

# Final Report

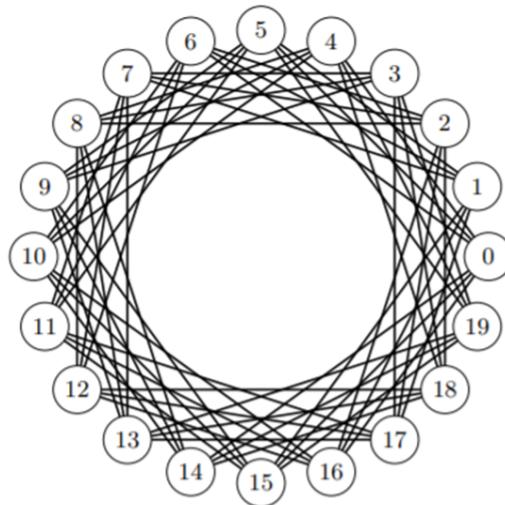
## 第二組

B08901002 林宸漢 B08901049 張原嘉 B08901209 林鈺翔

### 問題定義 [1]

Graph Collision Problem: 給定一個圖，其中有 N 個點和若干條邊，若我們在若干個點上標記，我們想要找出是否有兩個標記的點存在一條邊連接

在本次期末 project 中，我們討論特定的環形圖，共有 N 個點，每個點跟距離自己 a 至 b 的點有連線，並且假設每個點有被標記的機率為  $p(0 < p < 1)$ ，我們用量子演算法在  $O(\sqrt{N})$  的時間內判斷是否有 collision。 [2]



(圖一: CI 圖 (20,4,6))

### 輸入輸出

#### 輸入

布林函數  $f$ , 有 N 個 element, 每個 element 都是 0 或 1,  $f[i] = 0$  代表在第  $i$  個點沒有標記,  $f[i] = 1$  代表在第  $i$  個點上標記

#### 輸出

0 或是 1, 0 代表圖中不存在任何兩個標記的點有邊連接 (沒有 collision), 1 代表圖中存在兩個標記的點有邊連接 (有 collision)

### 程式演算法 (括號內為使用的函式名稱)

- 先把所有頂點切成好幾個區塊，每個區塊有  $b - a$  個頂點，共有  $\frac{n}{b-a}$  個區塊
- 在每個區塊中找出有被標記的點中最小的 index, 如果找不到 return -1, 如果找得到再找出有被標記的點中最大的 index  
(`find_min_one_idx_update_quantum`、`find_min_one_idx_quantum_sub`)

3. 在每個區塊中找出最小和最大的被標記 index 後，將最小的 index 加上 nearest\_connect, 將最大的 index 加上 farthest\_connect, 在加完後的 index 之間找尋有沒有任何的 1, 如果有 1 設定這個區塊的值為 1, 如果沒有任何的 1 將這個區塊的值設為 0 ( is\_any\_one\_quantum )
4. 對於每一個區塊, 如果全部都沒有 1, 即沒有 collision, 回傳 0, 如果有任何一個區塊有 1, 有 collision, 回傳 1(is\_any\_one\_quantum)

## 量子函式簡介

1. TruthTableOracle [3]: 在這個函式中, input string 的長度需為 2 的冪次方, output 有高機率會是在 input string 中數值為 1 的 index
2. is\_any\_one\_quantum: 在這個函式中, 紿定一個 input string, 稱 f, 我們找尋在這之中有沒有任何一個 1。若有, 則 return 1, 否則 return 0。而我們的方法是由 TruthTableOracle 使用 Grover Search 去 measure 出 1 所在的 index with high probability。最後再確認 f 的那個位置是否真的等於 1。以上的確認是必要的, 因為 Grover Search 一定會給我們一個 index。假若 string 是全 0, 則我們仍會隨機的得到一個 index。所以我們拿到一個 index 後, 我們不從 f 做再確認的話無法知道 f 是全 0 然後隨機給了我們這個 index, 抑或 f 內有 1 然後有高機率會給我們這個index。
3. find\_min\_one\_idx\_quantum\_sub: 在這個函式中, 我們用 is\_any\_one\_quantum 函式找尋有沒有任何一個 1, 如果找不到 1, 即在 string 中沒有 1, 回傳 -1, 由於 TruthTableOracle 使用 Grover Search, 因此時間複雜度為  $O(\sqrt{b-a})$
4. find\_min\_one\_idx\_update\_quantum: 當上個函式有找到 1, 我們進入這個函式, 這個函式會在原本的第一個 index 到上個函式找到的 1 之前的 index 中找有沒有任何一個 1(用 is\_any\_one\_quantum), 如果找不到 1, min index 即為傳入的 index, 如果有找到 1, 我們遞迴執行這個函式, 範圍從第一個 index 到上個函式找到的 1, 直到找不到任何一個 1 為止, 由於 TruthTableOracle 使用 Grover Search, 期望值而言我們每次範圍都會縮小一半, 因此整體複雜度為  

