2017 年臺灣國際科學展覽會
優勝作品專輯

作品編號　**190003**

參展科別　電腦科學與資訊工程科

作品名稱　確定有限狀態自動機與量子有限狀態自動機之間的轉換與比較

得獎獎項　大會獎：二等獎

就讀學校　臺北市立第一女子高級中學

指導教師　江介宏、黃芳蘭

作者姓名　官欣、陳穎

關鍵字　自動機、量子計算、資料轉換

作者簡介

我是陳穎（左），目前就讀北一女中三年級。高中時老師教我們寫程式，使我觸碰到資訊領域，開了眼界。遇見官欣後，發現我們都對量子計算有興趣，很幸運地找到一位願意指導我們的教授，開始研究計算理論。回首最初，我們已踏入第三個年頭，在教授與老師的指導下，我們在這條有點冷的路上越走越遠，收穫了太多東西。由衷感謝一路相伴的所有人！

我是官欣(右)，目前就讀北一女中三年級。國中的時候從未想過會就讀北一，更沒想過會進入數理資優班選擇資訊組，結果專題研究卻成為了高中一大美好的回憶。認識陳穎後，一起學習關於計算機的許多知識，很幸運地想到一個很有意思的研究題目，在教授的指導下不斷改進方向。走到現在，有著無盡的感謝，深深希望未來能有機會繼續研究相關的問題！

摘要

量子計算的效率相較傳統計算有指數級成長。然而此領域中多數研究皆專注於量子計算的性質本身，鮮少討論如何將傳統環境中的既有資訊轉換至量子環境下。一旦量子電腦實現，受量子效應限制，傳統資料多半不能相容於量子環境中。因此，本研究的目的是發想出一種系統性的演算法以確實跨越量子資訊與傳統資訊之間資料結構的藩籬。我們選擇的計算機模型是確定有限狀態自動機（Deterministic Finite Automaton，簡稱 DFA）。

本研究由自動機的轉移矩陣（Transition Matrix）及量子環境要求的可逆性（Reversibility）出發，自傳統 DFA 一步步轉換至量子有限狀態自動機（Quantum Finite Automaton，簡稱 QFA）並進行優化。最終，我們定義出一種新的 QFA 模型（**QDFA**）能在量子環境下運行，具有增大的字母表（Alphabet Set）但功能完全等價於 DFA（能辨認正則語言）。本研究獨創的演算法的時間複雜度為$\mathrm{O}(C\times N^2)$。

Abstract

Quantum computation can be exponentially more efficient than classical computation. While the studies in the field of quantum computing are focusing on the properties of quantum computation itself, the way of transforming classical computation into quantum computation is seldom discussed. Once quantum computer is realized, the quantum environment won't necessarily be compatible with classical information. Therefore, the target of this study is to create a systematic algorithm to assuredly cross the barrier of data structure between quantum and classical environment. We choose deterministic finite automata (DFA) as our computation model.

Starting from the transition matrices of finite automata and the property of reversibility required for quantum computation, this study discuss about the transformation from classical DFA to reversible DFA, then to quantum finite automata (QFA), and to quantum circuit as well, along with the optimization. Eventually, we define a new type of quantum finite automata model (**QDFA**), which is able to run under quantum environment with enlargement of the alphabet set, and is functionally equivalent to DFA (recognizing exactly the set of formal language). The time complexity of the algorithm this study originated is$\mathrm{O}(C\times N^2)$.

一、 前言

（一）、 研究動機

現今，積體電路越做越小，量子的效應將越來越不可忽視，電晶體的大小可能也即將接近半導體的物理極限[11]，於是由其他原理建構而成的電腦備受期待。在這些可能性之中，以使用量子效應作為計算的前提，發展出與傳統電腦截然不同的訊息處理方式的，即為量子電腦。由於量子力學的特殊性質，例如粒子可處於疊加狀態，使量子位元可同時以不同機率存在為 0 與 1 之間的狀態，這使得量子電腦之演算效率相當強大。這樣擁有詭譎特質的跨領域科技引起了我們極大的興趣。

如果利用量子的特性來建構電腦，與傳統電腦之間將會有許多演算法以及儲存資料方式的差異。以演算法為例，秀爾演算法（Shor's Algorithm）對因式分解的效率，能達成指數時間級別的提升。至於儲存方式，由於量子的疊加性，使每個儲存單位能容納的資訊亦呈現指數級成長[13]。

多數情況下，「量子電腦」指的是通用量子圖靈機（Universal Quantum Turing Machine），亦即可藉由模擬任何圖靈機達成任何任務。然而，此種模型雖然強大，但同時也相當複雜且成本巨大，也並不是所有情況下皆需要如此功能強大的模型；因此我們所關注的是一種功能較侷限，但成本較低的模型－－有限狀態自動機（Finite State Automata），以此作為基礎再加上我們自己的定義而創造出一種新的模型（**QDFA**）。

（二）、 研究目的

１、 藉由研究自動機的矩陣表達方式與狀態圖之對應關係，將確定有限狀態自動機（Deterministic Finite Automaton）轉換成量子有限狀態自動機（Quantum Finite Automaton）。

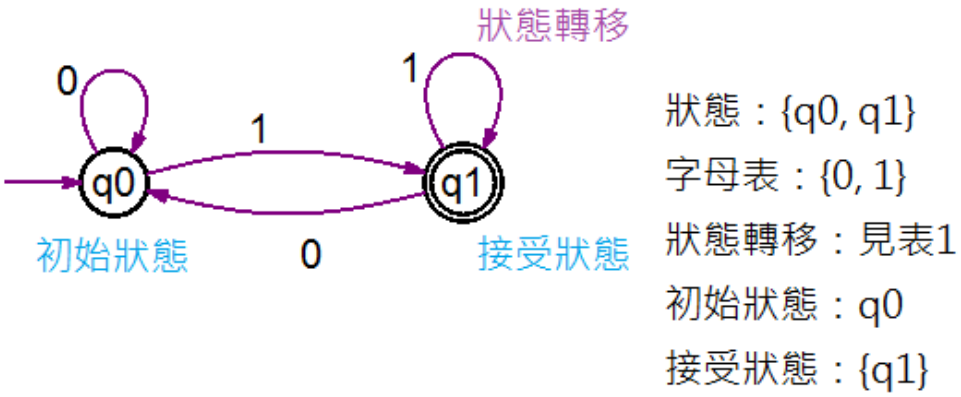
２、 使用量子邏輯閘合成轉換出來的自動機，以模擬其在量子電路中的運行，並進行演算法的優化。

二、 研究方法或過程

（一）、 背景知識探討

1 、 有限狀態自動機

（１）、 確定有限狀態自動機（Deterministic Finite Automaton，簡稱 DFA），是一種表現有限個狀態，以及狀態間轉換、動作的計算機模型。



（圖 1：確定有限狀態自動機M_1的狀態圖）

目前狀態 \ 輸入符號	0	1
q_0	q_0	q_1
q_1	q_0	q_1

（表 1：M_1的狀態轉移表）

a、 定義：

DFA 為一個 5-元組$< Q, \Sigma, \delta, q_0, F >$，且對於任一狀態$q$、輸入字符$a$，經過轉移函數$\delta$，將到達「唯一」的下一個狀態。

Q是狀態的集合。

Σ是輸入字符的有限集合，也稱字母表 (Alphabet)。

δ是轉移函數，$\delta: Q \times \Sigma \times Q \rightarrow$ C。

q_0是開始狀態，$q_0 \in Q$。

F是接受狀態的集合，$F \subseteq Q$。

表 1 對應的轉移函數可表示為：

$$\delta(q_0, 0, q_0) = 1 \quad -- (1)$$
$$\delta(q_0, 1, q_0) = 0 \quad -- (2)$$
$$\delta(q_0, 0, q_1) = 0$$
$$\delta(q_0, 1, q_1) = 1$$
$$\delta(q_1, 0, q_0) = 1$$
$$\delta(q_1, 1, q_0) = 0$$
$$\delta(q_1, 0, q_1) = 0$$
$$\delta(q_1, 1, q_1) = 1$$

