固定，因此數獨也擁有這個特徵。

二、布林可滿足性問題

(一)、定義

布林可滿足性問題(Boolean Satisfiability Problem，簡稱SAT)是一個用於確認布林運算式是否可以滿足所有給定條件的方法。若布林運算式內的所有布林變數均可以被賦值，且被給定的值都能夠滿足所有條件，則該問題為有解;反之則為無解。

(二)、SAT Solver

SAT Solver是一套用於解開可滿足性問題的演算法。此演算法會持續的將每個布林變數賦予一個值True(是)或False(非)，直到有一組解滿足所有給予的條件。若沒有一組解可滿足所有的條件，則該問題為無解。將用CNF表示的布林變數丟給SAT Solver運算後，即可得到該問題的解答。以下列這個CNF為例:

[[1, -5, 4], [-1, 5, 3, 4], [-3, -4], [2, 3]]

經過SAT Solver運算後，即可得到以下的結果:

[1, 2, 3, -4, -5]

三、何謂CNF

CNF(Conjunctive Normal Form) 是一種用於表示布林變數(Boolean Variable)的方法。一個布林變數(v)只有兩種可能的值 : True(T) 或者 False(F)。若 v = T，則 ¬v= F。CNF表示法中，有文字(literal)、子句(clause)跟CNFs。

- 一個文字(literal) l 的形式為 v 或 ¬v
- 一個子句(clause) c 的形式為 $(l_1 \vee l_2 \vee ... \vee l_n)$
- 一個CNFs的形式為 $c_1 \wedge c_2 \wedge \ldots \wedge c_n$

*注 : 在布林邏輯中，∨為「或 (or)」，∧為「且 (and)」，¬為「非 (not)」

舉例來說，(A ∨ B) ∧ (C ∨ ¬D)是CNF ;

((A ∨ B) ∧ (C ∨ ¬D)) ∧ ((¬E ∨ F) ∧ (G ∨ ¬H))是CNFs。

另外，在布林邏輯中，¬(A ∧ B)=(¬A ∨ ¬B)也是個重要的觀念。

四、以迷你數獨為例

這裡我們用一個3×3的迷你數獨來解釋如何用SAT Solver解數獨，題目如右圖：

1		
		2
	3	

(一)、定義變數

首先我們先定義格子。由左上角到右下角分別為第一格~第九格。如下圖：

一	二	三
四	五	六
七	八	九

我們再來定義27個變數。變數1代表數獨中第一格的數字為1、變數2代表數獨中第一格的數字為2、變數3代表數獨中第一格的數字為3、變數4代表數獨中第二格的數字為1...。

(二)、依據題目取得條件

1. 題目給予的線索

```
CNF = [[1], [17], [24]]
```

[1]代表第一格為1、[17]代表第六格為2、[24]代表第八格為3。

2. 每格至少有一個數字為正確答案

```
CNF = [[1,2,3], [4,5,6], [7,8,9],
[10,11,12], [13,14,15], [16,17,18],
[19,20,21], [22,23,24], [25,26,27]]
```

[1,2,3]代表第一格可能為1或2或3、[4,5,6]代表第二格可能為1或2或3、以此類推，[25,26,27]代表第九格可能為1或2或3。

3. 每格只能有一個數字為正確答案

```
CNF = [[-1,-2], [-1,-3], [-2,-3],
[-4,-5], [-4,-6], [-5,-6],
[-7,-8], [-7,-9], [-8,-9],
[-10,-11], [-10,-12], [-11,-12],
[-13,-14], [-13,-15], [-14,-15],
[-16,-17], [-16,-18], [-17,-18],
[-19,-20], [-19,-21], [-20,-21],
[-22,-23], [-22,-24], [-23,-24],
[-25,-26], [-25,-27], [-26,-27]]
```

[-1,-2]代表第一格不能同時為1和2、[-1,-3]代表第一格不能同時為1和3、[-2,-3]代表第一格不能同時為2和3，

所以[-1,-2],[-1,-3],[-2,-3]代表第一格的答案只能是1、2、3其中一個、[-4,-5],[-4,-6],[-5,-6]代表第二格的答案只能是1、2、3其中一個，以此類推。