$$O(\sqrt{b-a}) + O(\sqrt{\frac{b-a}{2}}) + O(\sqrt{\frac{b-a}{4}}) + O(\sqrt{\frac{b-a}{8}}) + O(\sqrt{\frac{b-a}{16}}) \dots$$

$$= O(\sqrt{b-a})(1 + \sqrt{\frac{1}{2}} + (\sqrt{\frac{1}{2}})^2 + (\sqrt{\frac{1}{2}})^3 \dots) = O(\sqrt{b-a}), \text{複雜度依舊是 } O(\sqrt{b-a})$$
5. find\_min\_one\_idx\_quantum: 為提高正確率, 我們將 call find\_min\_one\_idx\_quantum\_sub 許多次( itr 次)。

6. `find_max_one_idx_quantum_sub /`  
`find_max_one_idx_update_quantum/find_max_one_idx_quantum`：與 3, 4, 5  
中 `min` 的版本相對

## 實驗與討論

我們發現如果只有執行一次量子演算法，有機率會出錯，為了讓出錯機率降低

### 實驗 1：

我們在某一個 chunk 中重複執行多次量子演算法，看結果是較多有 collision 還是較多沒有 collision 當成是這個 chunk 是否有 collision，並統計量子演算法判斷在這個 chunk 中是否有 collision 的正確率

### 實驗2：

我們得到每一個 chunk 有沒有 collision 後，得到一個 string，並重複執行多次 `is_any_one_quantum` 演算法，看結果是較多有 collision 還是較多沒有 collision，並統計量子演算法判斷整張圖有沒有 collision 的正確率

我們取  $N = 12$ ,  $\text{nearest\_connect} = 2$ ,  $\text{farthest\_connect} = 4$ ,  $\text{prob} = 0.178$  (手動試出來的，為了要讓在某一個 chunk collision 的機率趨近 0.5，實際調出來後約為 0.517)，我們操縱量子演算法 iteration 的次數為 1, 3, 5, 7, 9。不能是偶數的原因是因為可能兩派恰好結果一半一半，我們將無法判斷誰對誰錯。

1. 觀察在一個 chunk collision 出錯的機率
2. 觀察決定 chunk 有沒有 collision 後，整張圖 collision 出錯的機率

## 實驗結果

### 實驗 1：觀察在一個 chunk collision 出錯的機率

實驗情境：因為在  $\text{CI}(12, 2, 4)$  的 graph 中，我們的演算法一共會切出 6 個 chunk 去計算。而我們預計做約 1000 個實驗樣本，所以我們跑了演算法共 170 次，產生 170 個不同的 f 作為 input (預期每個 f 對於每個 chunk 發生 collision 的機率為 0.5)，每次會幫我們計算出 6 個 chunk 的結果，所以我們一共有  $6 * 170 = 1020$  個實驗樣本。而每個樣本我們用 classical 的方式作為我們的參考答案。最後依據以上實驗樣本去計算出正確率。