舉 (1) 式為例，從狀態q_0至q_0，若輸入字符 「0」，存在一條轉移的邊。(2) 式則代表從狀態q_0至q_0，若輸入字符 「1」，不存在轉移的邊。

b、運作方式：

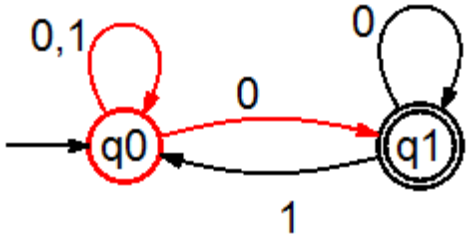
設定一個初始狀態，當給定一個字符串輸入，每讀入一個字符，自動機都會依據現在狀態（Current state）、轉移函數（Transition function）以及字符而決定將要轉移的目標狀態（Next state），如表 1 所示。

逐個讀取輸入中的字符，當字符耗盡，就稱自動機為停止。停止時的狀態，如果屬於接受狀態的集合（Set of accept states），稱呼這個自動機「接受」這個輸入。反之，則是拒絕這個輸入。

以自動機M為例，當輸入「011」時，狀態轉移的過程是$q_0 \rightarrow q_0 \rightarrow q_1 \rightarrow q_1$，最終停留在$q_1$，屬於接受狀態，判定為接受。反之，當輸入「010」時，狀態轉移的過程是$q_0 \rightarrow q_0 \rightarrow q_1 \rightarrow q_0$，最終停留在$q_0$，不屬於接受狀態，判定為拒絕。

（２）、非確定有限狀態自動機（Nondeterministic Finite Automaton，簡稱為 NFA）

也是一個 5-元組$< Q, \Sigma, \delta, q_0, F >$，但是對於任一狀態 q、輸入字符 a，經過轉移函數δ，可能到達「多個」下一個狀態。



（圖 2：非確定有限狀態自動機 M_2的狀態圖）

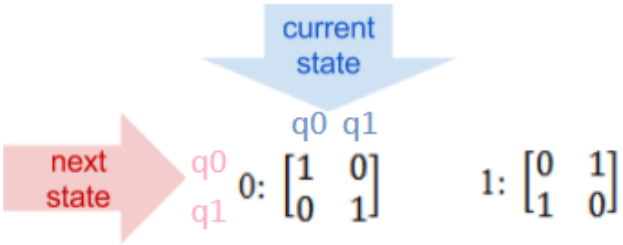
對於自動機M_2，在狀態q_0輸入字符 「0」時，下一狀態既是q_0，也是q_1。由於所有 NFA 都能轉換為 DFA [15]，故本研究僅討論 DFA。

以矩陣及行向量的角度表示自動機 [10]：

本研究討論如何將 DFA 轉換成能在量子電路（Quantum Circuit）中運行的 QFA，而量子電路中的一切運算為許多矩陣（比擬為量子閘）與一個行向量（比擬為量子態）相乘的結果。因此，我們以矩陣及行向量的角度表示所有自動機。

b、 字母表中的每個字符以**矩陣**方式表達其轉移：

以圖 1 的自動機M_1為例，總共有 0 與 1 兩種字符，各以一個矩陣表示使用到它的狀態轉移：

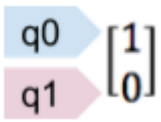


（圖 3：自動機M_1每個字符對應的轉移矩陣）

此時矩陣內的 1 與 0 代表可否轉移狀態，1 為是、0 為否；而矩陣的行代表現在狀態（current state），列代表目標狀態（next state），以編號照順序排列。

c、 現在狀態以**行向量**表達：

設一自動機M有n個狀態，則M的所有狀態以一個$n\times1$的行向量表示，其元素為一個 1 與多個 0。1 所在的位置即為目前自動機所處的狀態。



（圖 4-1：M_1處在狀態q_0時的行向量）

自動機尚未運作時，現在狀態即q_0，因此q_0在行向量中對應到的元素為 1，其餘元素為 0。

d、 相乘計算狀態轉移

輸入字串時，每讀入一個字符，就將此字符矩陣與行向量相乘，行向量即產生變化，相當於狀態轉移。

$$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}=\begin{bmatrix}0\\ 1\end{bmatrix}$$

（圖 4-2：M_1處在狀態q_0時，輸入符號 1 之結果為轉移至狀態q_1）

2、 語言

語言（Language）是一個字串集合。一個自動機M接受的語言 $L(M)$，即M可接受的所有字符串的集合。

（圖 5：形式語言示意圖）

（1）、 形式語言（Formal language）：

以數學方式建構的語言，包含正則語言（Regular language）、上下文無關語言（Context-free language）等等。其中由於正則語言與上下文無關語言都十分嚴謹，因此得以設計效率極高的演算法。

（２）、 正則語言（Regular language）：

a、 定義：

當一語言A被稱為正則語言，則必定存在一個確定有限狀態自動機M，使得$A = L(M)$。

b、 在字母表Σ中，正則語言的規律如下：

- 空集合ø是正則語言。
- 空字串$\{\varepsilon\}$。
- 對所有的字符$a \in \Sigma$，$\{a\}$都是正則語言。
- 若A、B是正則語言，則$A \cdot B$、$A \cup B$、A^*（kleene 星號，表示重複任意遍）都是正則語言。
- 除此之外，都不是正則語言。

c、 確定有限狀態自動機接受的語言都是正則語言，反之，任一正則語言都必定存在一個能接受它的確定有限狀態自動機。

３、 特殊矩陣

（１）、 么正矩陣（Unitary matrix）：令矩陣U，若$UU^\dagger = I$ （$U^\dagger$是$\underline{U}$的共軛轉置矩陣，I是單位矩陣），則U為么正矩陣。

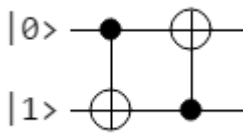
（２）、 排列矩陣（Permutation matrix）：每一列、行皆剛好只有一個 1，其餘皆是 0 的矩陣。排列矩陣皆是么正矩陣，且符合么正性質的(0,1)-矩陣皆為排列矩陣。

以矩陣 $U=\begin{bmatrix}0&1&0&0\\0&0&1&0\\1&0&0&0\\0&0&0&1\end{bmatrix}$為例，其共軛轉置矩陣$U^{\dagger}=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$，則$UU^{\dagger}=\begin{bmatrix}0&1&0&0\\0&0&1&0\\1&0&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&0&1&0\\1&0&0&0\\0&1&0&0\\0&0&0&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$，符合么正性質。

4、 量子電路（Quantum Circuit）[12]

量子電路對應於傳統電腦的電路圖，用以表示量子邏輯閘（Quantum Gate）對量子位元（Qubit）的運算。

以圖 6 為例，兩條橫線代表的即是量子位元，其上的特殊符號則是兩個量子邏輯閘。我們將在稍後的內文介紹常用的量子邏輯閘。



（圖 6：簡單的量子電路）

（1）、 量子位元：

傳統的位元僅有兩個狀態：0 或 1。量子位元則處於|0⟩與|1⟩之間的模糊態，是兩者的線性組合。因此一個量子位元的狀態可描述成：$|\varphi\rangle=\alpha|0\rangle+\beta|1\rangle$，且 $\alpha^2+\beta^2=1$。由於在此討論的是 DFA 轉換之 QDFA，因此α 與 β 之機率只可能是 1 或 0，因此一個量子位元只可能為|0⟩或|1⟩，即$\binom{1}{0}$或$\binom{0}{1}$。

（2）、 量子邏輯閘：

傳統電腦中，邏輯閘（如 AND、OR、NOT）是運算的最基本元件，在量子電腦中，則使用 NOT、CNOT、CCNOT 等可逆的邏輯閘來對量子位元做運算。

a、 NOT 閘

$$\text{NOT}=\begin{bmatrix}0&1\\1&0\end{bmatrix}$$

在量子計算的作用相當於傳統電路的 NOT，會對一個量子位元做反相變換。作用是$|0\rangle \to |1\rangle$、$|1\rangle \to |0\rangle$。

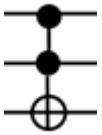
b、 Control-NOT 閘

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

作用於兩個量子位元，對一個量子位元進行變換。由 1 個控制位元（圖 7 以黑點表示），及 1 個受控的目標位元組成。若控制位元的值為 1，則目標位元進行反相變換。反之，則目標位元不變。控制位元本身不受影響。

