行政院國家科學委員會專題研究計畫　成果報告

量子計算理論之研究(I)

計畫類別：個別型計畫
計畫編號：NSC93-2213-E-009-117-
執行期間：93 年 08 月 01 日至 94 年 07 月 31 日
執行單位：國立交通大學資訊工程學系(所)

計畫主持人：蔡錫鈞

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 94 年 10 月 27 日

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告
□期中進度報告

量子計算理論之研究(I)

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC93-2213-E009-117
執行期間： 93 年 8 月 1 日至 94 年 7 月 31 日

計畫主持人：蔡錫鈞
共同主持人：
計畫參與人員： 吳信龍。

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、
列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：

中 華 民 國 94 年 10 月 25 日

中文摘要

本計畫將從計算理論(computational complexity)的角度研究量子計算(quantum computation)的計算能力。吾人期望找尋一般量子計算中的複雜度等級(complexity class),即 **BQP**，相對應於一般傳統計算機之隨機演算法(probabilistic computation)之複雜度等級,即 **BPP**，之間的集合包含關係。若能找出一問題屬於 **BQP**，但不屬於 **BPP**，則將可確定量子計算確實優於傳統計算機模型。反之，若可證明 **BQP**=**BPP**，則顯示量子計算將失去價值。以現有之計算理論對於量子計算之初步研究結果顯示，對於大多數的布林函數，量子計算均無法如同「質因數分解」問題一般給出指數倍加快(exponential speed-up)，由於實現量子計算之成本極鉅，吾人希望藉由計算理論之角度，仔細探討量子計算之真實能力。

研究方向將包括:

1. 以通訊複雜度(communication complexity)角度,研究多台量子電腦間合作解決問題時傳輸所需耗費的資料量大小，其中吾人將著重於研究「指紋法(fingerprinting method)」及相關問題之通訊協定所能處理的範圍。
2. 以計數複雜度(counting complexity)角度，並配合「相對化(relativized)」的證明技巧，吾人期望能定義新的複雜度等級(complexity class)使 **BQP** 之位階能被更加確定。並將試圖發掘「非相對化(unrelativized)」之結果，以便更加真實描繪 **BQP** 與其餘複雜度等級之包含關係。
3. 以決策樹複雜度(decision tree complexity)角度，了解一般布林函數轉化為決策樹後之查詢複雜度(query complexity)，並試圖改進現有理論之下限邊界值(lower bounds)。

在後半部的計畫，我們探討了排列之嵌合問題(permutation array embedding)，並給出相對應之演算法與結果。

關鍵詞: 量子計算、計算理論、通訊複雜度、計數複雜度、決策樹複雜度、排列嵌合問題

英文摘要

This research will study the computational power of quantum computation, from the computational complexity-theoretic point. We want to find the set relationship among the common complexity class of quantum computation, **BQP**, and its probabilistic counterpart in classical computation, **BPP**. It we could find a problem, which resides in BQP, but not in BPP, then quantum computation is really a superior computation model than traditional computers. On the other hand, if we could show that **BQP = BPP**. Then quantum computation would be of not much use since it could be done in classical ways.

The results to date show that for most of Boolean functions, quantum computer could not give exponential speed-ups as it does on the integer prime-factoring problem. Due to the high cost of realizing quantum computers physically, we want to study the true power of quantum computation in order to predict its usefulness.

Our research would focus on:

1. Communication complexity: we want to study the communication resources required on data transmission between quantum computers. We would especially focus on the "fingerprinting method", and find the set of problem, which it could be applied to.
2. Counting complexity: with the prove method of relativization, we would like to define a new complexity classes in which the role of BQP with existing classes could be understood better. We will also pursuit unrelativized results in which more accurate relationships could be devised.
3. Decision tree complexity: we would want to more of the query complexities of general Boolean functions under the quantum computation model, and study on the possibilities of improving the existing bounds.

In the rest of part of this project, we studied the problem of permutation array embedding. Moreover, we have obtained some results for its lower bounds.

Keywords: quantum computation, computational complexity, communication complexity, counting complexity, decision tree complexity, permutation array embedding

目錄

1 Background

1.1 Overview

This research studies the power of quantum computation in the eye of computational complexity. We examine the set inclusion relation of one complexity class in quantum computation, BQP, with its classical counter part in probabilistic computation, BPP. If one could devise a computational problem which resides in BQP but not in BPP, then we could conclude that quantum computation is surely a superior model than traditional Turing Machine. On the other hand, quantum computation would lose its advantage if $BQP = BPP$.

During the eighties of the 20th century, physicists encountered various computational problem in statistical physics of which efficient algorithms could not been found. What's worse, classical computers seemed not being able to simulate quantum physics in a reasonble speed. Therefore the concept of building computers in the light of quantum physics, the so-called "quantum computers", began to emerge. It had been shown that quantum computers outperform traditional computers in the problem of integer factoring and database search [14][18] and it could provide exponential speedup in the integer factoring problem, which is unlikely in classical case. Therefore researches on the computational power of quantum computation become a interesting topic in the society of theoretical computer science from the nities.