`iteration = 1`  
正確率 : 0.9774509803921568

`iteration = 3`  
正確率 : 0.995098039

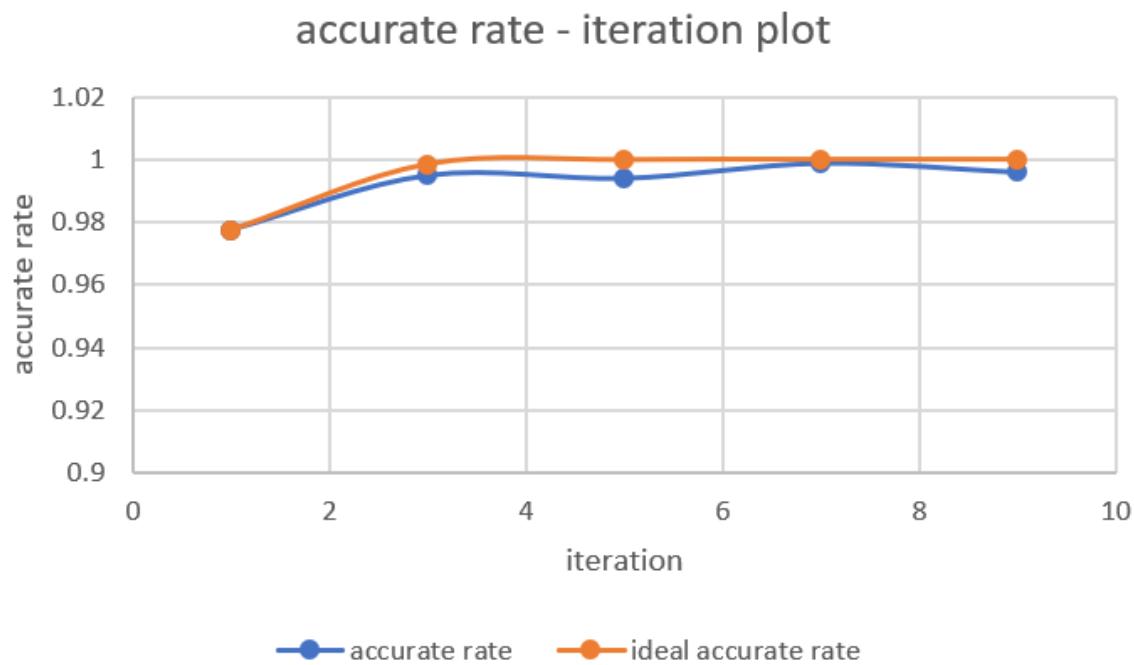
`iteration = 5`  
正確率 : 0.994117647

iteration = 7

正確率 : 0.999019608

iteration = 9

正確率 : 0.996078431



### 觀察

- 由以上實驗結果我們可觀察出實驗結果確實未大於理論估計值 (upper bound)
- 由以上實驗結果我們可觀察出 iteration 的次數對於每個 chunk 的正確率影響不大

### 實驗 2 : 觀察決定 chunk 有沒有 collision 後，整張圖 collision 出錯的機率

實驗情境：由於 runtime 上的考量，我們假設由 quantum approach 得出各個 chunk 有沒有 collision 的結果是正確的。在這個假設下，我們把每個 chunk 發生 collision 的機率設成 0.5，shot 設為 1，重複 measure 的次數為 iteration，以計算整張圖 graph collision 的結果出錯的機率。