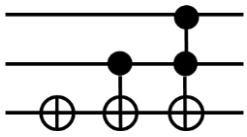
若將第一個量子位元設為「控制位元」，則作用是$|00\rangle \to |00\rangle$ 、$|01\rangle \to |01\rangle$、$|10\rangle \to |11\rangle$、$|11\rangle \to |10\rangle$。意即當第一個位元為 1 時，進行變換，否則不變換。

c、 Control-Control-NOT 閘（狹義的 Toffoli 閘）



$$\text{CCNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

作用於三個量子位元，對一個量子位元進行變換。由 2 個控制位元及 1 個目標位元組成。



（圖 7：NOT、CNOT、CCNOT）

d、 廣義的 Toffoli 閘

由 n-1 個控制位元及 1 個目標位元組成，n>3 時皆可化簡為 NOT、

CNOT、CCNOT 三種邏輯閘；換句話說，此三種量子閘的組合具備通用性，可以組成任意邏輯閘。

5、 量子有限狀態自動機

量子有限狀態自動機（Quantum Finite Automaton，簡稱 QFA）是假想自動機在量子世界中該有的表現而定義出的模型，出於不同的需求與觀點，不同研究中的定義也不盡相同。

量子計算領域蓬勃發展，Moore 與 Crutcheld 對於量子版本的語言進行探討，並定義 QFA 模型[10]。Kondacs 與 Watrous 研究 QFA 所接受的語言與正則語言之間的關係，並使用與前者不盡相同的 QFA 模型[8]。兩者最大的不同是，前者在自動機完成所有轉換後測量（稱為 Measure-once QFA），後者則在每次轉換後都進行測量（稱為 Measure-many QFA）[5]。

無論使用何種模型，基於量子電路中不得遺失任一位元資訊的原理，欲使任何轉換在量子電路（Quantum circuit）中運作，必須具備可逆的性質。以量子閘的觀點來看，即么正矩陣（Unitary matrix）[12]。

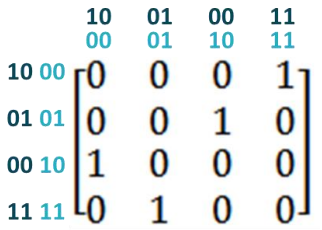
（1）、 自動機的執行：

自動機之每一字符以一組邏輯閘合成，由線路表示當前狀態，每輸入一字符即代表執行一組邏輯閘，線路之輸出即為目標狀態。

（2）、 狀態編碼（State Encoding）：

通常將么正矩陣轉換為邏輯閘時會自然將狀態依左至右、數字由小到大的二進位編碼，方便由線路表示狀態，但若將狀態的命名互相調換也並不會影響原先矩陣之資訊。

然而，因線路不會中途改變位置，故同一自動機之所有字符矩陣仍需使用同一狀態編碼。



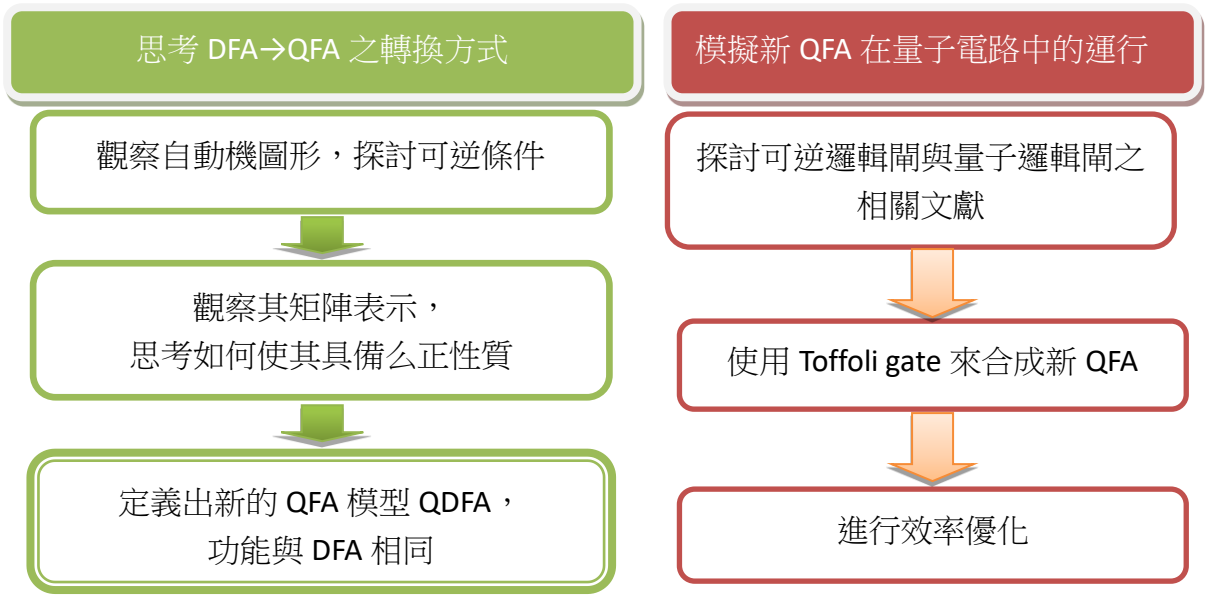
（圖 8：改變狀態編碼之矩陣）

（３）、成本（Cost）：

a、 邏輯閘數（Gate Count）：合成字符邏輯閘時，等價之表示方式不唯一，其中邏輯閘數量越多者，實現之成本越高昂。

b、 位元差（Bit Difference）：以二進位表示當前狀態與目標狀態時，兩者每有一位元之不同，位元差即加一，位元差越大時，資訊改變的程度越大。

（二）、研究流程



量子電路中的運算皆經由量子閘（Quantum gate，相當於傳統邏輯閘）組成，而量子閘有兩個特性：

1. 通常以矩陣表示。
2. 此矩陣是**么正矩陣**。

在滿足抽象的么正性質之前，我們先由自動機的狀態圖來觀察「不可逆」的路徑如

何產生，並想辦法解決，將 DFA 轉換成可逆的自動機。

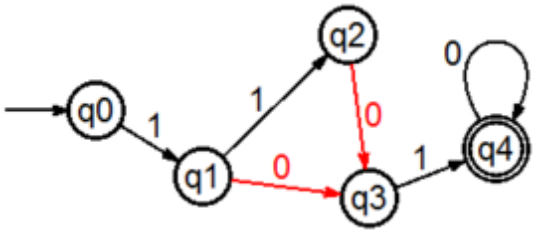
此後，再進一步將可逆的自動機轉換成符合么正性質（Unitary）的 QDFA，完成 DFA 與 QFA 之間的轉換。

1、 進行識別，使其來源可回溯，達成**可逆**。

由於量子的物理性質使得自動機必須是可逆的，即為不能喪失資訊，因此代表在自動機中的每個符號的矩陣必須符合么正性質（$U^{\dagger}U = UU^{\dagger} = I.$）。

從自動機圖形的角度，達成資訊不喪失的條件可解釋為在任一狀態給定上一個輸入的符號皆能回朔至前一狀態（可逆）。

因此，不能允許同時有兩個以上的狀態可輸入同一符號而轉換至同一狀態，如圖 6 所示。



（圖 9：不可逆的確定有限狀態自動機M_3）

以圖 9 之自動機M_3為例，狀態$q3$無法回溯上一狀態，因為在得到輸入符號為0的情況下，無法得知上一個狀態是$q1$或者是$q2$。以矩陣的角度來檢視，則是不符合么正性質。

$$0:\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&1&1&0&0\\0&0&0&0&0\\0&0&0&0&1\end{bmatrix}\quad 1:\begin{bmatrix}0&0&0&0&0\\1&0&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&1&0\end{bmatrix}$$

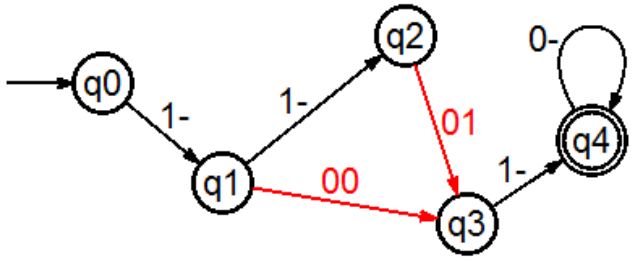
（圖 10：M_3時之轉移矩陣—不可逆）

因為不可逆由「同一字符」、「同一目標狀態」而來自不同狀態的轉移造成，又各字符矩陣的**每行**代表**現在狀態**，**每列**代表**目標狀態**，所以不可逆在各字符的轉移

