中華民國第 63 屆中小學科學展覽會

作品說明書

高中組　電腦與資訊學科

第二名

052511

箇中「橋」楚—研究不同演算法對數橋遊戲的差異性

學校名稱：高雄市立高雄高級中學

作者：	指導老師：
高二 鄭勝宇 高二 郭典丞 高二 謝昀翰	李青育

關鍵詞：數橋、窮舉策略、演算法

摘要

本研究提出有關數橋謎題遊戲的程式實作辦法，我們已知數橋被證明是個 NP-complete 問題。我們研製出了數種解題條件與窮舉策略在演算法上的實現，並且比較了這幾種解決辦法的相對效率，再者也提出了數個 NP 問題的解決策略。本研究發現在進行窮舉的時候以取最小頂點值期望效率最高(表示該程式能有相對多，相對快的執行結果)。

壹、前言

一、研究動機

在學習演算法的過程中，我們習學到利用程式的演算法可以去解決很多有趣的問題，其中也包含一些數學謎題，例如：數獨、八皇后問題等。數學謎題有很多，我們好奇的想找出可以解出各式數學謎題的不同的程式，並且對那些程式進行分析了解不同演算法對於解題的影響，也想藉由分析程式中的項目來更加了解這些數學遊戲的特性。

在琳瑯滿目的數學遊戲中，我們選擇了數橋(Hashiwokakero)這個遊戲，因為他是由數個點和邊組成的遊戲，玩法相當獨特，也使我們聯想到曾經學過有關圖論的演算法。

二、研究目的

(一) 找出能夠解出數橋的程式

(二) 比較不同策略與演算法在解決數橋遊戲上的執行效率

(三) 比較不同的窮舉策略

三、文獻回顧

本研究主要是利用演算法來解決數橋遊戲，因此在此介紹與數橋相關的資訊以及一些我們會使用的演算法及資料結構。

(一)　數橋的歷史

數橋首次出現在 Nikoli 所發行的智力遊戲雜誌《パズル通信ニコリ》中的第 31 期（1990 年秋刊）。

(二)　數橋規則介紹

數橋的題目通常設在矩形中，矩形中的圓圈會有數字，我們稱作「島嶼」遊戲的目標是要將「橋梁」連在正確的地方，其中有以下條件：

1. 橋梁必須要是直線，且連接兩座島嶼
2. 橋梁中間不可跨越任何島嶼
3. 橋梁不可跨越另一座橋
4. 兩座島嶼之間最多只可以連接兩座橋
5. 每座島嶼所連接的橋數必須與島嶼上的數字相等
6. 所有島嶼必須要是一個群體，舉例來說：

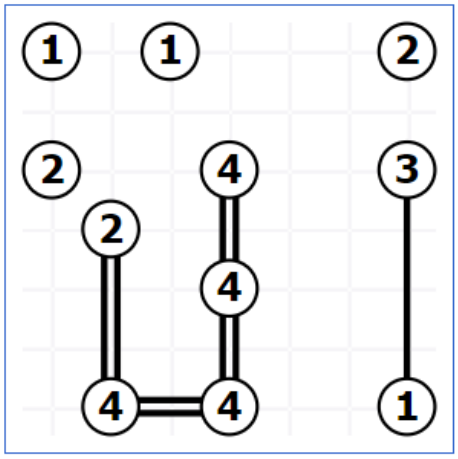
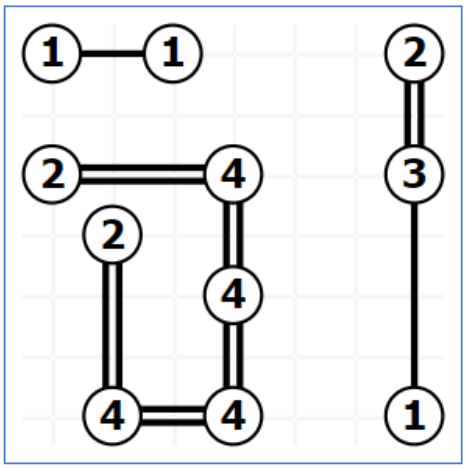


圖 1：需要用到連通性規則的情況

遇到這樣的情況時，我們有兩種連法會符合前 5 項規則



圖二：錯誤解

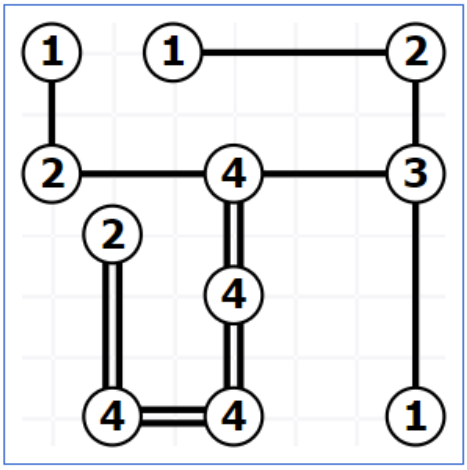


圖 3：正解

其中圖二將整個圖形分成了三個區塊，不符合規則 6，因此只有圖三是唯一正解

(三) P/NP 問題

P 與 NP 問題是在是計算複雜度理論中，決定性問題的等級之一。若有一群演算法用 DTM(deterministic Turing machine)來做計算所需時間是多項式時間(polynomial time)，那這類演算法或問題被稱為 P 問題；若有一群演算法用 NTM(non-deterministic Turing machine)來做計算所需時間是多項式時間，那這類問題被稱為 NP 問題。

數學理論指出任何一個 NP 裡面的問題都可以在多項式時間內，使用 DTM 化約成「一個布林方程式是否存在解」的問題，又將這個問題稱為 NP-complete 問題。

一般的電腦屬於 DTM，利用 DTM 來解 NP-complete 問題時需要花上指數時間以

上的複雜度，與 P 問題相差極大。雖然尚未有人證出 P 與 NP-complete 問題是否等價，不過目前普遍認為$P \neq NP - complete$。因此若一個問題是 NP-complete 問題，則常被認定為難題。很多的加密方法也是利用 NP-complete 問題難解的特性而產生。

而數橋問題已被證明出為 NP-complete 問題。

(四) DFS(depth-first search)

深度優先搜尋法，是用於圖論中遍歷整張圖或樹的方法，會盡可能以當前的點所連接到的點尋找，而當所有點皆搜尋過，則會回溯到先前的點繼續搜尋直到所有點皆被搜尋過。舉例來說：

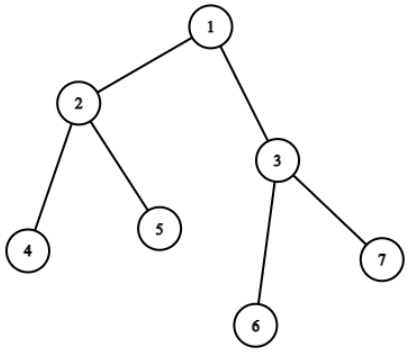


圖 4：圖(graph)

以這張圖為例，若 1 號點為起點，用 DFS 的遍歷順序為$1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 6 \rightarrow 7$。

(五) 並查集(Disjoint-set)