iteration = 1

正確率 : 0.89

iteration = 3

正確率 : 0.93

iteration = 5

正確率:0.91

iteration = 7

正確率:0.9

iteration = 9

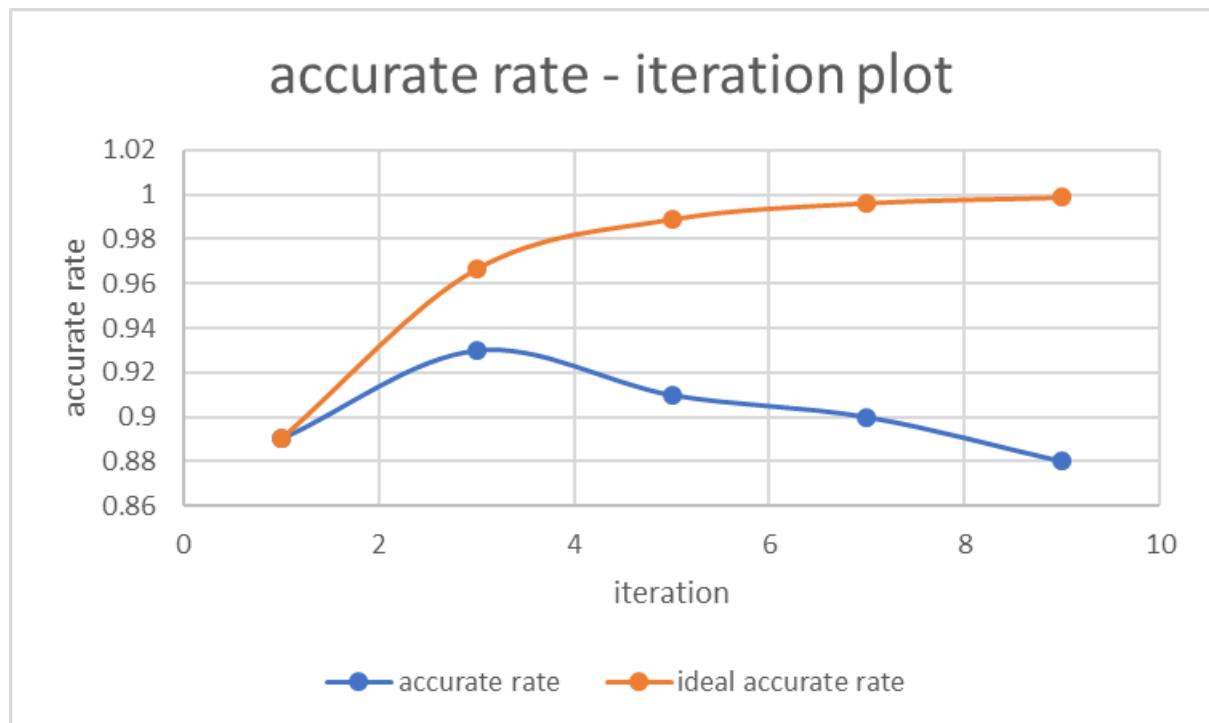
正確率:0.88

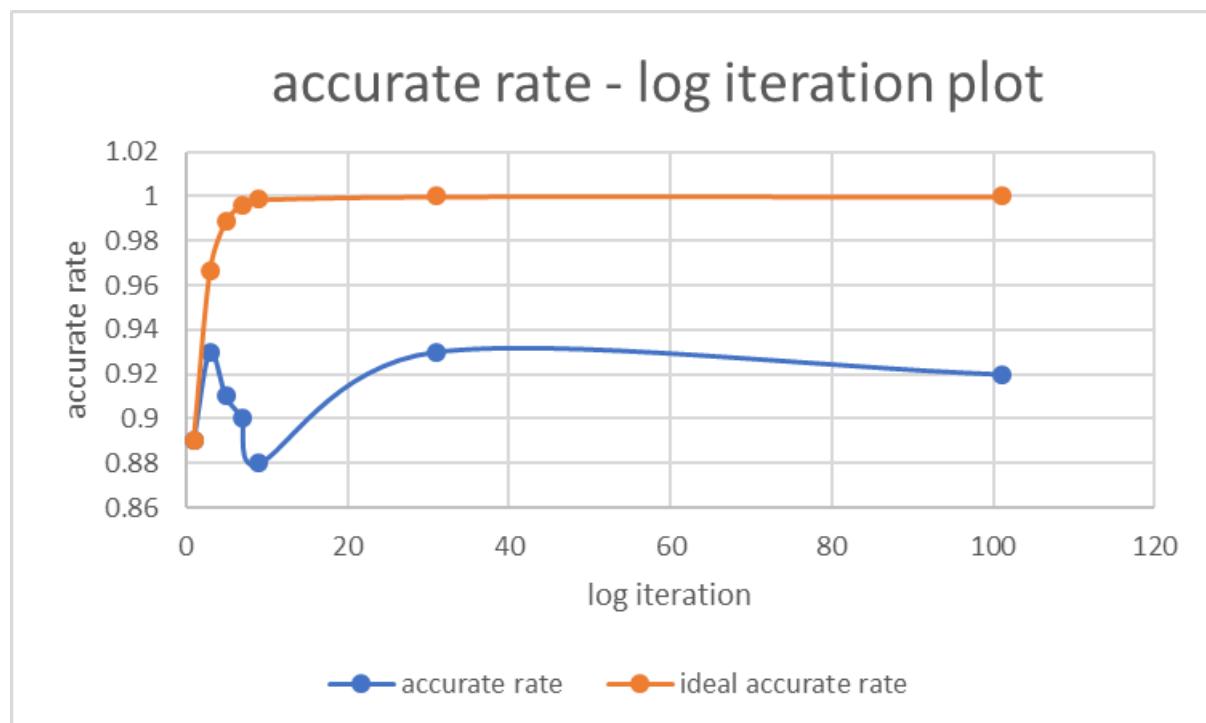
iteration = 31

正確率:0.93

iteration = 101

正確率:0.92





## 討論

### 實驗 1

#### 理論值估計

- 觀察在  $\text{iteration} = 1$  的情況下，一個 chunk collision 出錯的機率，做實驗後得到該機率  $p = 1 - 0.9774509803921568 = 0.02254901961$ 。
- 我們可由  $p$  推算出在  $\text{iteration} = 3, 5, 7, 9$  的情況下，一個 chunk collision 出錯的機率。舉例來說， $\text{iteration} = 3$ ，則出錯機率  $= \Pr(\text{三次都錯}) + \Pr(\text{三次中有兩次錯}) = p^3 + C(3, 1) * p^2 * (1 - p) = 0.998497556$ 。
- 同理，在  $\text{iteration} = 5$  的情況下，出錯機率  $= \Pr(\text{五次都錯}) + \Pr(\text{五次中有四次錯}) + \Pr(\text{五次中有三次錯})$   
 $= p^5 + C(5, 1) * p^4 * (1 - p) + C(5, 2) * p^3 * (1 - p)^2 = 0.999889191$
- 在  $\text{iteration} = 7, 9$  的情況下，依此類推。

同理，實驗 2 的理論估計也用同樣方法

#### 誤差討論

- 我們做的實驗的樣本數不夠多，為看出  $\text{iteration}$  間的正確率，我們必須取夠多的位數。但我們無法隨之增加我們跑的樣本數，runtime 將變好幾個小時（結論 2）。

## 誤差改進

1. 花時間下去跑足夠多次的樣本數。
2. 改進我們的演算法，把 runtime 確實壓到  $O(\sqrt{b - a})$  (結論 2)。