矩陣中，對應的性質是同一字符矩陣的**同一列**中，有多個 1 存在（見圖 10）。

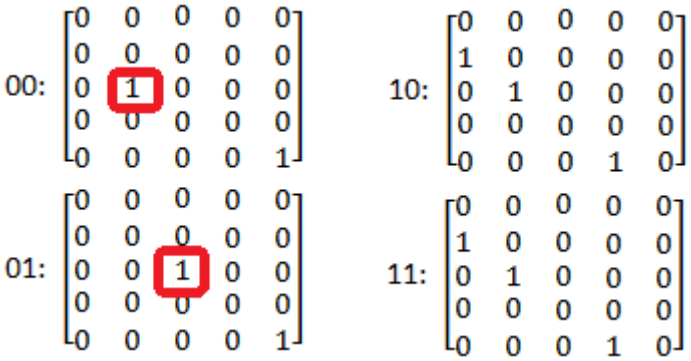
為了能加以區別兩者使資訊不喪失，我們想到能為原有字符添加位元以辨識不同的來源。

對於M_3而言，如果將圖 9 所標出的兩個轉移編上附加位元，一個標為 0，一個標為 1，則$q3$能清楚知道上一個狀態究竟是$q1$還是$q2$。至於其他沒有不可逆問題的轉移，則在其字符旁邊標上「-」，表示所有字母表內的字符（見圖 11）。

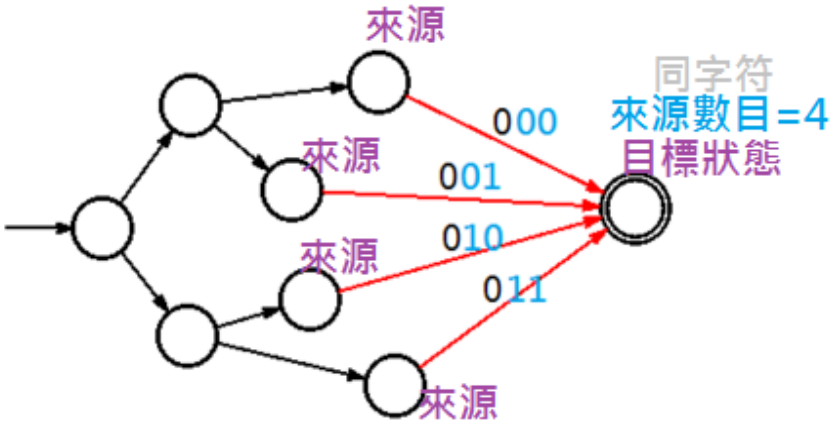


（圖 11：將M_3字符重新編碼，成為可逆的自動機）

亦可表示為將圖 7 之不可逆矩陣同列之 1 個以上的 1 分配至標記後兩個不同來源之同一字符矩陣（見圖 12）。



（圖 12：M_3可逆時之轉移矩陣）



（圖 13：同一字符有多個轉移指向同一目標狀態）

由此可知，對於一個目標狀態尋找同字符、不同來源的動作，相當於在一個字符的轉移矩陣中的一列中尋找值為 1 的元素。若有多個 1，就需要將其分配到不同矩陣，即是將字符重新編碼；若此列只有小於一個 1，則保留原本數值。

因此，我們得到將普通 DFA 轉為可逆 DFA 的演算法：

以表 2 舉例。

1. 得到 DFA 資訊$< Q, \Sigma, \delta, q_0, F >$後，對每個字符都運算出其在自動機中的轉移矩陣。

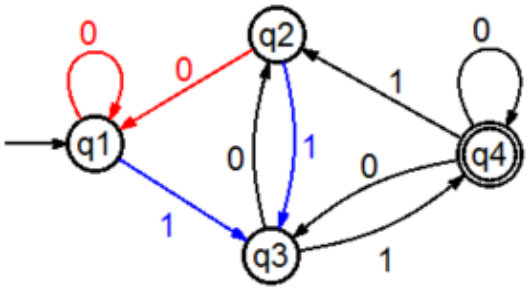
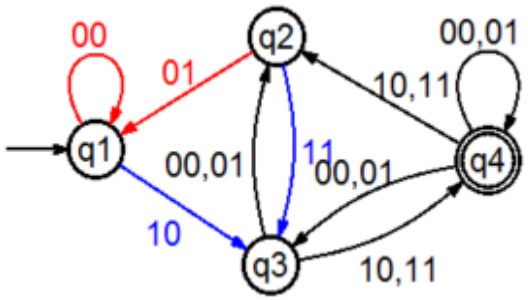
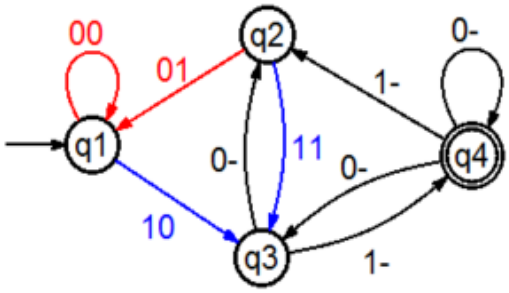
2. 對於各字符的轉移矩陣的每一列，逐行查看其值是否為 1，是則代表此行對應的狀態，在輸入此字符時，能到達此目標狀態，為其來源。記錄其來源數目。

3. 比較所有來源數目，得出 MaxSource，從而得出附加位元數目$\lceil log_2 MaxSource \rceil$。

4. 將所有字符編上附加位元，

5. 將各矩陣之各列中的多個 1，分配到重新編碼後各個不同的轉移矩陣。

（表 2：DFA 轉為可逆 DFA 之步驟）

自動機狀態圖	矩陣之實際運作
 自動機M_4之狀態圖。狀態q_0、q_1無法推知上一狀態。	0: $\begin{bmatrix}1&1&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$ 1: $\begin{bmatrix}0&0&0&0\\0&0&0&1\\1&1&0&0\\0&0&1&0\end{bmatrix}$ 得出 MaxSource=2，要增加 1 位附加位元。
 增添附加位元。	00: $\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$ 01: $\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$ 10: $\begin{bmatrix}0&0&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{bmatrix}$ 11: $\begin{bmatrix}0&0&0&0\\0&0&0&1\\0&1&0&0\\0&0&1&0\end{bmatrix}$ 分配原本同一列的 1 到新矩陣中。
 狀態圖中省略多餘的表示。	同上。

此種辦法在任意的 DFA 可行，其證明：

Finite Automata 只有有限狀態

⇒ 矩陣之中同一列最多只會有有限的 1 元素

⇒ 必定可以分配完成

2、 增添虛擬路徑，使轉移矩陣具備么正性質

首先，我們各字符的轉移矩陣皆(0,1)-矩陣。同時，具備么正性質的(0,1)-矩陣皆為排列矩陣。因此，我們欲使各矩陣具備么正性質，就要使其成為排列矩陣，即每一行、每一列都剛好有 1 個 1。

由於我們討論的是 DFA，因此每一行自然最多只有 1 個 1。先前做完轉換成可逆的步驟自動機具備可逆的性質後，任一字符矩陣的任一列最多只有 1 個 1。

然而，在完成分配後依舊無法保證能符合么正性質，因分配後可能產生一列有少於一個 1 元素的情形。

$$00:\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&\mathbf{0}&0&0\\0&0&0&1\end{bmatrix}\quad 01:\begin{bmatrix}0&1&0&0\\0&0&1&0\\\mathbf{0}&0&0&0\\0&0&0&1\end{bmatrix}$$

$$10:\begin{bmatrix}0&\mathbf{0}&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{bmatrix}\quad 11:\begin{bmatrix}\mathbf{0}&0&0&0\\0&0&0&1\\0&1&0&0\\0&0&1&0\end{bmatrix}$$