並查集是一種資料結構，主要作用為查詢元素存在於哪個子集，每個子集通常會將一個元素作為代表編號，其中並查集基本有以下之操作：

1. 查詢：查詢某個元素屬於哪個集合，通常是返回集合內的一個「代表元素」。這個操作是為了判斷兩個元素是否在同一個集合之中。

2. 合併：將兩個集合合併為一個。

3. 添加：添加一個新集合，其中有一個新元素。添加操作不如查詢和合併操作重要，常常被忽略。

貳、研究設備及器材

一、 硬體環境

(一) 桌機 PC2-046：

1. CPU：i3-7100

2. GPU：HD Graphics 630

(二) 軟體環境

1. Codeblocks 16.01

參、研究過程或方法

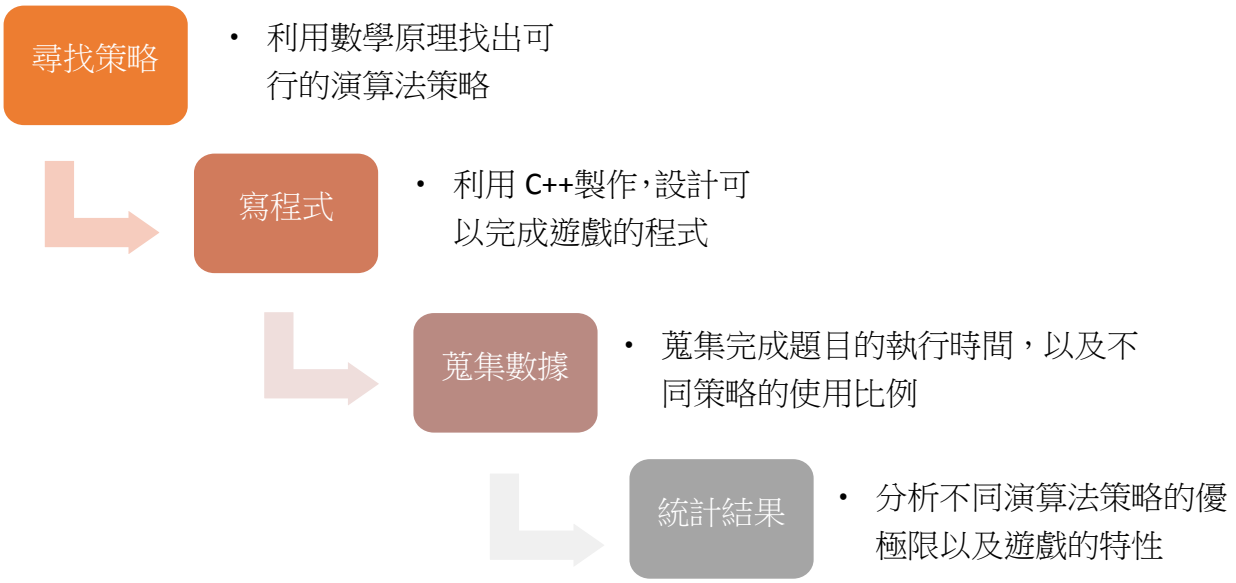


圖 5：研究流程圖

肆、研究結果

一、符號及名詞定義

為了讓式子變得整潔，在此先定義一些符號：

1. 頂點V：島嶼上的數字

2. 邊E：橋梁

3. 點群G：很多個以橋樑連接的島嶼群

4. 剩餘度數N(V)：島嶼上剩餘數字

5. 可連邊數量Way(V)：島嶼四周尚可以連邊的數量

6. 可連邊方向Adj(V)：該島嶼四周有其他可以連的島嶼數

二、條件解題

在由於規則的限制，在某些情況下我可以直接判斷出哪一條邊要連，哪一條邊不要連，以下是我們所找出來的條件：

(一) $N(V) = Way(V)$

若島上的數字等於剩餘可連的橋，可把所有橋連起來。

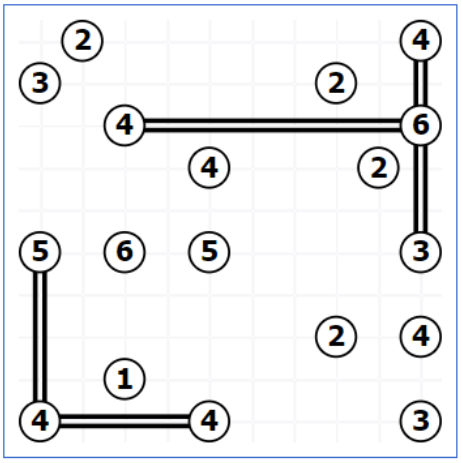


圖 6：策略(一)例圖

圖中島嶼 4 和島嶼 6 適用於這個策略

(二)$\mathrm{N(V)} > (\mathrm{Adj(V)} - 1) \times 2$

因為每個相鄰的島最多只能連兩座橋，由鴿籠原理，在這種情況下可以得出每一個相鄰的島嶼皆可連一條邊。

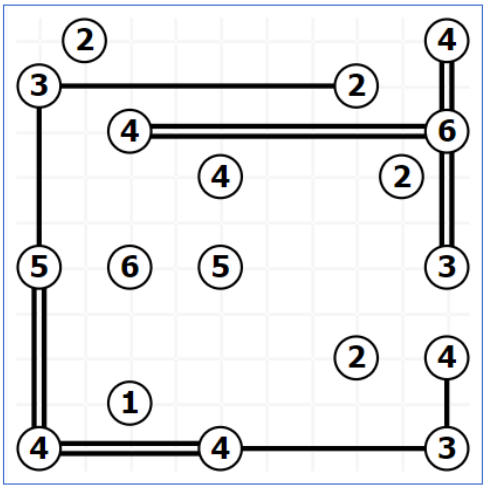


圖 7：策略(二)例圖

圖中左上的島嶼 3 和右下的島嶼 3 可以用於這個策略

(三)若V_0中的可連的點∃i 使得 $\mathrm{N}(V_i) = 1$，則可將V_0視為V_0'其中$\mathrm{N}(V_0') = N(V_0) - 1$,

$\mathrm{Adj}(V_0') = \mathrm{Adj}(V_0) - 1$

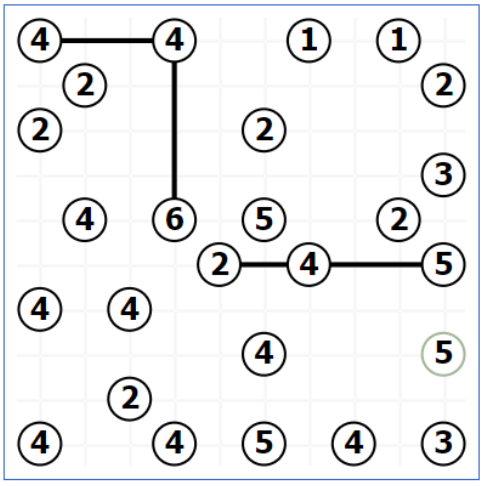


圖 8：策略(三)例圖

圖中的兩個島嶼 4 可以用於這個策略

(四)連通性問題

利用所有島嶼必須要是一個群體這個規則，能夠排除掉某些邊不可連：

1. 1 鄰 1 – X

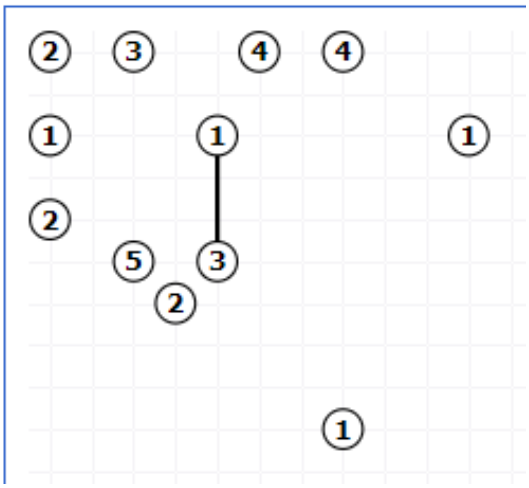


圖 9：策略(四) – 1 例圖

圖中中間的島嶼 1 適用於這個策略

2. 2 鄰 2 – X

因為不可能兩個邊都連接在相鄰的島嶼 2 上，因此必有一邊與另一相鄰島嶼相鄰

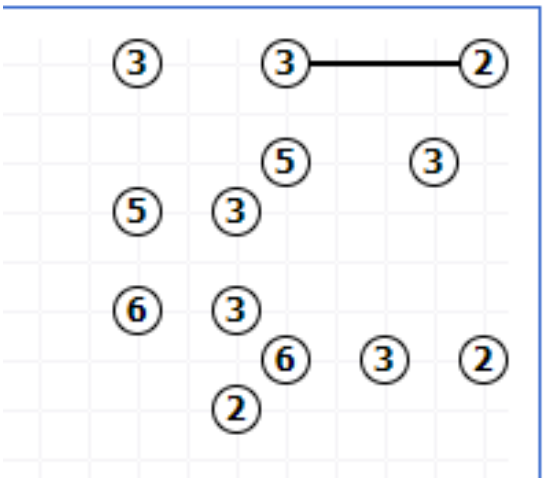


圖 10：策略(四) – 2 例圖

圖中的島嶼 2 適用於這個策略

3. 4 鄰 2 – 2 – X

理由同(四) - 2

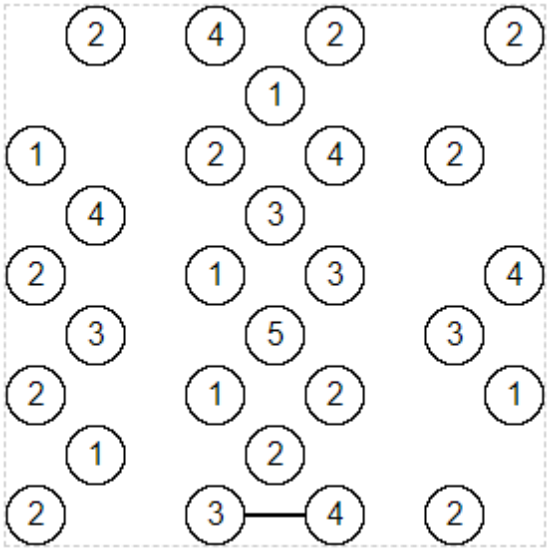


圖 10：策略(四) – 3 例圖

圖中的島嶼 4 適用於這個策略

4. 3 鄰 2 – 1 – X

理由同(四) - 2

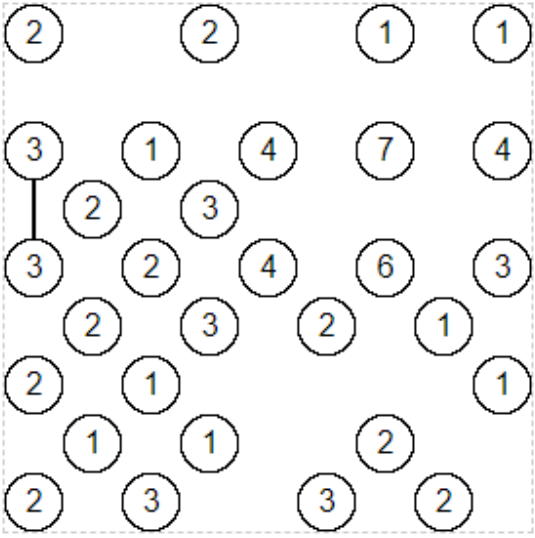


圖 11：策略(四) – 4 例圖

圖中的島嶼 3(上方)適用於這個策略

5. 2 鄰 1 – 1 – X

理由同(四) - 2

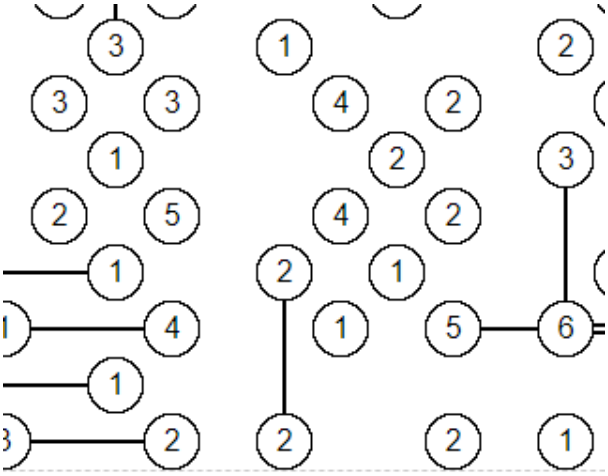


圖 12：策略(四) – 5 例圖

圖中間的島嶼 2 適用於這個策略

其他可以推出這個規則的還有 5 鄰 2 – 2 – 1 – X 等，不過經過人工測題過後仍沒發現這種情況。

三、窮舉策略

當所有的條件都使用過之後，仍有一些題目無法完成，這時就需要用到窮舉的方法。先假設性連出一條線，之後回到條件解題的方法，若遇到無法解開的情況則將假設的線擦掉在假設新的一條線，重複這些步驟直到完成遊戲。窮舉的方法有以下這些：取最大頂點值、取最小頂點值、取最大度數值、取最小度數值、取最大集合、取最小集合這六種方法。我們將會利用程式實測出從哪種方法對於成功率或執行時間是最好的。