4. 每一列至少有一個格子為1、2、3

```
CNF = [[1,4,7], [2,5,8], [3,6,9],
[10,13,16], [11,14,17], [12,15,18],
[19,22,25], [20,23,26], [21,24,27]]
```

[1,4,7]代表第一列至少有一格為1、[2,5,8]代表第一列至少有一格為2、[3,6,9]代表第一列至少有一格為，以此類推。

5. 每一行至少有一個格子為1、2、3

```
CNF = [[1,10,19], [2,11,20], [3,12,21],
[4,13,22], [5,14,23], [6,15,24],
[7,16,25], [8,17,26], [9,18,27]]
```

[1,10,19]代表第一行至少有一格為1、[2,11,20]代表第一行至少有一格為2、[3,12,21]代表第一行至少有一格為3，以此類推。

6. 每一列只能有一個格子為1、2、3

```
CNF = [[-1,-4],[-1,-7],[-4,-7],
[-2,-5],[-2,-8],[-5,-8],
[-3,-6],[-3,-9],[-6,-9],
[-10,-13],[-10,-16],[-13,-16],
[-11,-14],[-11,-17],[-14,-17],
[-12,-15],[-12,-18],[-15,-18],
[-19,-22],[-19,-25],[-22,-25],
[-20,-23],[-20,-26],[-23,-26],
[-21,-24],[-21,-27],[-24,-27]]
```

[-1,-4]第一列的第一格和第二格不可同時為1、

[-1,-7]第一列的第一格和第三格不可同時為1、

[-4,-7]第一列的第二格和第三格不可同時為1，

所以[-1,-4],[-1,-7],[-4,-7]代表第一列只能有一個格子為1、

[-2,-5],[-2,-8],[-5,-8]代表第一列只能有一個格子為2、

[-3,-6],[-3,-9],[-6,-9]代表第一列只能有一個格子為3，

以此類推。

7. 每一行只能有一個格子為1、2、3

```
CNF = [[-1,-10],[-1,-19],[-10,-19],
[-2,-11],[-2,-20],[-11,-20],
[-3,-12],[-3,-21],[-12,-21],
[-4,-13],[-4,-22],[-13,-22],
[-5,-14],[-5,-23],[-14,-23],
[-6,-15],[-6,-24],[-15,-24],
[-7,-16],[-7,-25],[-16,-25],
[-8,-17],[-8,-26],[-17,-26],
[-9,-18],[-9,-27],[-18,-27]]
```

[-1,-10]第一行的第一格和第二格不可同時為1、
[-1,-19]第一行的第一格和第三格不可同時為1、
[-10,-19]第一行的第二格和第三格不可同時為1,
所以[-1,-10],[-1,-19],[-10,-19]代表第一行只能有一個格子為1、
[-2,-11],[-2,-20],[-11,-20]代表第一行只能有一個格子為2、
[-3,-12],[-3,-21],[-12,-21]代表第一行只能有一個格子為3,
以此類推。

8. 將以上所有條件交由SAT Solver去解, 得到的答案為:

```
[1, -2, -3,    -4, 5, -6,     -7, -8, 9,
-10, -11, 12,  13, -14, -15,  -16, 17, -18,
-19, 20, -21,  -22, -23, 24,  25, -26, -27]
```

也就是:

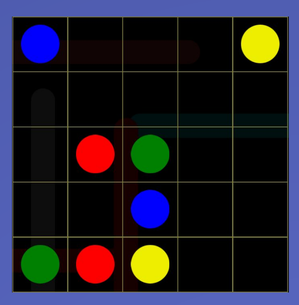
1	2	3
3	1	2
2	3	1

伍、研究結果

一、解開Flow Free

(一)、定義Flow Free

研究完迷你數獨後, 我們用相同將Flow Free的格子以編號0~m的方式表示, 每一個格子有dColor+4種可能性(總顏色數+4種方向)。因此在一個完整的Flow Free中, 總共會有(m + 1)*(dColor + 4)種可能性。我們以下列這個5x5的Flow Free為預設題目:



(二)、條件設置

必須要有初始顏色格, 電腦再依下列條件判斷出其他格的顏色及方向。

1. 每一格至少且只能有一種顏色
2. 每個非終點格至少且只能有一個連出方向
3. 每個起始格與終點格鄰近至少且只能有一個與它相同顏色的格子
4. 每個非終點格必須與它所連向的格顏色相同
5. 兩個格子不得互相連向
6. 方向不可指出邊界外
7. 方向不可指向起始格
8. 每個非起始格至少且只能被一個格子連向

(三)、演算法解析

1. 定義變數的表示方式

(1)以二維座標(x, y)的方式將條件變數輸入至CNF中

```
def xyVar(n, dMax, x, y, d):
    return dMax * (n * y + x) + d
```

(2)以一維座標(m)的方式將條件變數輸入至CNF中

```
def mVar(dMax, m, d):
    return dMax * m + d
```

2. 將Flow Free轉換為CNF

(1)變數宣告

a. 函數定義 `def getClauses(flowfree):`
b. Flow Free的總格子數 `m = len(flowfree)`
c. Flow Free的行/列格子數 `n = int(m ** 0.5)`
d. Flow Free的總顏色 `dColor = 0`
e. Flow Free的總變化數 `dMax = 0`
f. CNF clauses `clauses = []`
g.起始點和終點 `terminalPoints = []`
h.起始點 `beginPoints = []`
i.起始點的顏色 `valBeginPoints = []`
j.起始點的座標 `idxBeginPoints = []`
k.非終點格的座標 `idxNonEndPoints = []`

(2)取得初始變數

a. 取得Flow Free的總顏色數 `dColor = max(flowfree)`
b.取得Flow Free的總變化數 `dMax = dColor + 4`
c.取得Flow Free的起始格、終點格以及非終點格的座標

```
for i in range(m):
    if flowfree[i] != 0:
        terminalPoints += [mVar(dMax, i ,flowfree[i])]
    if flowfree[i] != 0 and flowfree[i] not in valBeginPoints:
        beginPoints += [mVar(dMax, i, flowfree[i])]
        idxBeginPoints += [i]
        valBeginPoints += [flowfree[i]]
    if flowfree[i] == 0:
        idxNonEndPoints += [i]
        idxNonEndPoints += idxBeginPoints
        idxNonEndPoints.sort()
```

d.將題目原有顏色的格子輸入至CNF中

```
for i in range(m):
        if flowfree[i] != 0:
        clauses += [[mVar(dMax, i, flowfree[i])]]
```

CNF : [[1], [34], [91], [100], [137], [164], [171], [178]] (共8個)

(3)條件設置(詳細程式請參照科展報告書)

a.每一格至少要有一種顏色
CNF : [[1, 2, 3, 4], [9, 10, 11, 12], …, [193, 194, 195, 196]] (共25個)
b.每一格只能有一種顏色
CNF : [[-1, -2], [-1, -3], …, [-195, -196]] (共150個)
c.每一個非終點格至少要有一個連出方向
CNF : [[5, 6, 7, 8], [13, 14, 15, 16], …, [197, 198, 199, 200]] (共21個)
d.每一個非終點格只能有一個連出方向
CNF : [[-5, -6], [-5, -7], …, [-199, -200]] (共126個)
e.每個起始格與終點格鄰近至少有一個與它相同顏色的格子
CNF : [[9, 41], [26, 74], ..., [170, 186, 138]] (共8個)
f.每個起始格與終點格鄰近只能有一個與它相同顏色的格子
CNF : [[9, 41], [-9, -41], …, [-186, -138]] (共27個)
g.每個非終點格必須與它所連向的格顏色相同
CNF : [[-6, -1, 41], [-8, -1, 9], ..., [-199, -196, 188]] (共272個)
h.兩個格子不得互相連向)
CNF : [[-45, -6], [-85, -46], ..., [-192, -199]] (共80個)
i.連出方向不可連向邊界
CNF : [[-5], [-166], [-7], [-40], [-13], [-174], [-47], [-80], [-21], [-182], [-87], [-120], [-29], [-190], [-127], [-160], [-37], [-198], [-167], [-200]] (共20個)
j.連出方向不可連向起始格
CNF : [[-15], [-45], [-32], [-77], [-88], [-103], [-54], [-133], [-96], [-111], [-62], [-141]] (共12個)
k.每個非起始格至少被一個格子連向)
CNF : [[8, 23, 53], [16, 31, 61], ..., [192, 158]] (共21個)
l.每個非起始格只能被一個格子指向
CNF : [[-8, -23], [-8, -53], ..., [-192,-158]] (共80個)
3.轉換\輸出函式(詳細程式請參照科展報告書)
4.程式主體