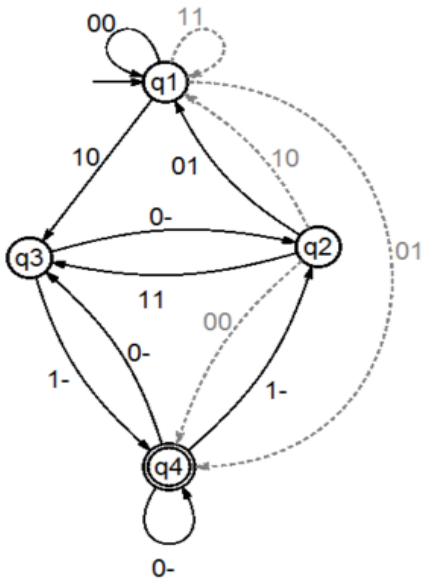
（圖 14-1：不符合么正性質的轉移矩陣）

此時，我們在矩陣中修改違反性質的元素，但在實際操作中禁止其執行（以圖 15 中的虛線表示）。

$$00:\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&\mathbf{1}&0&0\\0&0&0&1\end{bmatrix}\quad 01:\begin{bmatrix}0&1&0&0\\0&0&1&0\\\mathbf{1}&0&0&0\\0&0&0&1\end{bmatrix}$$

$$10:\begin{bmatrix}0&\mathbf{1}&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{bmatrix}\quad 11:\begin{bmatrix}\mathbf{1}&0&0&0\\0&0&0&1\\0&1&0&0\\0&0&1&0\end{bmatrix}$$

（圖 14-2：改正後的轉移矩陣）

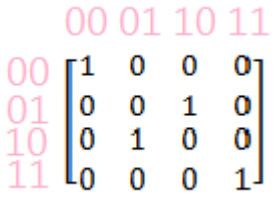


（圖 15：虛擬路徑以虛線表示，此自動機我們命名為 QDFA）

3、 以量子邏輯閘形式完整表達轉換的結果

在完成由確定有限狀態自動機至矩陣形式的轉換之後，研究以量子邏輯閘的形式表達並模擬自動機在量子環境下的運行，以驗證演算法的效率及可行性。

將 QDFA 轉以量子邏輯閘表達的方法參考自[9]，此前我們先將 QDFA 的各字符矩陣轉成真值表以便操作。以圖 16 中的字符矩陣為例，我們先將使用最直接的狀態編碼方式—各狀態「依序」由二進制編碼，接著根據狀態轉移，做出真值表。

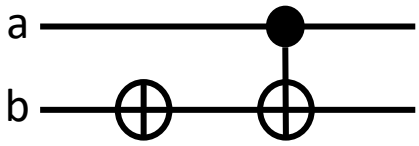


00	00
01	10
10	01
11	11

（圖 16：字符矩陣轉為真值表）

每次逐個狀態查看當前狀態是否與目標狀態不同，若不同，則操作相對應的邏

輯閘。每進行一次邏輯閘的操作，被影響的狀態範圍就往下減小，不影響到上面已轉換完成的狀態。如圖 17 所示，藍色標示的狀態即是未轉換完成的狀態。每增添一個邏輯閘，都遵循不影響已完成轉換的狀態的原則，只對下面的狀態做變換。



目標	ab	a'b'(NOT)	a''b''(CNOT)
00	01	00	00
01	10	11	01
10	11	10	10
11	00	01	11

（圖 17：自真值表轉換為量子邏輯閘）

4、 優化量子邏輯閘

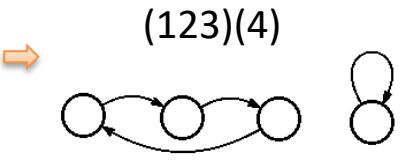
優化之目標為：在不改變原先自動機所有資訊下，使用最少之成本（Cost）完成 DFA 至 QDFA 之轉換與實現。

（1）、 應最小化何種成本：位元差（Bit Difference）還是邏輯閘數（Gate Count）？

在思考如何定義成本的最小化目標時，有兩個主要方向，即轉換過程中變動的位元數目，以及總共運用到的邏輯閘數目，而兩者不一定能同時減少。考慮量子邏輯閘實際成本之高昂，優先考慮優化後者。至於位元差雖可能減少能量耗損但優化效果較不顯著，列為次要條件。

（2）、 優化方式：根據置換群（permutation group）的概念。

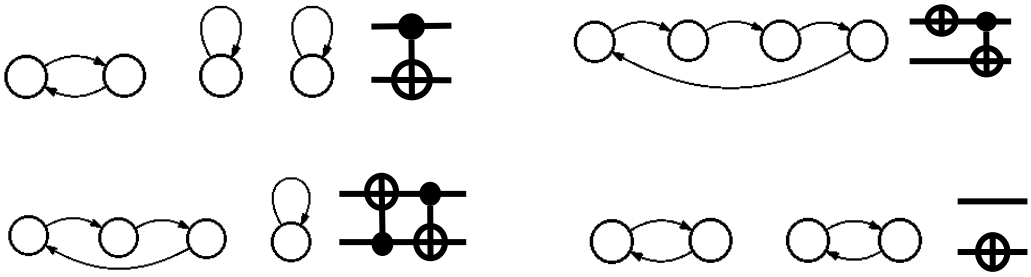
$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



（圖 18：一個字符矩陣以及其置換群）

我們發現可以先暫時不指定狀態編碼，畢竟名字可以另行指配。然而，狀態編碼卻會對所需的邏輯閘數目產生影響。因此，我們藉由矩陣畫出其置換群，由一個個循環所組成，而每一種圖形都有其最適合的狀態編碼方式。

我們可以尋找各種狀態圖形的規律，由狀態數目少的簡單圖形往複雜的圖形遞推。圖 19 是四個狀態時可能有的一些狀態圖，以及可用最少的邏輯閘數目來合成的組合方式。



（圖 19：四個狀態時的一些循環圖）

每一種組合方式都有其可對應的狀態編碼，且往往不只一種編碼方式，因此對於自動機的每一個字符矩陣，我們都能找出一列最佳的狀態編碼清單。若能於各字符間找到相符的最佳狀態編碼，比起原本的依序編碼方式，有優化的效果。

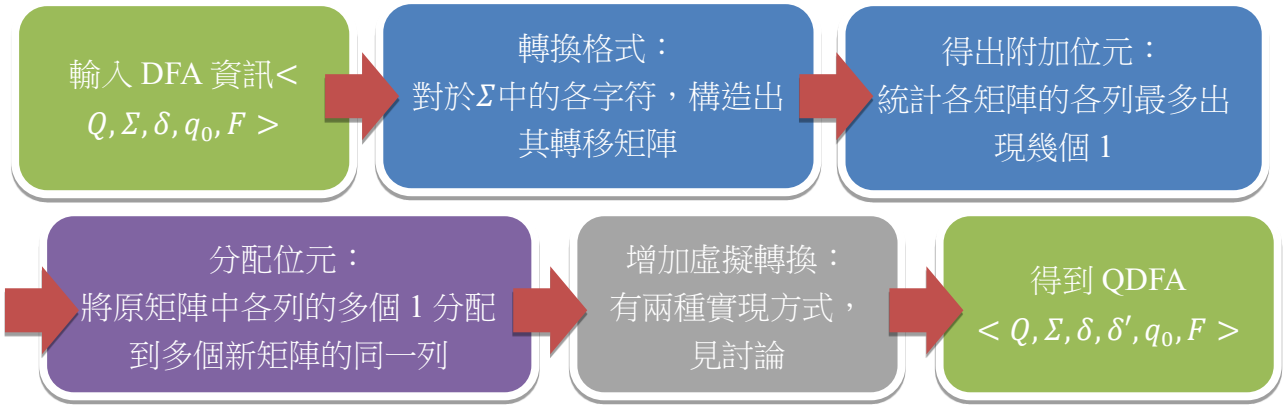
三、 研究結果與討論

本研究對確定有限狀態自動機進行轉換，使其能在量子電路運行。其中，藉由分配位元、添加虛擬路徑的操作，將確定有限狀態自動機轉換成具備量子有限狀態自動機性質的 QDFA。下一步，我們將 QDFA 的各字符矩陣化為真值表，將其轉換成量子邏輯閘。同時，我們也尋找出將 QDFA 轉換成量子邏輯閘的優化方式。