四、程式實作

以下為本研究所使用的演算法之重要片段的流程圖

(一)純窮舉策略

1. 輸入輸出

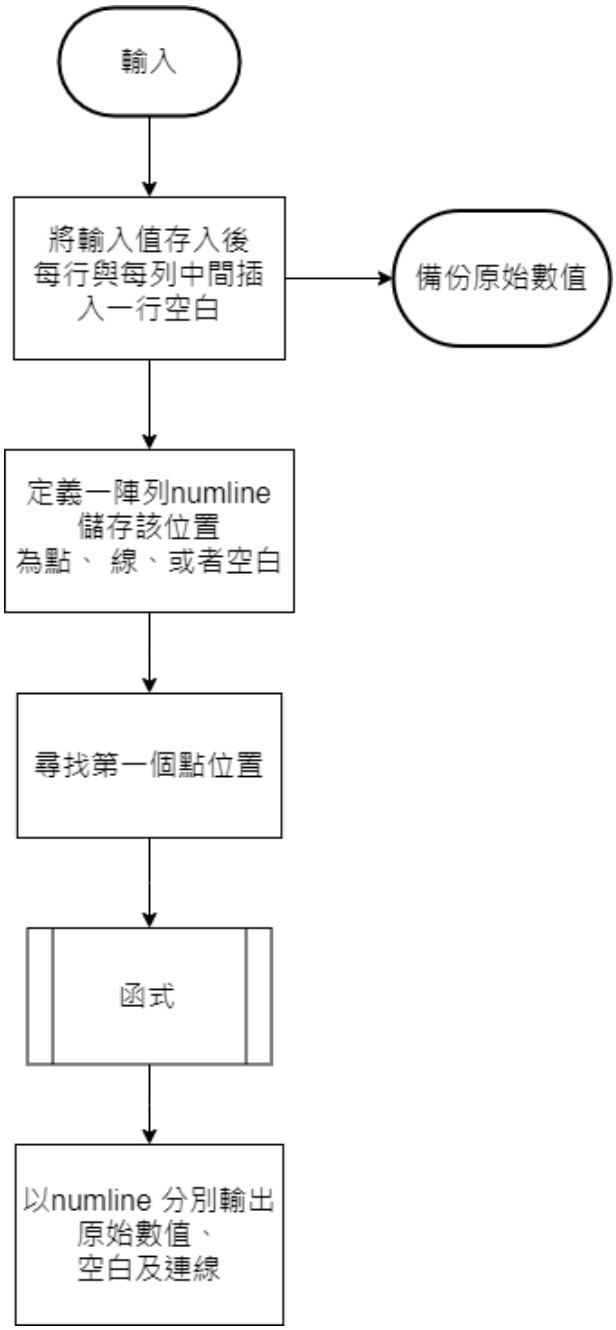


圖 13：DFS 輸入輸出流程圖

2. DFS 部分

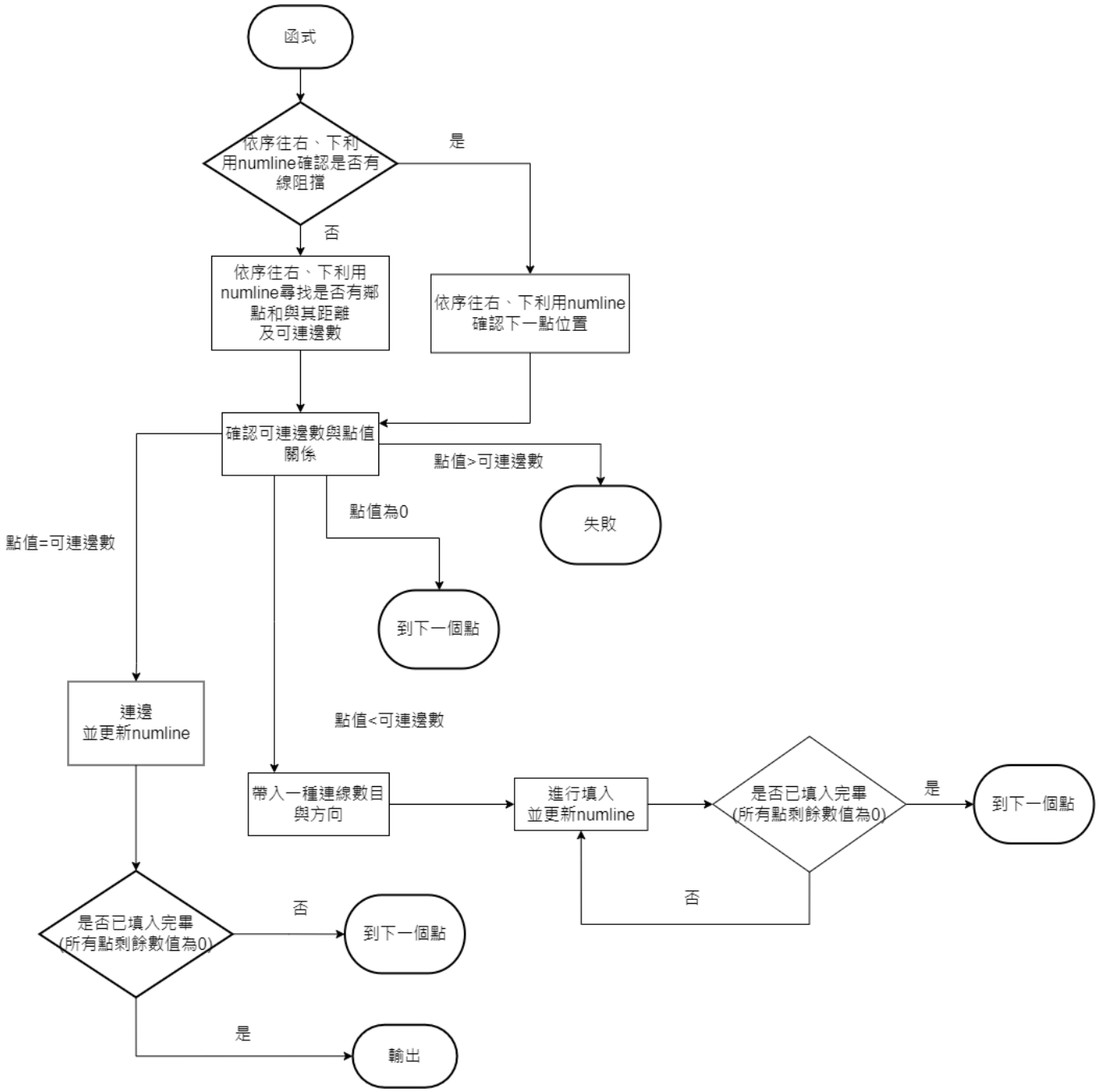


圖 14：DFS 函式流程圖

(二)「條件解題」 + 「窮舉策略」

1. 建立鄰接矩陣

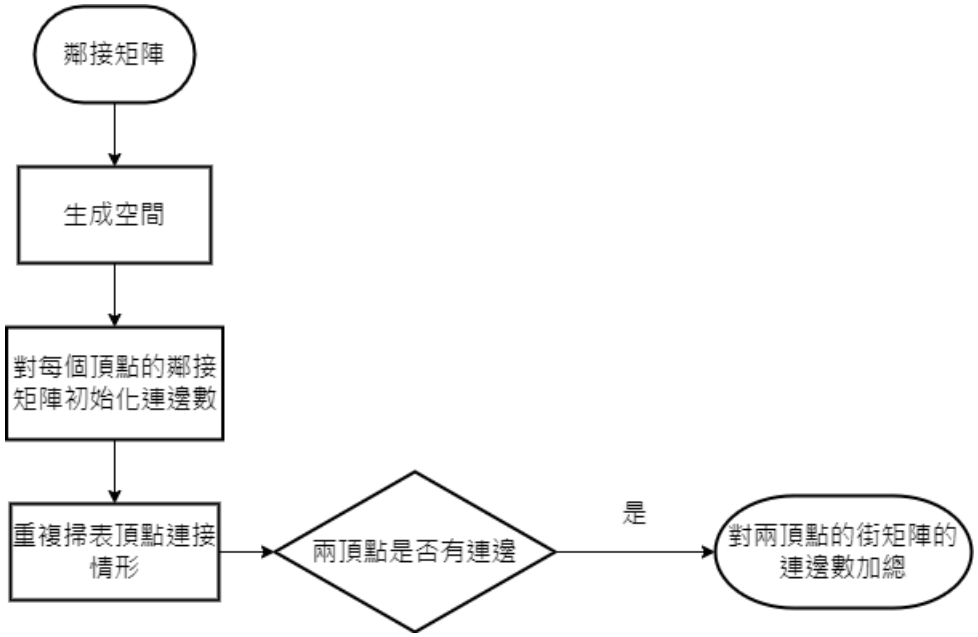


圖 15：鴿籠策略建立鄰接矩陣流程圖

2. 建立紀錄解答的平面

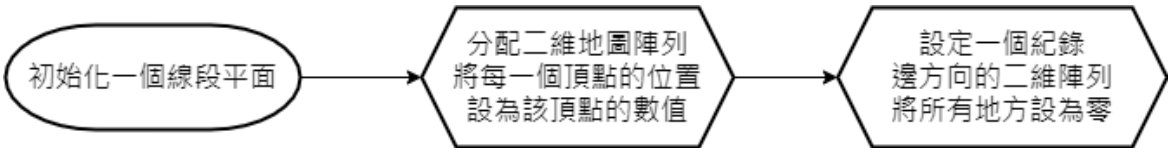


圖 16：鴿籠策略建立紀錄解答的平面流程圖

3. 並查集

(1). 初始化

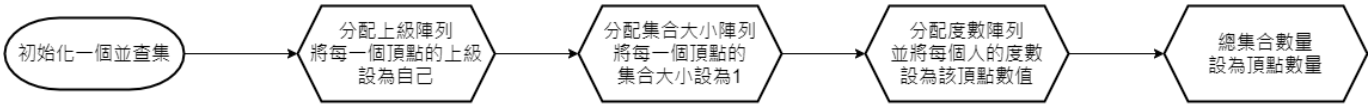


圖 17：鴿籠策略初始化並查集流程圖

(2). 查詢上級

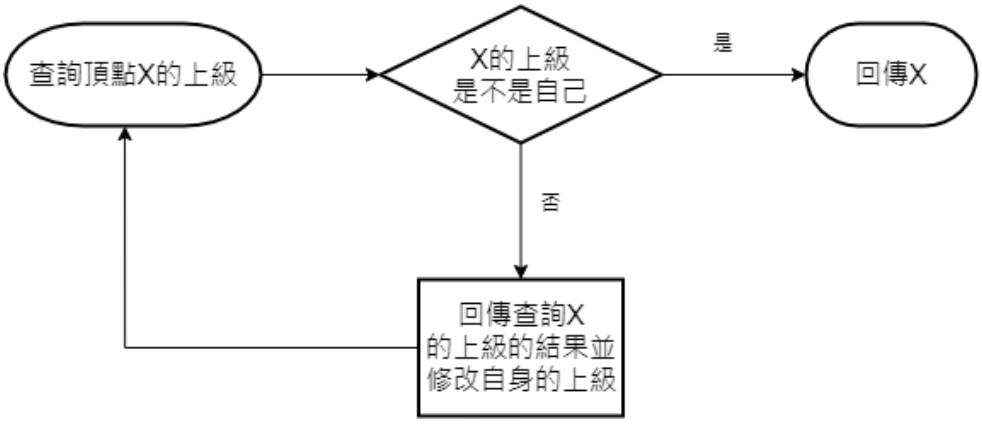


圖 18：鴿籠策略並查集查詢流程圖

(3). 合併集合

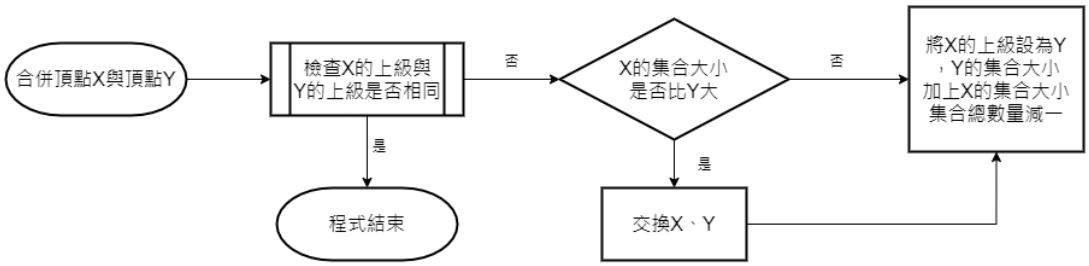


圖 19：鴿籠策略並查集合併流程圖

4. 頂點