## 實驗 2

我們可以看到雖然實驗 2. 和 1. 是在討論完全相同類型的問題，但整張圖 graph collision 出錯的機率卻明顯高出許多。此外，我們也觀察到 graph collision 出錯的 case 常常是搜尋範圍中只有一個 1 的情況，例如：0010000000000000。我們猜測在搜尋範圍中只有少數個 1 的情況下，決定 graph collision 時 grover search 的範圍 (16) 比決定一個 chunk 有沒有 collision 時更大，因此出錯率較高，也造成了 graph collision 整體的錯誤率較單一 chunk collision 的錯誤率更高。為了驗證我們的猜測，我們進行了一個小實驗，分別計算 is\_any\_one\_quantum 中 TruthTableOracle 的 input (output\_str) 為 0010, 00100000, 0010000000000000，在 shot = 1, 重複 measure 的次數 iteration = 100 次之下，is\_any\_one\_quantum 的正確率。結果顯示，三種不同長度的搜尋範圍得到的正確率依序為 1, 0.81, 0.48。由此可以推論，我們的猜測是正確的，即在搜尋範圍中只有少數個 1 的情況下，決定 graph collision 時 grover search 的搜尋範圍較大，因此該次 graph collision 的出錯率較高，造成整體 graph collision 出錯的機率較高。

以上結果也可以說明為何實驗 1 的錯誤率為什麼會這麼低。因為在實驗 1 中，我們只關心 1 個 chunk，所以我們用 TruthTableOracle 的 Grover Search 的範圍最多僅  $2(b - a) = 4$ 。我們等同是在長度是 8 (結論 2) 的 string 內尋找 1。而由以上知在 iteration 為 100 的情況下，我們即使是在容易出錯的 case 也有 0.81 的正確率。

此外，可以觀察到實驗結果中，錯誤率和 iteration 是沒有明顯關係的。我們為此又進行了另一個小實驗，探討 0010000000000000 作為 TruthTableOracle 的 input (也就是 graph collision 容易出錯的 case)，shot = 1, iteration 分別為 100 與 1000 之下，is\_any\_one\_quantum 的錯誤率，結果分別為 0.48, 0.462。由此可以推論，若使用 TruthTableOracle，iteration 的增加無法提升容易出錯的 case 的正確率，也因此無法有效提升整體 graph collision 的正確率。