（圖 20：使 DFA 在量子電路運行之總流程）

（一）、 從 DFA 轉換至 QDFA

我們將 DFA 轉換成能在量子電路中運行的量子自動機，將其命名為「QDFA」。其定義為一個 6-元組$< Q, \Sigma, \delta, \delta', q_0, F >$，$\delta'$為禁止函數，其餘定義皆與 DFA 相同。



1、 轉換 DFA 格式：

藉已知 DFA 資訊，構造出字母表中各字符的轉移矩陣。

2、 得出附加位元：

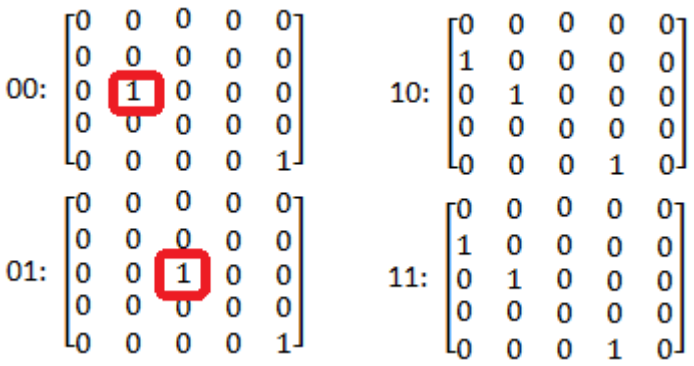
對各字符矩陣的各列，查看同列最多出現幾個 1，此數命名為 MaxSource。附加位元數目即$\lceil log_2 MaxSource \rceil$。

```
//陣列 Mat[symbol][state][state]為各字符之矩陣
//row: Next State, column: Current State
MaxSource = 0
for each symbol 's' (s = 0 ~ 2^inbit -1) do
        //inbit 為字母表之最大位元數
```

```
            for row = 0 to row = state-1 do
                count = 0
                for column = 0 to column = state-1 do
                    if Mat[s][row][column] == 1 then
                        count = count + 1
                    end if
                end for
                save[s][row] = count //下一步驟會需要
                MaxSource = max (MaxSource, count)
            end for
        end for
        AddBit = ceil(log2MaxSource)
```

3、分配位元：

將各列的 1 分配到不同矩陣中，這些矩陣由原本字符加上附加位元而產生。以先前的自動機M_3舉例操作。



（圖 21-1：M_3可逆時之轉移矩陣）

```
bool used[state], over
//used 記錄此行是否已經用過 1，over 表示此列已用過 1
//i: 字符原始編碼, j: 增添位元後的編碼
for i = 2^inbit-1 to i = 0 do
    set all elements in used[] to 0
    for j = (i+1) ×2^Addbit-1 to j = 2^Addbit do
```

```
            for row = 0 to row = state-1 do
                if save[i][row]>1 then
                        over=0
                        for column = 0 to column = state-1 do
                            if Mat[i][row][column]==1 and used[column]==false
                                    and over==false then
                                    Mat[j][row][column]=1
                                    used[column]=over=1
                            else Mat[j][row][column]=0
                            end if
                        end for
                    else copy Mat[i][row] to Mat[j][row]
                    end if
                end for
            end for
        end for
```

4、 增加虛擬轉換：

將新矩陣中各行各列缺少的 1 補齊，使其成為排列矩陣，並標示虛擬路徑「不可通過」。

00: $\begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ 10: $\begin{bmatrix} 0 & 0 & 0 & 0 & \mathbf{1} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

01: $\begin{bmatrix} 0 & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ 11: $\begin{bmatrix} 0 & 0 & \mathbf{1} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

（圖 21-2：M_3符合么正性質時之轉移矩陣）

```
for each symbol ‘s’ (s = 0 ~ 2^(inbit +AddBit) -1) do
```

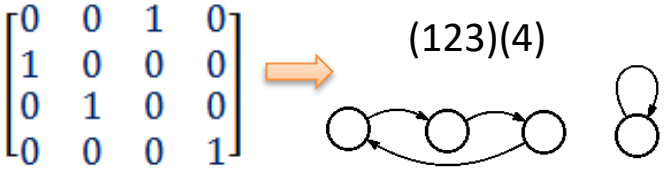
```
            count = 0
            for row = 0 to row = state-1 do
                if used[count] == true then
                    count = count+1
                end if
                Mat[s][row][count] = 1
                count = count + 1
            end for
        end for
```

（二）、 QDFA 至量子邏輯閘

QDFA 已可在量子環境下運行，為了使實際操作達到最低成本，轉換至邏輯閘的過程中同時考慮優化。

１、 由矩陣畫出狀態圖

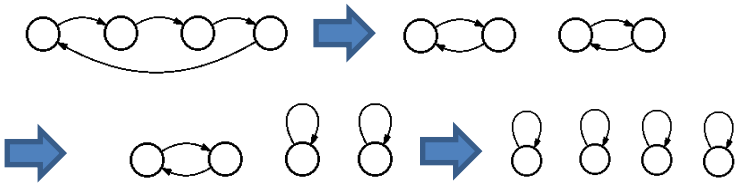
由矩陣最左邊之狀態開始，暫設為第一狀態，至目標狀態曾經出現為止為一循環；可以以群論中的置換圖形或排列表示。

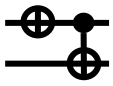


（圖 22：矩陣至狀態圖）

２、 由狀態圖找出最適狀態編碼

（１）、 狀態圖圖形：循環長度（Circle Length）較短之狀態圖圖形可輕易推導出最佳解，較長之循環則可以遞迴向上推導。





（圖 23：由長度 4 之循環解構至單循環）

（２）、共同排列（Common Permutation）：同時考慮自動機中的所有字符，存在同一循環中之狀態排列即為，出現越頻繁則須考慮的比重越重。

a: (12345)(678)
b: (12)(45)(3678)
c: (213)(45678)

（圖 24：a b c 三個字符之共同排列）

NOT 可將所有狀態兩兩置換，CNOT 可將所有狀態之 1/2 兩兩置換，CCNOT 可將所有狀態之 1/4 兩兩置換（見圖）。依以上之條件選出成本最低廉的狀態編碼（State Encoding）而可達成優化效果。

NOT	CNOT	CCNOT
(12)(34)(56)(78)	(1)(24)(3)(5)(68)(7)	(1)(2)(3)(4)(5)(6)(78)
(13)(24)(57)(68)	(1)(26)(3)(48)(5)(7)	(1)(2)(3)(48)(5)(6)(7)
(15)(26)(37)(48)	(1)(2)(34)(5)(6)(78)	(1)(2)(3)(4)(5)(68)(7)
	(1)(2)(37)(48)(5)(6)	
	(1)(2)(3)(4)(56)(78)	
	(1)(2)(3)(4)(57)(68)	

（圖 25：8 個狀態情況下 Toffoli Gate 可代表之置換）

（三）、演算法之正確性

任意 DFA 以矩陣型態表現時，其各個字符的對應矩陣可藉由「分配位元」操作，轉為可逆 DFA。再藉由「添加虛擬轉換」操作，轉為 QDFA。

證明：

DFA 以矩陣型態表現時，定義為各個字符皆有一對應矩陣，其直行代表當前狀

態，橫列代表目標狀態，以 0 與 1 表達在輸入此字符的情況下各狀態之間的轉移是否可行，0 為 False，1 為 True。

為了使 DFA 能於量子環境下執行，其矩陣型態必須符合 Unitary 性質，又因 DFA 之矩陣皆僅出現 0 與 1，故 QDFA 之矩陣型態必須皆為排列矩陣。

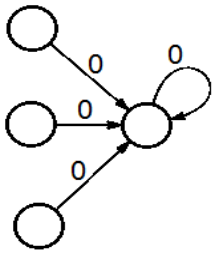
根據 DFA 的確定性質，每次轉移狀態時僅會轉移至一個以下的狀態（下一狀態或直接拒絕），故其所有字符的對應矩陣中之每一直行皆僅會包含一個以下的 1。⇒只需考慮分配橫列中的 1。