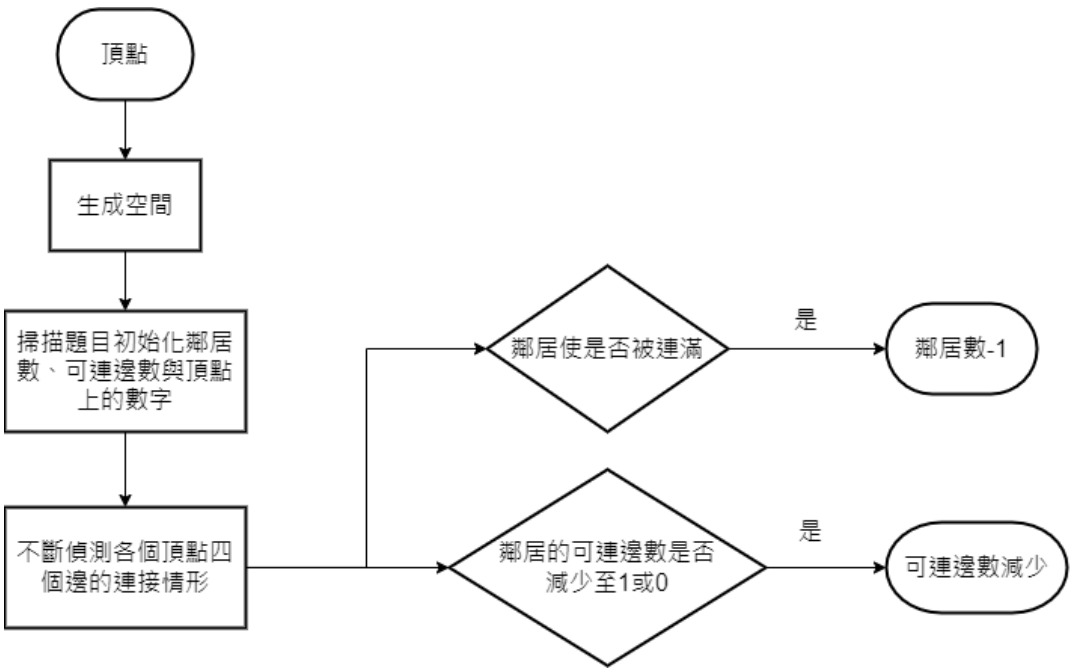


圖 20：鴿籠策略頂點操作流程圖

5. 建立圖

(1). 初始化

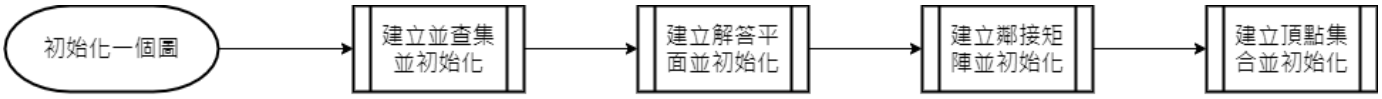


圖 21：鴿籠策略初始化圖流程圖

(2). 刪邊函式

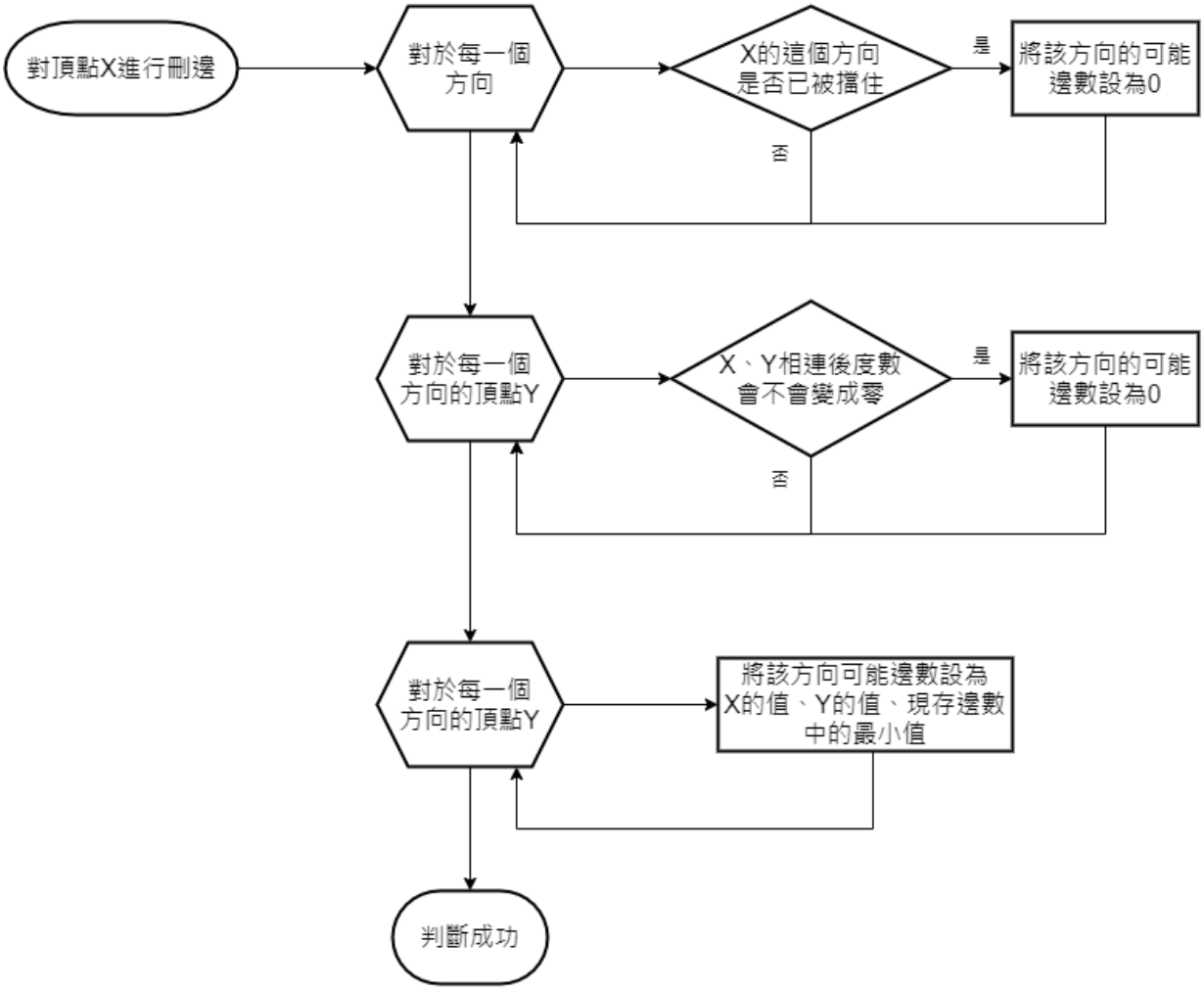


圖 22：鴿籠策略圖的操作流程圖

(3). 鴿籠原理函式

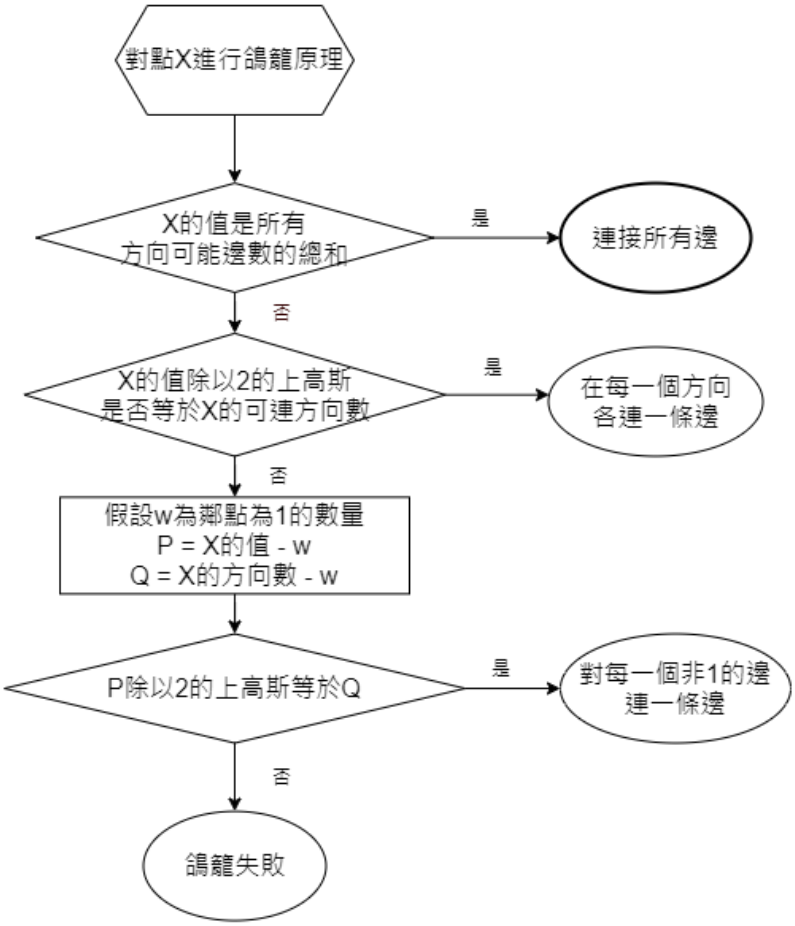


圖 23：鴿籠策略鴿籠函式流程圖

(4). 鴿籠原理迴圈

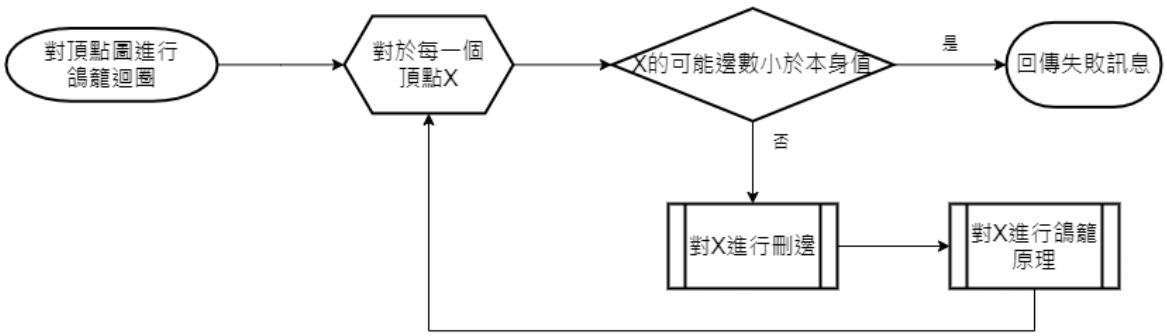


圖 24：鴿籠策略鴿籠迴圈流程圖

6. 鴿籠+窮舉總程式

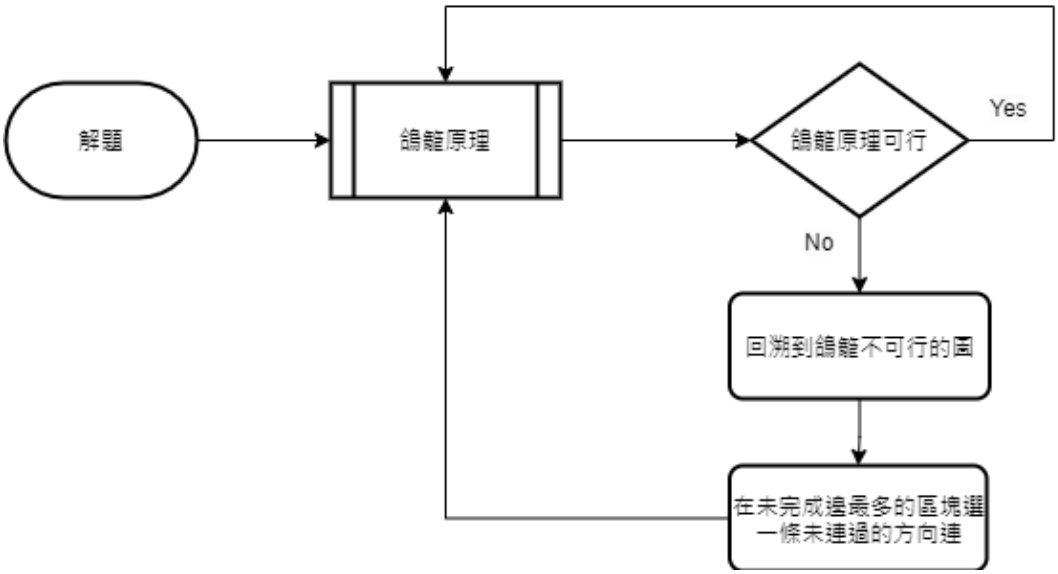


圖 25：條件解題總函式流程圖

五、分析數據

測試資料來源來自於 Thibaut VIDAL, Ph.D.線上網頁之 Benchmark Instances & Detailed Results 文章，題目難度係數與「連通性問題」及「雙邊數量」呈正相關。項目(二)和(三)皆為使用(二)「條件解題」+「窮舉策略」的程式做測試。