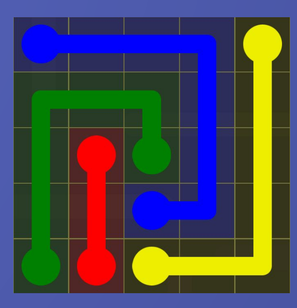
(1)輸入要解開的題目

```
flowfree = [1, 0, 0, 0, 2, \
            0, 0, 0, 0, 0, \
            0, 3, 4, 0, 0, \
            0, 0, 1, 0, 0, \
            4, 3, 2, 0, 0]
```

(2)將題目丟至函數內並輸出
(詳細程式請參照科展報告書)

(四)、結果

將我們設計的演算法得出的CNF丟給SAT Solver運算後, 再經過轉換輸出函式, 即可得到以下的結果 :



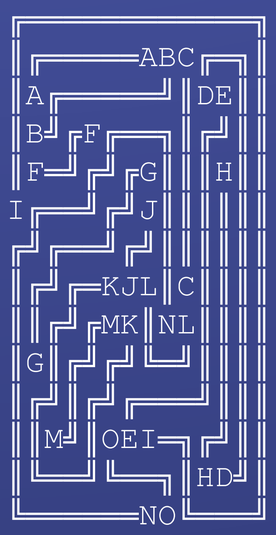
二、運算結果

經過測試, 5x5至14x14的題目皆可用我們設計的程式解開。
下面是我們的程式解開一個14x14的Flow Free遊戲的例子。

(一)、題目

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 4, 5, 0, 0,
0, 2, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 6, 0, 0, 0, 0, 0, 7, 0, 0, 0, 8, 0, 0,
9, 0, 0, 0, 0, 0, 0,10, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,11,10,12, 0, 3, 0, 0, 0, 0,
0, 0, 0, 0, 0,13,11, 0,14,12, 0, 0, 0, 0,
0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,13, 0, 0,15, 5, 9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 4, 0, 0,
0, 0, 0, 0, 0, 0, 0,14,15, 0, 0, 0, 0, 0
```

(二)、輸出



(三)、CNF數: 20132

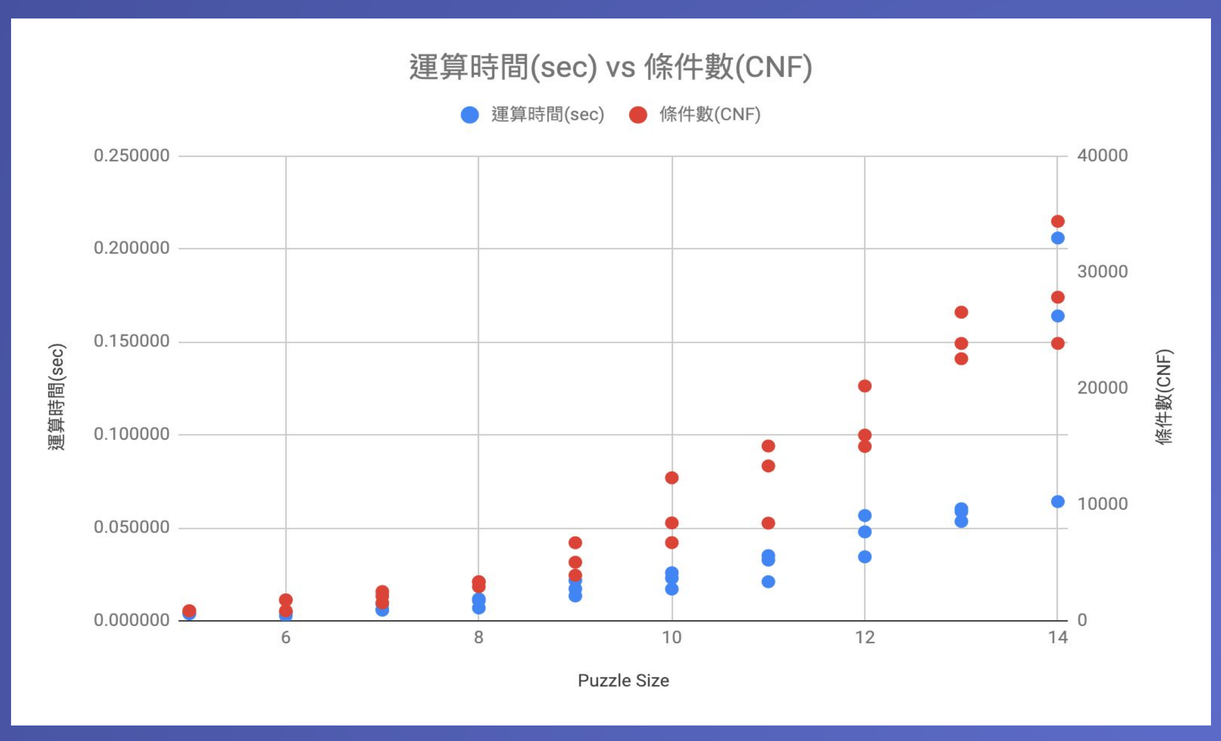
陸、討論

一、運算時間

(一)、5x5~14x14的結果

我們在Flow Free的程式中選了5x5到14x14各3個關卡去解，將結果記錄並做了整理。從表格我們可以發現用SAT Solver在解5x5到14x14的Flow Free時，不論題目的大小、顏色數有怎麼樣的改變，運算速度都可以維持在1秒以內。

Puzzle Size	運算時間(秒)	條件數(CNF)	Puzzle Size	運算時間(秒)	條件數(CNF)
5	0.003900	864	10	0.025868	12303
5	0.003876	863	10	0.022942	8425
5	0.002755	863	10	0.021053	8404
6	0.004590	1791	11	0.032673	15040