當某一狀態出現同時輸入同一字符的不同來源箭頭時，其矩陣表示型態會在該字符的對應矩陣的橫列中出現一個以上的 1；此時為了符合 Unitary 性質，將原先橫列有多個 1 的字符依照 1 的個數加上編碼，分配為不同的矩陣，定義此操作為「分配位元」。此時原先的字符於自動機內部分散為多個字符，每個字符的對應矩陣的每個橫列皆分配到一個 1，由於自動機的狀態有限，其字符的矩陣表示型態之行數與列數亦有限，故一定能完成分配。

（四）、 效率分析

1、 由 DFA 至可逆 DFA：

設字母表中有C種字符，狀態數目是N。



（圖 26：眾多同樣字符的轉移指向同一狀態）

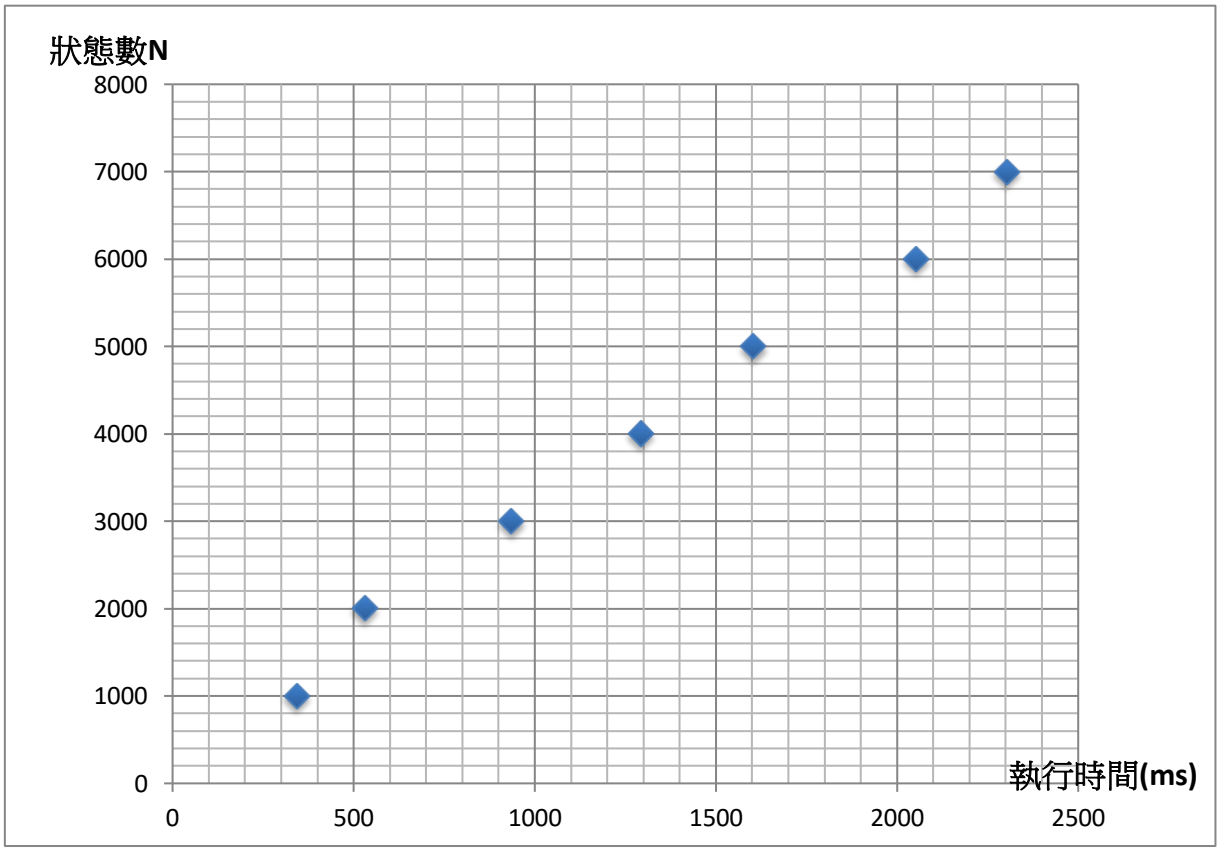
依據前文所述的演算法，若出現如圖 26 的極端情況，字母表的大小會擴增到$O(C\times N)$的層級。同時，每個字符都要儲存一個N^2的矩陣，因此空間複雜度為$O(C\times N^3)$。為了填滿這些矩陣，所需的時間複雜度也是$O(C\times N^3)$。.

然而在真正實作時，會發現這些字符矩陣儲存許多不必要的資訊，都是 0 多 1 少的稀疏矩陣。事實上，根據 DFA 的性質，同一直行最多只能有 1 個 1，因此在矩陣的N^2個元素中最多只會有N個 1。當N的數目越大，1 的數量所能占的比例越小。

字符	轉換
0	0→1, 1→2
1	0→2

（表 3：存下各字符在自動機中的轉移路徑）

因此，在這階段的轉換過程中，只需儲存各字符在自動機中有哪些轉移路徑即可。如此，空間複雜度為$O(C\times N^2)$。既然已無需填滿矩陣，則只需掃過C個字符的各轉移路徑，一一分配到擴增後的字符的轉移路徑隊伍中，所需時間複雜度為$O(C\times N)$。



（圖 27：字母表大小為 1024 時，狀態數N與執行時間的對照圖）

圖 27 為使用普通 Windows 7 作業系統的桌上型電腦，對於初始字母表大小為 1024 的 DFA 進行轉換的程式執行結果。之所以將字母表大小設為 1024 是認為這個數字足夠大，可以看出不同狀態數之間，執行時間的顯著差距。輸入程式的 DFA 是依據「使 MaxSource」最大化的原則生成。由圖 27 可見，這一階段的轉換所需的時間與N大略呈線型關係。

２、 由可逆 DFA 至 QDFA：

QDFA 如果使用矩陣的方式表示，則其每個字符矩陣都是排列矩陣。若依據此種資料儲存方式，空間複雜度是$O(C\times N^3)$，而轉換所需的時間複雜度是$O(C\times N^2)$。對於$O(C\times N)$個字符矩陣，只要如上文的虛擬碼一樣儲存「每一直行是否已有 1 存在」，就能以$O(N)$的時間完成轉換，因此是$O(C\times N^2)$。

但是 QDFA 也不需要真的以排列矩陣的形式儲存，同樣地，只要存下各轉移路徑即可。每個字符的轉移路徑隊列都要填滿至N個轉移路徑，因此空間複雜度為$O(C\times N^2)$，時間複雜度是$O(C\times N^2)$。這一步所需的時間複雜度相較於前一步增長了N倍，是因為字母表大小已擴增到$O(C\times N)$。

（四）、 討論

１、 QDFA 語言與 DFA 之比較

由於先前已證明任意 DFA 都能轉換成 QDFA，因此 QDFA 可接納的語言與 DFA 能接納的語言完全等價，即正則語言。

２、 與過往研究之比較

此領域近期的最相關研究是一篇 2015 年的文章，其中定義了一種新的模型 REV-DFA，是針對可逆 DFA 的研究，旨在尋找將 DFA 轉為可逆 DFA 的方法[7]。不同於本研究之方法，這篇文章選擇從複製自動機狀態的角度出發。

至於 QFA 領域中最常被提到的 Measure-once QFA[10]以及 Measure-many

QFA[8]，雖然效率較高，卻因為利用機率疊加的特性，因此其接納的語言僅為正則語言的子集合。這兩種 QFA 的相關研究都是基於假設量子電腦存在的前提下，對 QFA 可接受的語言集合進行研究。本研究則提供將傳統電腦的 DFA 轉換成 QDFA 的方式，且能接納正則語言，與 DFA 等價。

3、 避開虛擬路徑之機制

有兩種方式可以避開虛擬路徑：

（1）、 進行轉換操作前，先進行 Completion（使自動機在所有狀態下對任何輸入的字符皆有對應的目標），雖然在轉換後便不再 Complete，但這是對內部情況而言，對外界而言任何輸入的字符皆有對應的箭頭。

（2）、 QDFA 可由外界電子零件控制，一旦遇見虛擬路徑即保證被拒絕（Reject），因不存在的箭頭皆指向受困狀態（Trap State，輸入任何字符皆指向自己）。而為了判別當前操作是否被禁止，需在每輸入一個符號後測量，近似於 Measure-many QFA 模型。