(一)執行時間比較

1. 不同演算法解決不同難度的時間關係圖

純窮舉條件+ 窮舉

純窮舉的做法在難度達一定程度時，執行時間超過 30 分鐘仍尚未出現解果，我們將其認定為無法解決。

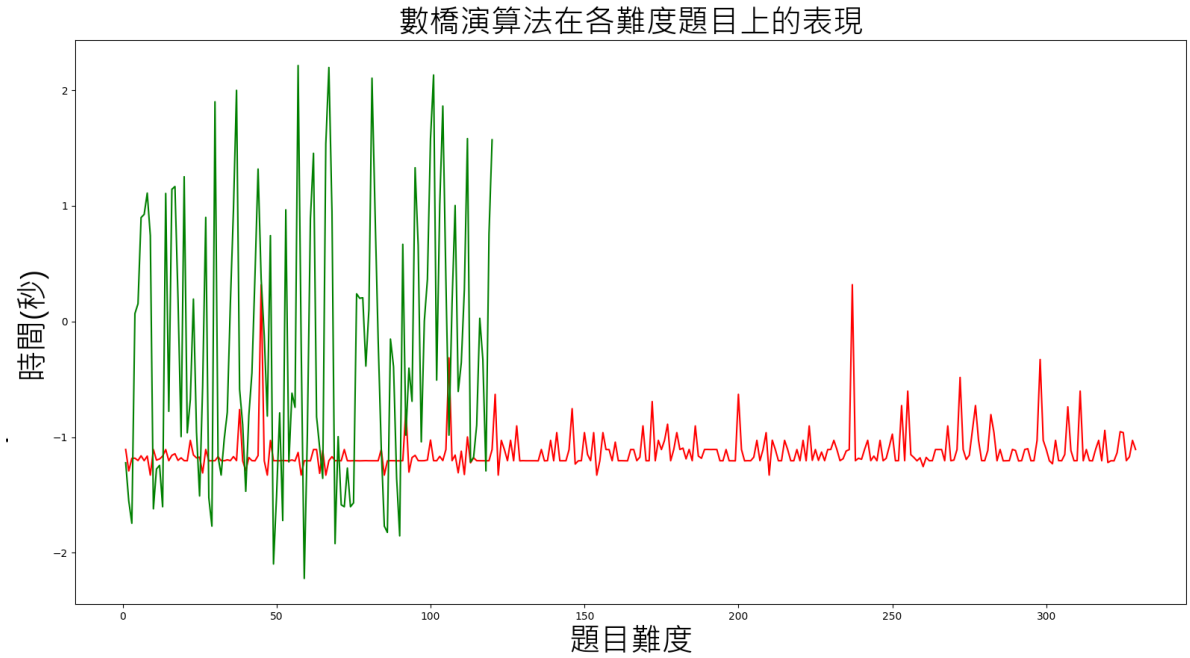


圖 26：數橋演算法在各題目難度上的表現

(二)條件解題與窮舉方法所使用的比例

取最小頂點值　取最小頂點值　取最大度數值

取最小度數值　取最大集合

取最小集合

1. 所有方法在各難度題目上的窮舉邊比例

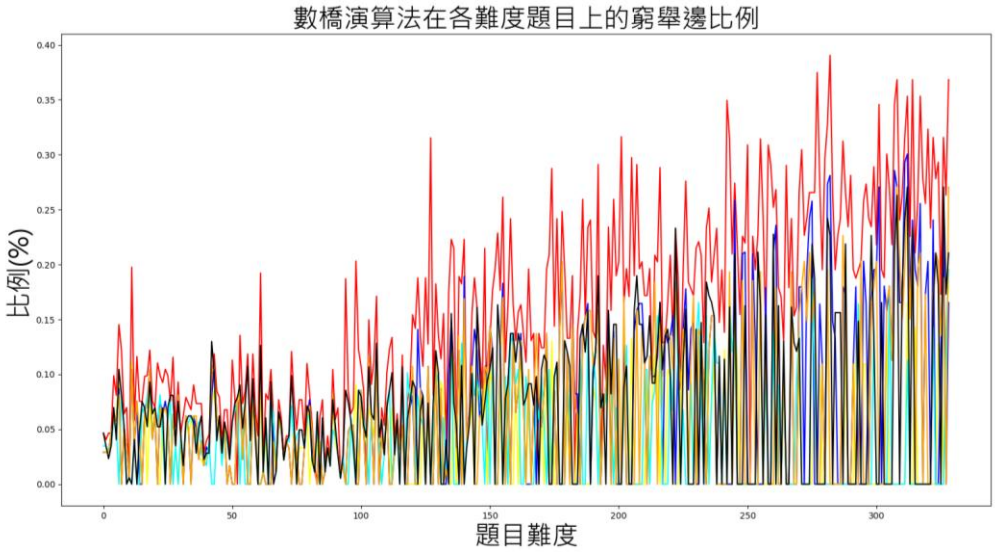


圖 27：所有方法在各難度題目上的窮舉邊比例

2. 窮舉比例較小方法(取度數最小的點(群)、取值最小的點(群)、取點集大小最小的點(群))在各難度題目上的窮舉邊比例

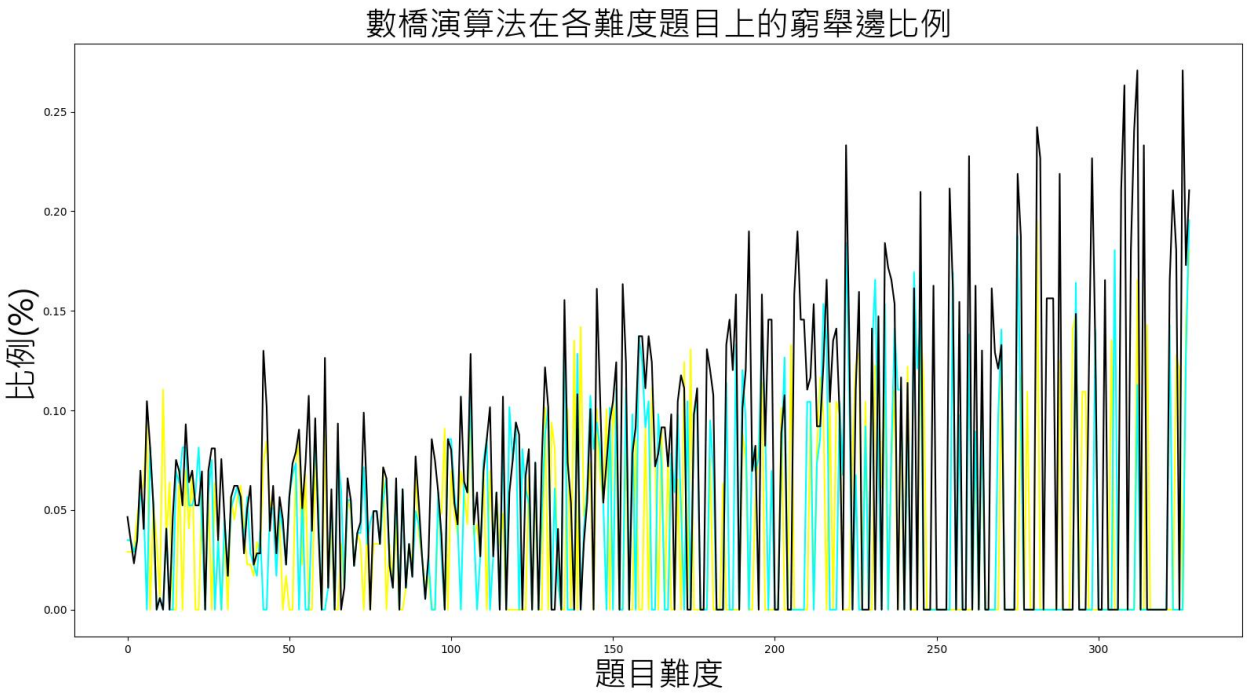


圖 28：窮舉比例較小方法在各難度題目上的窮舉邊比例

(三)窮舉策略討論

1. 不同窮舉策略在不同難度下對時間的關係圖

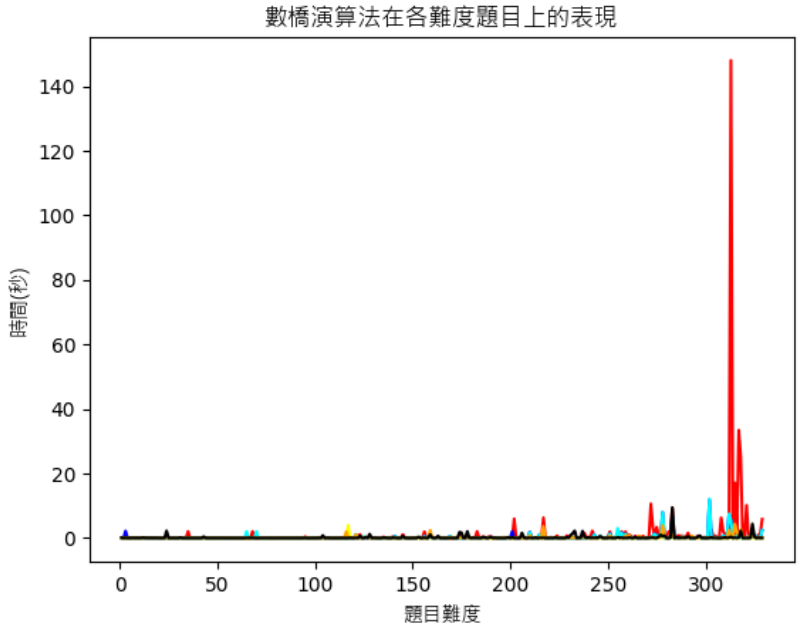


圖 29：不同窮舉策略在不同難度下對時間的關係圖

2. 不同窮舉策略在不同難度下對時間的關係圖(對時間取$\log_{10}$)