6	0.004829	1805	11	0.035083	13324
6	0.005747	1541	11	0.034429	15002
7	0.009029	2521	12	0.047828	20201
7	0.006078	2127	12	0.056616	15976
7	0.006930	2937	12	0.053452	22547
8	0.010902	3354	13	0.060272	26547
8	0.011800	3368	13	0.058736	23860
8	0.013377	3920	13	0.064113	23866
9	0.017249	6719	14	0.163909	34361
9	0.021728	5040	14	0.205814	27836
9	0.017088	6729	14	0.088295	34360



二、延伸應用

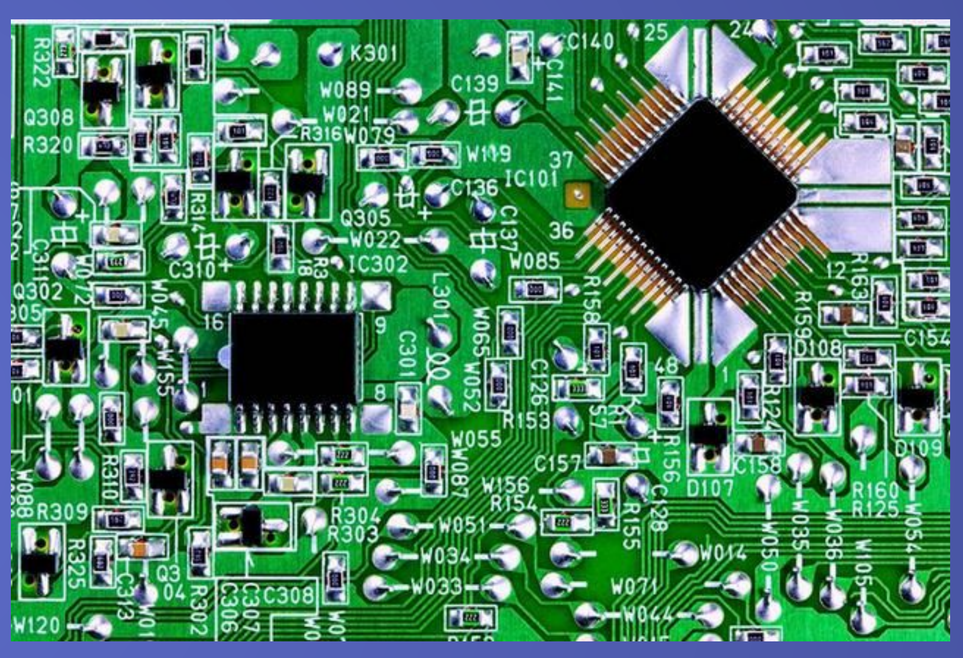
(一)、自動倉儲

由上述的研究我們可以得知SAT Solver的運算十分快速，使程式可以運用在即時運算上面。例如：自動倉儲管理的無人車路線規劃。現在網路購物與郵件寄送越來越進步，隨著使用人數的增加，物流公司每天都必須處理成千上萬的貨物。許多國際知名的物流公司都已經在使用「自動倉儲系統」來增加效率以及減少人為失誤。這種系統需要及時運算出每一台機器人的路線規劃，而起點終點的連線、即時運算、以及路線不衝突這三個重要的特性，正好符合SAT Solver的特點。

▲阿里巴巴-菜鳥物流的自動倉儲機器人　圖片來源：https://itw01.com/2TVRNEG.html

(二)、繪製電路板

所有電子產品中都少不了一個重要元素一電路板。電路板上精細的電線不能交叉，否則便會造成短路。現在的社會充斥著各種運用到電路板的電子產品，能夠有效率地進行大量生產是各大電子廠的主要目標之一。電路板上的電線佈局有點與點的連線和不交錯這兩個特性，與Flow Free非常相似，因此我們認為可以用SAT Solver去解開「如何繪製出電路板」的問題。



▲電路板示意圖　圖片來源：https://itw01.com/XZS2EOO.html

柒、結論

一、數獨與Flow Free共同的特徵：

(一)無步驟性

(二)所有線索都能以多個是非條件式表示

從以上結果我們可以推斷，當一個問題具備上述的特徵時便能夠使用CNF表示法 (Conjunctive Normal Form)將所有條件與線索列出，進而可以用SAT Solver (Boolean Satisfiability Problem Solver或作Propositional Satisfiability Problem Solver)解開。

二、無解

當一個Flow Free遊戲的條件不完整(例如缺少某些格子的答案)或彼此會相互矛盾(例如得到的線索結果為第一格為1、第一格為2時)，會導致程式判斷錯誤，產生無解情況。

(一)、條件不完整

當一個Flow Free遊戲所提供的條件不足(不完整)時，就無法被解開。以下列這個Flow Free為例：