4、 合成量子電路的其他優化方式

（1）、 根據真值表（改變狀態編碼）：

找出所有狀態編碼之中，位元差（Bit Difference）最少之編碼。這亦為格雷碼（Gray Code）問題。

原始	目標	目標
00	10	01
01	11	11
10	01	00
11	00	10
位元差	**6**	**4**
邏輯閘	NOT+CNOT	NOT+CNOT*3

（圖 28：以減少位元差做優化）

見圖 28 可發現，以減少位元差為目標而改變狀態編碼後，卻使得邏輯閘數增加了，故知兩者可能產生衝突，又因目前邏輯閘之成本明顯較為高昂，因此我們暫時捨棄以位元差作為優化條件。未來可能以此為次要之條件考慮更進一步的優化演算法。

（２）、根據轉移矩陣（改變虛擬位元）:

在矩陣中出現 2 個以上的缺少 1 時，虛擬位元之可能位置便有 2 個以上的組合。

$$\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix}$$

$$\Rightarrow \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \text{ or } \begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix}$$

（圖 29：虛擬位元之位置可有 2 種組合）

改變虛擬位元的位置等於在過程中改變矩陣本身，如同改變狀態編碼一般能減少成本，但與狀態編碼不同，是各個矩陣獨立。若嘗試與狀態編碼配合，將會大量增加優化可能的多樣性及複雜性。

四、 結論與應用

本研究定義出一種新的模型 QDFA，可以在量子電路中運行，同時也能接受與 DFA 等價的語言。由於是從傳統電腦的世界搭一棟通往量子計算的橋樑，我們的轉換方式都是利用傳統電腦的邏輯，以能用傳統方式完成轉換。

轉換成量子有限狀態自動機的 QDFA，我們則討論如何優化量子邏輯閘的合成方式，使其不但能在量子電路運行，且更有效率。針對 QDFA 的特質而設計出的優化方式，能配合由傳統 DFA 轉換而成的 QDFA 使其不喪失資訊，以接納正則語言。

目前，為數不少的研究者正在研究並嘗試實現量子電腦，我們相信這總有一天會成功，然而過往長久所累積下的資訊卻不能直接移植至量子電腦上沿用，因此需要轉換的手續；我們的研究能使得未來量子電腦實行時，得以轉移既有的自動機資訊至新的形態。

五、 參考文獻

[1] Ambainis, A., & Freivalds, R. (1998). 1-way quantum finite automata: strengths, weaknesses and generalizations. Proceedings of the 39th Annual Symposiumon Foundations of Computer Science, 332-342.

[2] Ambainis, A., Bonner, R., Freivalds, R., & Kikusts, A. (1999). Probabilities to accept languages by quantum finite automata. Lecture Notes in Computer Science, 1627, 174-183.

[3] Ambainis, A., Kikusts, A., & Valdats, M. (2001). On the class of languages recognizable by 1-way quantum finite automata. Lecture Notes in Computer Science, 2010, 75-86.

[4] Ambainis, A., Nayak, A., Ta-Shma, A., & Vazirani, U. (2002). Dense Quantum Coding and Quantum Finite Automata. Journal of the ACM, 49(4), 496-511.

[5] Brodsky, A., & Pippenger, N. (2002). Characterizations of 1-Way Quantum Finite Automata. SIAM Journal on Computing, 31(5), 1456-1478.

[6] Ciamarra, M. P. (2001). Quantum reversibility and a new model of quantum automaton. Fundamentals of Computation Theory, 2138, 376-379.

[7] Holzer, M., Jakobi, S., & Kutrib, M. (2015) Minimal Reversible Deterministic Finite Automata. Developments in Language Theory, 9168, 276-287.

[8] Kondacs, A. & Watrous, J. (1997). On the Power of Quantum Finite State Automata. Proceedings of the 38th Annual Symposium on Foundations of Computer Science, 66-75.

[9] Miller, D. M., Maslov, D, & Dueck, G. W. (2003). A Transformation Based Algorithm for Reversible Logic Synthesis. Proceedings of the 40th annual Design Automation Conference, 318-323.

[10] Moore, C., & Crutchfield, J. (1997). Quantum Automata and Quantum Grammars. Theoretical Computer Science, 237, 275-306.

[11] Moore, Gordon E. (1965). Cramming More Components onto Integrated Circuits. Proceedings of the IEEE, 86(1), 82-85.

[12] Nielsen, M. A., & Chuang I. L. (2000). Quantum Computation and Quantum Information.

Cambridge, United Kingdom: Cambridge University Press.

[13] Rieffel, E. (2000). An Introduction to Quantum Computing for Non-Physicists. Quantum Computers and Quantum Computing, 1(1), 4-57.

[14] Saeedi, M., & Markov, I. L. (2013). Synthesis and Optimization of Reversible Circuits - A Survey. ACM Computing Surveys, 45(2), 21-54.

[15] Sipser, M. (1997). Introduction to the Theory of Computation. Boston, USA: PWS Publishing Company.

[16] West Michigan University. Chomsky Presentation. Retrieved from https://www.cs.wmich.edu/~bhardin/cs4850/ChomskyPresentation.pdf

[17] Yakaryilmaz, A., & Cem Say A. C. (2010). Languages recognized by nondeterministic quantum finite automata. Quantum Information & Computation, 10(9), 747-770.

[18] 演算法筆記（無日期）。 Language ，檢自：http://www.csie.ntnu.edu.tw/~u91029/Language.html#1

六、 附錄

由 DFA 至可逆 DFA 的實驗用程式

```
#include<cstdio>
#include<ctime>
#include<vector>
#include<algorithm>
using namespace std;
#define x first
#define y second
typedef pair<int,int> pii;
const int maxbit = 25;
int inbit, trans, state, MaxSource, AddBit;
vector<pii> v[1<<maxbit];

int main(){
    clock_t T1, T2, T3, T4;
    T1 = clock();
    freopen("input.txt","r",stdin);
    freopen("output.txt","w",stdout);

    char ts[maxbit+1];
```

```
int cur, nxt, tmp, M=0;
double spend=0;
scanf("%d %d %d",&inbit,&trans,&state);
for(int i=0; i<trans; i++){
    T3=clock();
    scanf("%s %d %d",ts,&cur,&nxt);
    T4=clock();
    spend+=T4-T3;
    tmp=0;
    for(int j=0; j<inbit; j++){
     tmp= tmp*2 + ts[j]-'0';
     }
    M=max(M, tmp);
    v[tmp].push_back(make_pair(nxt, cur));
}
for(int i=0; i<=M; i++){
 sort(v[i].begin(), v[i].end());
 tmp=1;
 for(int j=1; j<v[i].size(); j++){
     if(v[i][j].x!=v[i][j-1].x) tmp=1;
     else tmp++;
     MaxSource= max(MaxSource, tmp);
     }
 }
 tmp=1, AddBit=0;
while(tmp < MaxSource){
    AddBit++;
    tmp*=2;
}
printf("MaxSource= %d, AddBit= %d\n",MaxSource, AddBit);

for(int i=(1<<inbit)-1; i>=0; i--){
 tmp= i<<AddBit;
 int cnt=1, len=v[i].size();
 if(len==0) continue;

 pii last=v[i][len-1];
 v[i].pop_back();

 for(int j=len-2; j>=0; j--){
```

```
            pii P=v[i][j];
            if(v[i][j].x!=v[i][j+1].x){
                cnt=0;
                v[i].pop_back();
                v[tmp+cnt].push_back(P);
                }
            else {
                v[i].pop_back();
                v[tmp+cnt].push_back(P);
                cnt++;
                }
            }
            v[tmp].push_back(last);
    }
        T2 = clock();
    printf("time= %.0lfms\n", T2-T1-spend);
     return 0;
}
```

【評語】190003

1. 本件作品有紮實的理論基礎，並提出創新的 DFA 轉 QDFA 演算法，值得肯定。

2. 建議提供正確的演算法複雜度分析。

Intel 特別獎評語：

1. 很好且很新的題目。

2. 運用新的演算法轉換量子（模糊）的資料為確定數據的資料。

3. 很有未來性，但須將較少人研究的量子電腦解釋清楚。

4. 也要想想應用面。

5. 評審建議；如何將量子電腦放到 FPGA。

6. 建議增加說明：Quantum computing 裡何時適合用 finite state machine 來模擬？