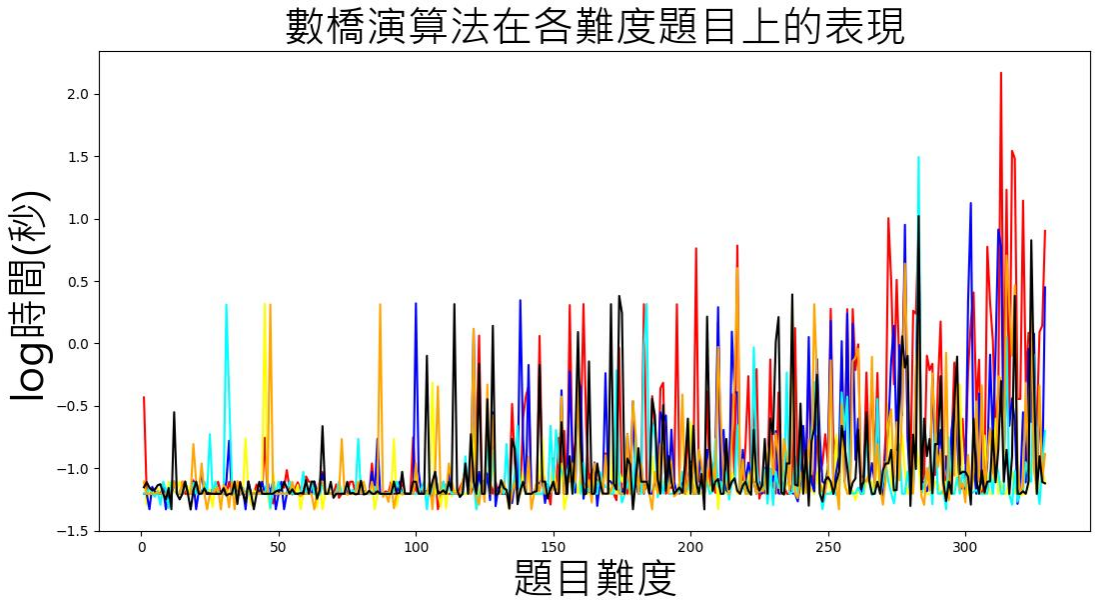


圖 30：不同窮舉策略在不同難度下對時間的關係圖(對時間取$\log_{10}$)

(1)數值超過 1(時間 10 秒)

取最大頂點值(紅)	取最小頂點值(黃)	取最大度數值(深藍)	取最小度數值(淺藍)	取最大集合(橘)	取最小集合(黑)
6 筆	0 筆	2 筆	1 筆	0 筆	1 筆

表 1：數值超過 1 的測資筆數

(2)平均執行時間

	取最大頂點值(紅)	取最小頂點值(黃)	取最大度數值(深藍)	取最小度數值(淺藍)	取最大集合(橘)	取最小集合(黑)
平均	1.148 秒	0.091 秒	0.317 秒	0.205 秒	0.211 秒	0.222 秒
標準差	8.604 秒	0.162 秒	1.152 秒	1.718 秒	0.529 秒	0.755 秒

表 2：平均執行時間

伍、討論

一、如何再增加解題效率？

本研究是利用程式的演算法來解題，除了本篇提到的數種條件下的解題策略，可以再發現更多可以直接解題的條件，以加快解題效率。

另外想再增加解題效率，可以訓練人工智慧，使其能夠在統計大量題目下找出快速解題的方法，並且有高的正確率。若將來有機會摸索到人工智慧，也相當期待人工智慧可以分析出不同的解題策略。

二、如何生成一個困難的問題？

在研究結果中可以發現，程式使用到窮舉的比例與程式的實行時間有很大的關係。因此我們要生成一個需要窮舉的部分較多的題目，讓做題者或程式多花時間去嘗試錯誤。而生成這類題目並非簡單的作法可以達成，有待研究。

三、NP – complete 問題的解法

我們知道利用動態規劃(dynamic programming)的方式可以解決某些部分的 NP – complete 問題，但是動態規劃的解法通常會產生某些偽多項式時間複雜度的解法，因而造成並無法完全解決大部分問題。而數橋問題已被證明為 NP – complete 問題，因此我們就思考某些部份的 NP – complete 問題是否可用「條件」與「窮舉」兩步驟循環解決來加快程式解題的速度。

陸、結論

一、分析與比較不同演算法

(一) 點群與點群之間的相對關係，可以由以下幾點解決：

1. $\mathrm{N(V)} = \mathrm{Way(V)}$
2. $\mathrm{N(V)} > (\mathrm{Adj(V)} - 1) \times 2$
3. 若V_0中的可連的點∃i 使得 $\mathrm{N}(V_i) = 1$，則可將V_0視為V_0'其中$\mathrm{N}(V_0') = N(V_0) - 1,$

$$\mathrm{Adj}(V_0') = \mathrm{Adj}(V_0) - 1$$

4. 連通性問題

(二) 全連通的規則可以做為特判以加快程式的運行速度。

(三) 由純窮舉的流程來解題，對於 100 個點的簡單題型需要平均 10.641 秒(標準差為 29.262 秒)來達成。

(四) 由條件解題 + 窮舉的流程來解題，對於 100 個點需要的執行時間分別為為：

1. 取最大頂點值： 平均 1.148 秒，標準差為 8.604 秒
2. 取最小頂點值： 平均 0.091 秒，標準差為 0.162 秒
3. 取最大度數值： 平均 0.317 秒，標準差為 1.152 秒
4. 取最小度數值： 平均 0.205 秒，標準差為 1.718 秒
5. 取最大集合： 平均 0.211 秒，標準差為 0.529 秒
6. 取最小集合： 平均 0.222 秒，標準差為 0.755 秒

二、窮舉的優先策略

根據測試結果能得到以下結論：

(一) 取最大頂點值是效率最差的窮舉策略

(二) 取最小頂點值是效率最佳的窮舉策略

(三) 我們可以將程式設計為在窮舉時，隨機在這三種策略選擇，並且在其中一種策略執行時間超過一定時間時，替換至剩餘的兩個策略，以增加執行效率。

三、未來展望

(一) 將來若想使執行速度更快，可以使用人工智慧來解題，也可更高效的找出相對優秀的策略或解答。

(二) 在數橋遊戲的方面將來也可將圖形擴充成三維空間，將島嶼間可連的邊改為三條等，都很有發展性。

(三) 添加無環的條件是否可以讓效率變高，抑或是使題目脫離 NP 問題的範疇，因為樹是一個含有多樣性質的結構，故很有潛力。

(四) 在數橋是否存在解的充分必要條件，可有機會再去挖掘。

(五) NP 問題的窮舉策略：對於每一個 NP 問題是否都存在一種在剪枝、合理推定等過程做加速的方法能使 NP 問題大幅提高其窮舉效率，有待之後研究。

有了這個研究的經驗，在之後對於其他數學謎題的研究也能夠有更加明確的方向。

柒、參考文獻資料

1. 輕鬆談演算法的複雜度分界：什麼是 P, NP, NP-Complete, NP-Hard 問題(YC Note ,2017)
https://ycc.idv.tw/algorithm-complexity-theory.html

2. The benchmark instances used in "Coelho, L., Laporte, G., Lindbeck, A. & Vidal, T. (2018).

3. Benchmark Instances and Branch-and-Cut Algorithm for the Hashiwokakero Puzzle(Leandro C. Coelho, Gilbert Laporte, Arinei Lindbeck, Thibaut Vidal, 2019)

4. Andersson, D. 2009. Hashiwokakero is NP-complete. Information Processing Letters 109(19) 1145–1146.

5. Hashiwokakero (T. Morsink, 2009)

6. Malik, R. F., R. Efendi, A. P. Eriska. 2012. Solving hashiwokakero puzzle game with hashi solving techniques and depth first search. Bulletin of Electrical Engineering and Informatics 1(1) 61–68.

6. Hashi Solver(Shi-Jim Yen, Shih-Yuan Chiu, Cheng-Wei Chou, Tsan-Cheng Su, 2011)

【評語】052511

1. 本作品以一個已被證明為 NP-Complete 的數橋謎題遊戲為基礎，提出已程式窮舉解決該問題的各種策略，並比較這幾種策略的相對效率。

2. 本作品題目具創新性，為資訊科學基礎理論的研究探討，作品完整度高，值得肯定與鼓勵。

作品海報

箇中「橋」楚

—研究不同演算法對數橋遊戲的差異性

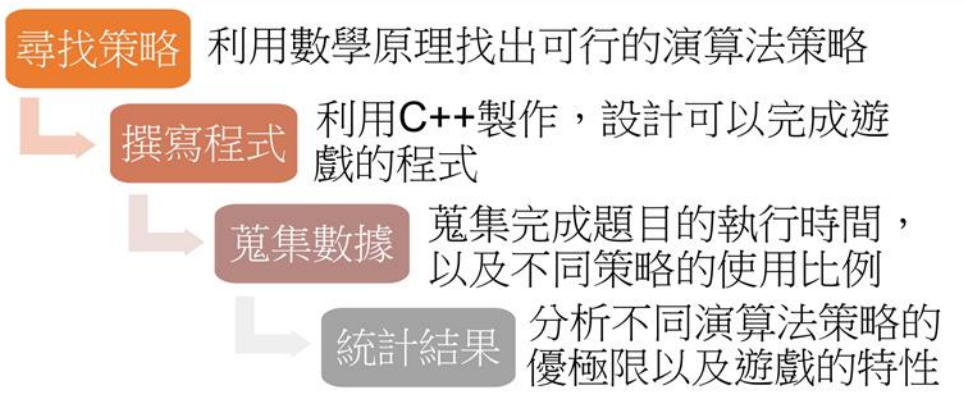
壹、摘要與研究動機

在學習演算法的過程中，我們習學到利用程式的演算法可以去解決很多有趣的問題，其中也包含一些數學謎題，例如：數獨、八皇后問題等。數學謎題有很多，我們好奇的想找出可以解出各式數學謎題的不同的程式，並且對那些程式進行分析了解不同演算法對於解題的影響，也想藉由分析程式中的項目來更加了解這些數學遊戲的特性。在琳瑯滿目的數學遊戲中，我們選擇了數橋(Hashiwokakero)這個遊戲，因為他是由數個點和邊組成的遊戲，玩法相當獨特，也使我們聯想到曾經學過有關圖論的演算法。本研究提出有關數橋謎題遊戲的程式實作辦法，研製出了數種鴿籠原理與窮舉策略的實現，並且比較了這幾種解決辦法的相對效率，再者也提出了數個NP問題的解決策略。本研究發現在進行窮舉的時候以最小度數有最好的期望效率(表示該程式能有相對多，相對快的執行結果)。