```
0, 0, 1, 0
0, 1, 0, 0
0, 3, 2, 0
0, 0, 0, 2
```

由於這個Flow Free所提供的起始與終點格並沒有全部兩兩成對(標號3的格子)，因此CNF的條件會不完整，造成SAT Solver無法對CNF的線索進行判斷。

(二)、條件互相矛盾

當一個問題所提供的條件彼此互相矛盾時，會出現無解的狀況。

以下列這個Flow Free為例：

```
3, 0, 1, 2
0, 0, 2, 1
0, 0, 0, 0
0, 0, 0, 3
```

此Flow Free遊戲中標號2的格子無法進行連線，條件出現互相矛盾，因此這個Flow Free遊戲無解。

捌、參考資料

1. A SAT-based Sudoku Solver by Tjark Weber
2. THE COMPLEXITY OF SATISFIABILITY PROBLEMS by Thomas J. Schaefer
3. Introduction to Boolean Algebra https://www.allaboutcircuits.com/textbook/digital/chpt-7/introduction-boolean-algebra
4. Understanding SAT by Implementing a Simple SAT Solver in Python https://sahandsaba.com/understanding-sat-by-implementing-a-simple-sat-solver-in-python.html
5. SAT problem 介紹 https://willyc20.github.io/2016/12/17/sat-problem-1
6. NP 問題 https://www.csie.ntu.edu.tw/~sprout/algo2016/homework/week10.pdf
7. Carnegie Melon University : DPLL-based SAT solvers
8. Flow Free redux: eating SAT-flavored crow https://mzucker.github.io/2016/09/02/eating-sat-flavored-crow.html