## 結論

1. 用 quantum algorithm 理論上可以讓複雜度降到  $O(\sqrt{N})$ ，但是我們還是要去創建所有大小為  $b - a$  的區塊(共  $\frac{n}{b-a}$  個區塊)，在論文 [2] 中說找尋只要花  $O(\sqrt{\frac{n}{b-a}})$  時間，但是我們判斷每一個區塊有沒有 collision 時就已經花了線性時間  $O(\frac{n}{b-a})$  了，因此最後的複雜度來到  $O(\frac{n}{\sqrt{b-a}})$
2. 由於我們使用 TruthTableOracle input string 長度一定要是 2 的冪次方，因此我們需要補 0 使得 input 長度滿足條件，另外原來的 input string 如果 0 和 1 一樣

多時，我們發現用 TruthTableOracle 得到 0 和 1 的機率是一樣的，這樣不符合我們想要得到 1 的結果，因此我們故意再補一倍的 0 在 input string 中，使得 output 出的結果是 1，但是處理 input string 會使得我們的複雜度從  $O(\sqrt{b-a})$  上升到  $O(b-a)$ ，結合前面的問題我們的複雜度來到  $O(N)$ ，跟古典的複雜度一樣，這樣子就失去的量子的優勢，但是如果我們不考慮字串處理的問題，量子還是具有比較少的複雜度。

3. 用量子的方法為了要減低錯誤率，我們每個量子步驟都重複執行好幾次以降低判斷錯誤的機率，這使得我們執行一次的時間在一分鐘左右，另外由於 IBMQ 要排隊等待，而且只有 5 個 qubit 的限制，因此以目前的狀況，用古典的方法執行 graph collision 還是比較有效率的。

## 心得

1. 做這個 project 上網查不到一樣的 code，全部都要自己重頭打，最後有寫出 quantum 版本，卻沒有辦法加速到  $O(\sqrt{N})$ ，可惜
2. 玩 qiskit 蠻有趣的，python 是強大的工具，謝謝老師、助教這學期的教學~
3. 論文中沒有說明 quantum oracle 的量子電路也沒有相關的數學式，僅用敘述的方式說可以用 grover search，這讓我們一開始為了建立 grover search 的oracle 感到相當頭痛，畢竟老師也說建造 quantum oracle 沒有一個 universal rule 可以去遵循，大部分都是靠 trial and error 的方式。最後我們使用了 truth table oracle 以避開 quantum oracle 的實作，才完成這次的 final project.
4. 對於 quantum 有了初步的認識，感覺還不錯。

## 參考資料

- [1] Magniez, F., Santha, M., Szegedy, M.: Quantum algorithms for the triangle problem. SIAM Journal on Computing 37(2), 413–424 (2007)
- [2] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Raitis Ozols, Juris Smotrovs. Parameterized Quantum Query Complexity of Graph Collision. arXiv:1305.1021
- [3] truth-tables-with-grovers-search-in-qiskit tutorial  
<https://quantumcomputinguk.org/tutorials/truth-tables-with-grovers-search-in-qiskit?fbclid=IwAR0vvGg0ch-ecO1raq1zmX4HBAhcAzTtzkg5cjwYMhfMLb71ZdOplBziMfM>

## 附錄

### 環境

python 3.9.7

pip 21.2.4

numpy 1.21.2  
turtle 0.0.1  
qiskit 0.30.0  
qiskit.aqua 0.9.5

### 程式執行方式

解壓縮 zip 檔, 用 jupyter notebook 打開 ./collision.ipynb 檔案, 點擊 Kernel／Restart & Run All 即可跑程式碼

### 可調參數

1. N:頂點數目
2. nearest\_connect、farthest\_connect:每個頂點最近和最遠相鄰頂點距離, 其中 $0 < \text{nearest connect} \leq \text{farthest connect} \leq \frac{N}{2}$
3. itr:量子演算法重複執行幾次, 重複越多次正確率越高, 但是時間也花越久, 另外itr 需要是奇數, 因為要避免 0 和 1 的數量相同
4. prob:每個頂點被標記的機率(為 1 的機率)
5. state:要在本地端跑(local)還是遠端跑(remote)
6. token:若要用 IBMQ 遠端跑, 需要設定 api token
7. plot\_graph:是(plot\_graph = 1) 否(plot\_graph = 0 )要用 turtle 繪圖程式將 graph 畫出來