貳、研究目的與流程

通過尋找策略並撰寫不同程式，可以比較並得到以下幾項問題的解答

(一) 找出能夠解出數橋的程式
(二) 比較不同策略與演算法在解決數橋遊戲上的執行效率
(三) 分析程式的執行時間及各部分使用比例
(四) 比較不同的窮舉策略



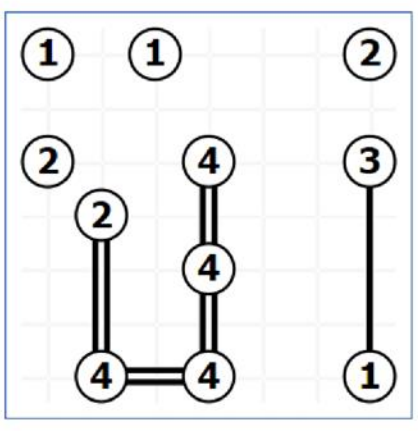
參、文獻回顧

- 數橋規則介紹

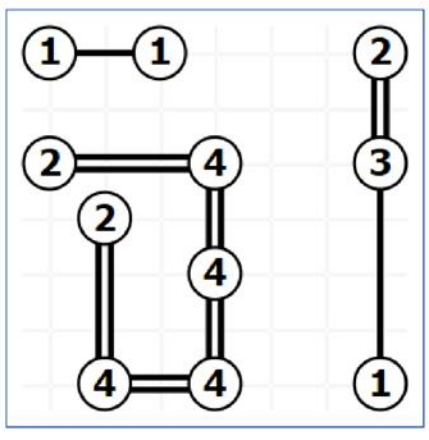
數橋的題目通常設在矩形中，矩形中的圓圈會有數字，我們稱作「島嶼，遊戲的目標是要將「橋梁」連在正確的地方，其中有以下條件：

1. 橋梁必須要是直線，且連接兩座島嶼
2. 橋梁中間不可跨越任何島嶼
3. 橋梁不可跨越另一座橋
4. 兩座島嶼之間最多只可以連接兩座橋
5. 每座島嶼所連接的橋數必須與島嶼上的數字相等
6. 所有島嶼必須要是一個群體

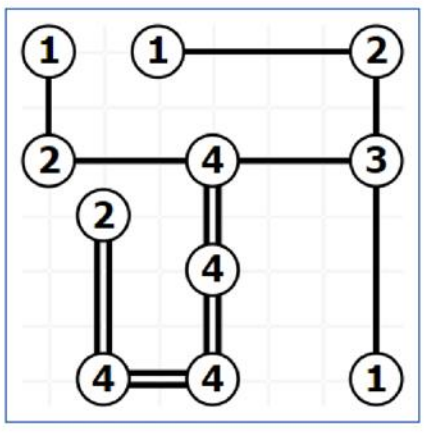
舉例來說：遇到圖一這樣的情況時，若我們將島嶼1往右連，則會出現如圖二 的兩個群體，不符合規則6，因此如圖三將島嶼1往上連是唯一正解。滿足以上6個條件時即完成遊戲。



圖一：需要用到連通性規則的情況



圖二：錯誤解



圖三：正解

- P/NP問題

P與NP問題是在是計算複雜度理論中，決定性問題的等級之一。若有一群演算法用DTM(deterministic Turing machine)來做計算所需時間是多項式時間(polynomial time)，那這類演算法或問題被稱為P問題；若有一群演算法用NTM(non-deterministic Turing machine)來做計算所需時間是多項式時間，那這類問題被稱為NP問題。
數學理論指出任何一個NP裡面的問題都可以在多項式時間內，使用DTM化約成「一個布林方程式是否存在解」的問題，又將這個問題稱為NP-complete 問題。
一般的電腦屬於DTM，利用DTM來解NP-complete問題時需要花上指數時間以上的複雜度，與P問題相差極大。雖然尚未有人證出P與NP-complete 問題是否等價，不過目前普遍認為P≠NP-complete。因此若一個問題是NP-complete問題，則常被認定為難題。很多的加密方法也是利用NP-complete問題難解的特性而產生。
而數橋問題已被證明出為NP-complete問題。

- 並查集(Disjoint-set)

並查集是一種資料結構，主要作用為查詢元素存在於哪個子集，每個子集通常會將一個元素作為代表編號，其中並查集基本有以下之操作：

1. 查詢：查詢某個元素屬於哪個集合，通常是返回集合內的一個「代表元素」。這個操作是為了判斷兩個元素是否在同一個集合之中。
2. 合併：將兩個集合合併為一個。
3. 添加：添加一個新集合，其中有一個新元素。添加操作不如查詢和合併操作重要，常常被忽略。

- 持久化並查集

持久化並查集是由持久化線段樹延伸的資料結構，比起普通的並查集多了回溯到歷史版本的功能，這個資料結構的複雜度如下：

功能	時間複雜度	空間複雜度
建立	$O(N \log N)$	$O(N \log N)$
查詢	$O(\log N)$	$O(1)$
合併	$O(\log N)$	$O(\log N)$
回溯	$O(1)$	$O(1)$

- 可回溯式連線紀錄平面

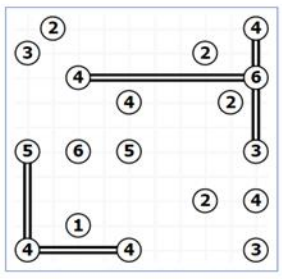
這個資料結構是一個可以記錄在平面上某處連了某數量的橋的資料結構，附帶了紀錄每一次連線的功能，因此可以回溯的某特定歷史版本，其複雜度如下：

功能	時間複雜度	空間複雜度
建立	$O(W^2)$	$O(W^2)$
增加	$O(W)$	$O(1)$
回溯(k個版本之前)	$O(kW)$	$O(1)$

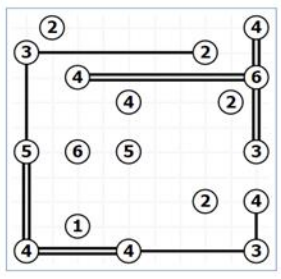
肆、研究結果

一、條件策略

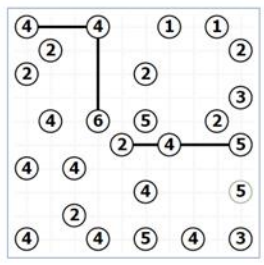
(一)島上數字 = 剩餘可連的橋



(二)島上的數字 > (相鄰島嶼數-1)×2

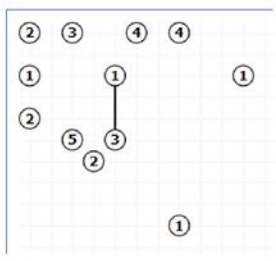


(三)若相鄰島嶼出現數字1的島，則可將原島嶼看作「相鄰島嶼數-1，島嶼數字-1」代入策略(二)做判斷

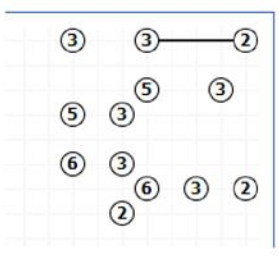


(四)連通性問題　運用規則6.「所有島嶼必須要是一個群體」得出

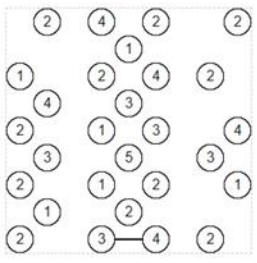
1鄰1－X



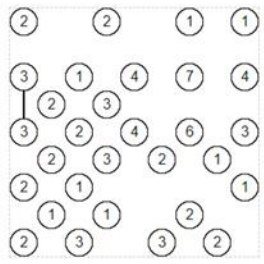
2鄰2－X



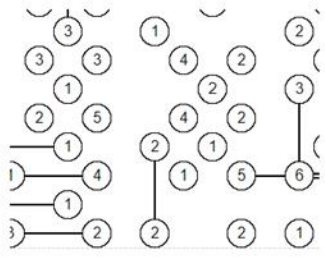
4鄰2－2－X



3鄰2－1－X



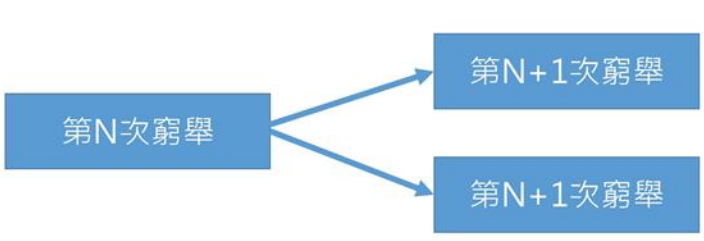
2鄰1－1－X



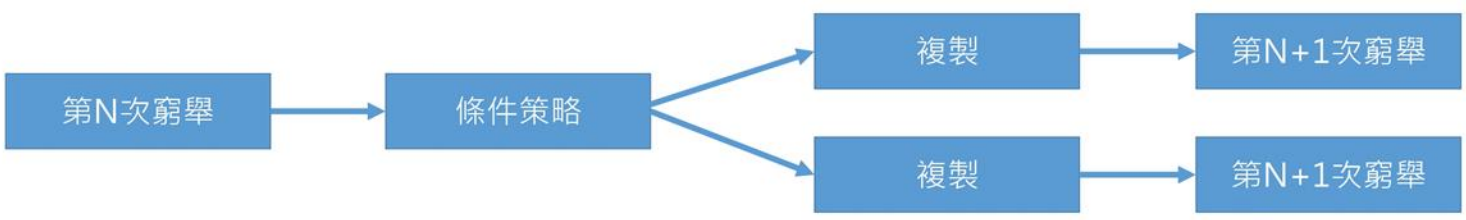
二、解決方法：

本研究運用三種解題方法解決數橋問題

1. 不使用任何條件策略直接進行DFS



2. 使用條件策略+六種窮舉策略，其中的資料結構不具有回溯性質，因此採取copy on write策略，在每一個DFS分支複製一份當下情況

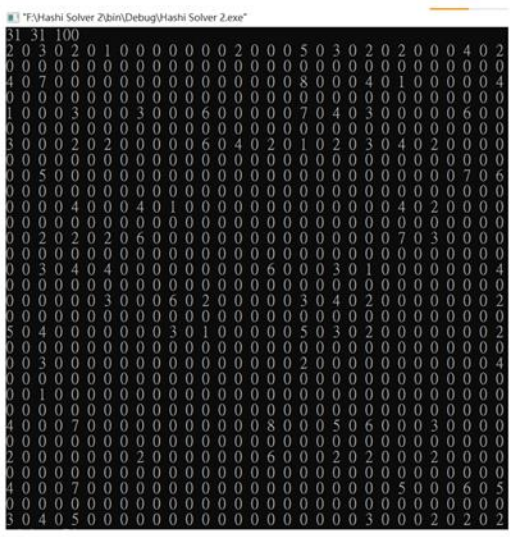


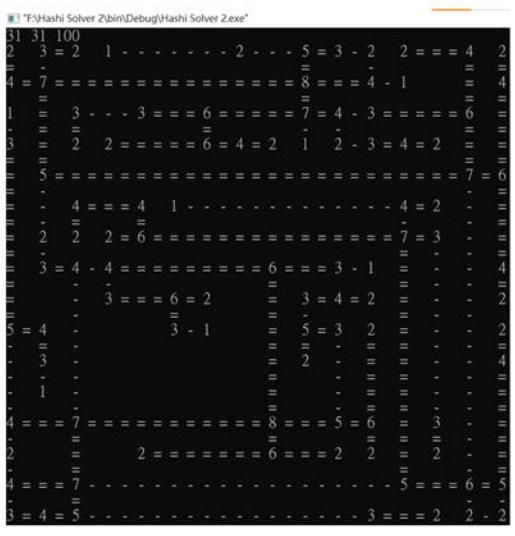
3. 使用條件策略+六種窮舉策略，其中的資料結構具回溯性質，可以應對DFS的前後修改

六種窮舉策略為：取最大點值點、取最小點值點、取最大點群任意點、取最小點群任意點、取最大度數點群任意點、取最小度數點群任意點

三、實作介面：

我們將數個測資輸入到程式裡，程式經過執行之後會產生解答及執行時間等數據，測資皆為有解的題目



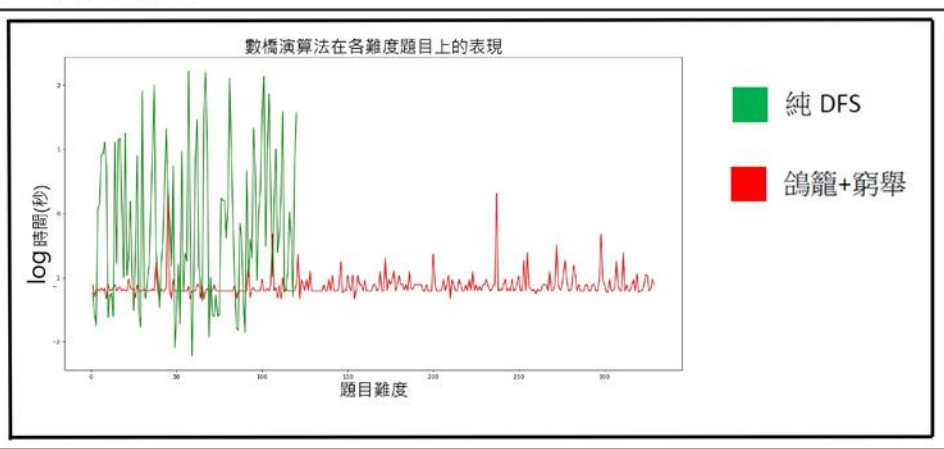


解數橋的程式　　生成圖表的程式　　示意圖

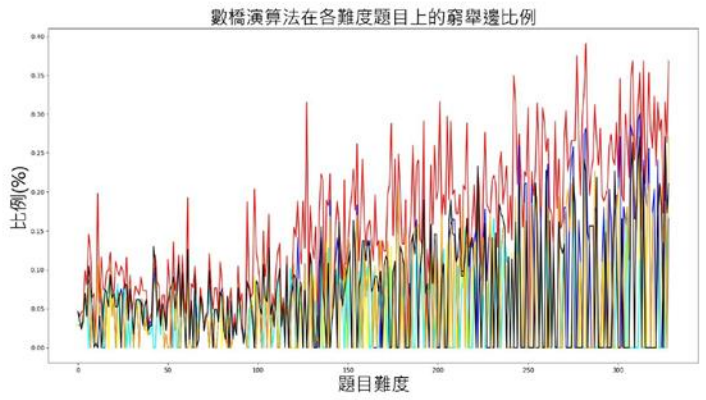
肆、研究結果

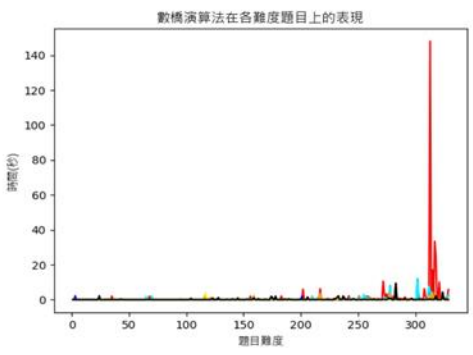
數據分析

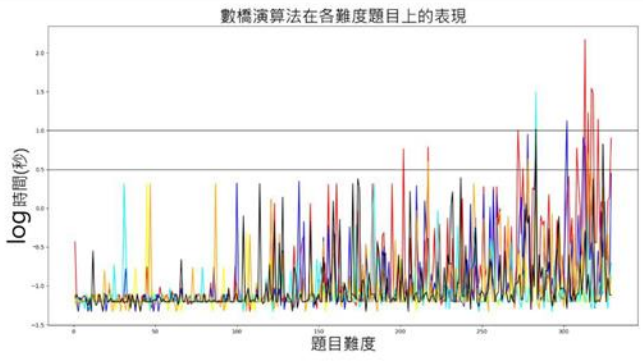
題目難度與「連通性問題」及「雙邊數量」呈正相關。項目(二)和(三)皆為使用「鴿籠原理」+「窮舉策略」的程式做測試。

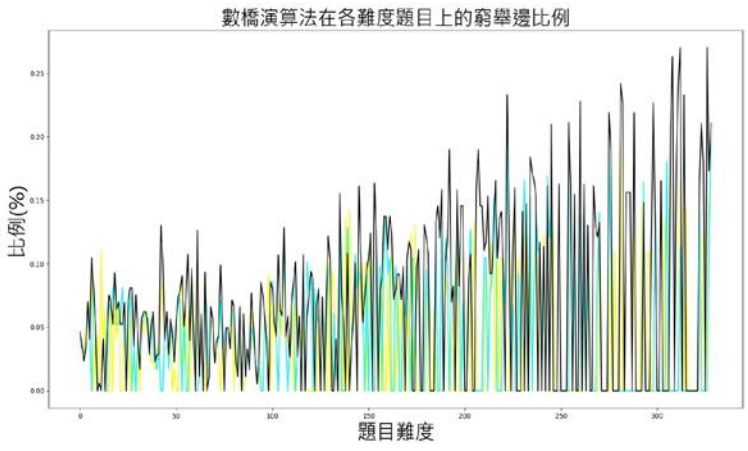


取度數最大的點(群)　取度數最小的點(群)　取值最大的點(群)
取值最小的點(群)　取點集大小最大的點(群)
取點集大小最小的點(群)









數值超過0.5(時間約3.16秒)

取度數最大的點(群)(紅)	取度數最小的點(群)(黃)	取值最大的點(群)(深藍)	取值最小的點(群)(淺藍)	取點集大小最大的點(群)(橘)	取點集大小最小的點(群)(黑)
11筆	1筆	5筆	1筆	4筆	2筆

數值超過1(時間10秒)

取度數最大的點(群)(紅)	取度數最小的點(群)(黃)	取值最大的點(群)(深藍)	取值最小的點(群)(淺藍)	取點集大小最大的點(群)(橘)	取點集大小最小的點(群)(黑)
6筆	0筆	2筆	1筆	0筆	1筆

伍、討論

如何再增加解題效率？

1.找出更多不同可以使用的鴿籠原理的策略以減少窮舉的比例
2.以大量題目訓練人工智慧以找出快速解題的方法並同時享有高正確率

如何生成一個困難的問題？

在研究結果中可以發現，程式使用到窮舉的比例與程式的實行時間有很大的關係。因此我們要生成一個需要窮舉的部分較多的題目，讓做題者或程式多花時間去嘗試錯誤。而生成這類題目並非簡單的作法可以達成，有待研究。

NP – complete問題的解法

我們知道利用動態規劃(dynamic programming)的方式可以解決某些部分的NP – complete問題，但是動態規劃的解法通常會產生某些偽多項式時間複雜度的解法，因而造成並無法完全解決大部分問題。而數橋問題已被證明為NP – complete問題，因此我們就思考某些部份的NP – complete問題是否可用「鴿籠」與「窮舉」兩步驟循環解決來加快程式解題的速度。

陸、結論

分析與比較不同演算法

點群與點群之間的相對關係，可以由以下幾點解決：

1.島上的數字 = 剩餘可連的橋　　2. 島上的數字 > (相鄰島嶼數 − 1) × 2

3.若相鄰島出現數字1的島，則可將原島嶼看作「相鄰島數 − 1，島嶼數字 − 1」代入策略(二)做計算

4.連通性問題 -----全連通的規則可以做為特判以加快程式的運行速度。

	純DFS	最大頂點值	最小頂點值	最大度數值	最小度數值	最大集合	最小集合
平均秒數(秒)	10.641	1.148	0.091	0.317	0.205	0.211	0.222
標準差(秒)	29.262	8.604	0.162	1.152	1.718	0.529	0.755

窮舉的優先策略

根據測試結果能得到以下結論：

1.取度數最大的點(群)是效率最差的策略，窮舉與用比例「取值最小的點(群)」是相對效率好的策略

2.我們可以將程式設計為在窮舉時，隨機較好策略選擇，並且在其中一種策略執行時間超過一定時間時，替換至剩餘的其他策略，以增加執行效率。

柒、未來展望

1.將來若想使執行速度更快，可以使用人工智慧來解題，也可更高效的找出相對優秀的策略或解答。

2.在數橋遊戲的方面將來也可將圖形擴充成三維空間，將島嶼間可連的邊改為三條等，都很有發展性。

3.添加無環的條件是否可以讓效率變高，抑或是使題目脫離NP問題的範疇，因為樹是一個含有多樣性質的結構，故很有潛力。

4.在數橋是否存在解的充分必要條件，可有機會再去挖掘。

5.NP問題的窮舉策略：對於每一個NP問題是否都存在一種在剪枝、合理推定等過程做加速的方法能使NP問題大幅降低其窮舉效率，有待之後研究。

有了這個研究的經驗，在之後對於其他數學謎題的研究也能夠有更加明確的方向。

捌、參考文獻

1. 輕鬆談演算法的複雜度分界：什麼是P, NP, NP-Complete, NP-Hard問題(YC Note ,2017) https://ycc.idv.tw/algorithm-complexity-theory.html
2. The benchmark instances used in "Coelho, L., Laporte, G., Lindbeck, A. & Vidal, T. (2018).
3. Benchmark Instances and Branch-and-Cut Algorithm for the Hashiwokakero Puzzle(Leandro C. Coelho, Gilbert Laporte, Arinei Lindbeck, Thibaut Vidal, 2019)
4. Andersson, D. 2009. Hashiwokakero is NP-complete. Information Processing Letters 109(19) 1145–1146.
5. Hashiwokakero (T. Morsink, 2009)
6. Malik, R. F., R. Efendi, A. P. Eriska. 2012. Solving hashiwokakero puzzle game with hashi solving techniques and depth first search. Bulletin of Electrical Engineering and Informatics 1(1) 61–68.
6. Hashi Solver(Shi-Jim Yen, Shih-Yuan Chiu, Cheng-Wei Chou, Tsan-Cheng Su, 2011)