One of the main topic of theoretical computer science is to classify the different computational resource requirements between different models of computer architectures [6] [19] [13]. On the model of deterministic Turing Machine, many common problems lie in the class of "NP-complete", in which efficient algorithms do not exist yet.

We will use three different approaches (not strictly restricted in them) for studying the computational complexity properties of quantum computation. They are:

1. Counting complexity
2. Communication compleixty
3. Decision Tree complexity

Below we give short introduction to each of these approaches.

1.2 Counting complexity

In computational complexity, we usually ask two related styles of problems: One asks whether a desired solution exists; the other requires that a solution be produced. But there is a third important, natural, and fundamentally different kind of problem: Th one that asks *how many solutions exist.* Counting complexity [11] is the general name for this kind of problems. Usually we want to find its relation to the complexity of nondeterministic and probabilistic computation.

Counting complexity won't do much help in designing algorithms. The main influence of it is to classify existing computational problems in that we could gain more insights on the relationships between them. Many complexity classes were defined in counting complexity to describe various computational ability. Among them, $\#P$ [21] and $GapP$ [9] are the most studied ones.

1.2.1 $\#P$ and $GapP$

We first give the definitions of $\#P$ and $GapP$. Before this we should give definitions of *counting machines (CM).* Simply put, a counting machine is an NP machine. It runs in polynomial time and moves nondeterministically with two halting states: *accepting* and *rejecting.* And we insists that any computation path of a CM must end in one these two states. We emphasize that the machine's acceptance criterion is based on the *number* of accepting and/or rejecting paths.

Definition 1 *Let M be a CM. We define the function $\#M : \Sigma^* \to \mathbb{Z}^+$ to be such that for all $x \in \Sigma^*$, $\#M(x)$ is the number of accepting computation paths of M on input x. Similarly, $Total_M : \Sigma^* \to \mathbb{Z}^+$ is the total number of computation paths of M on input x. The CM $\overline{M}$ is the machine identical to M but with the accepting and rejecting states interchanged (thus $\overline{M}$ rejects whenever M accepts and vice verse).*

We observe that for all $x \in \Sigma^*$:,

$$\#M(x) + \#\overline{M}(x) = Total_M(x) = Total_{\overline{M}}(x)$$

and $\#\overline{M}(x)$ is the number of rejecting paths of M on input x.

Definition 2

$$\#P \equiv \{\#M | M \textit{ is a CM}\}$$

.

Definition 3 *If M is a CM, define the function $gap_M : \Sigma^* \to \mathbb{Z}$ as follows:*

$$gap_M \equiv \#M - \#\overline{M}$$

.

The function gap_M represents the "gap" between the number of accepting and the number of rejecting paths of M.

Definition 4

$$GapP \equiv \{gap_M | M \text{ is a CM}\}$$

.

$\#P$ was developed in measuring the complexity of computing permanent of a matrix. This problem was proved to be $\#P$-complete. Some natural decision problems with simple answers have their counting counterparts being $\#P$-complete, while some of the counting problems related to NP-complete problems are fairly easy.

$GapP$ holds many closure properties in that it could be used to simplify many existing proofs on the set inclusion/exclusion relationships among complexity classes. The following are some of the most important closure properties of it.

Property 1 *$GapP \diamond FP = GapP$ and $FP \subseteq GapP$.*

Property 2 *If $f \in GapP$ then $-f \in GapP$.*

Property 3 *If $f \in GapP$ and q is a polynomial, then the function below is in $GapP$.*

$$g(x) \equiv \sum_{|y| \leq q(|x|)} f(< x, y >)$$

Property 4 *If $f \in GapP$ and q is a polynomial, then the function below is in $GapP$.*

$$g(x) \equiv \prod_{0 \leq y \leq q(|x|)} f(< x, y >)$$

Property 5 *If $f \in GapP$, $k \in FP$, and $k(x)$ is bounded by a polynomial in $|x|$, then the function below is in $GapP$.*

$$g(x) \equiv \binom{f(x)}{k(x)}$$

Property 6 *If $f, g \in GapP$ and $0 \leq g(x) \leq q(|x|)$ for some polynomial q, then the function below is in $GapP$.*

$$h(x) \equiv f(< x, g(x) >)$$

We could view existing complexity classes, especially the ones in probabilistic computation, in a "counting" fashion, and derive results using the closure properties above. In particular, one of the most important probabilistic complexity class, BPP, could be defined as:

Definition 5 *A language L is in BPP if there is a P predicate R and a polynomial p such that for all $x \in \Sigma^*$ and for $m = p(|x|)$,*

$$x \in L \Rightarrow \|\{y \in \Sigma^* : |y| = m \wedge R(x, y)\}\| \geq \frac{2}{3} \cdot 2^m,$$
$$x \notin L \Rightarrow \|\{y \in \Sigma^* : |y| = m \wedge R(x, y)\}\| \leq \frac{1}{3} \cdot 2^m$$

1.2.2 Toda's Theorem

One of the most celebrated result of counting complexity is Toda's Theorem. With this theorem, we could understand that either $\#P$ or $GapP$ is pretty big. And they might be big enough to swallow the entire polynomial hierarchy (PH).

Theorem 1 *Toda's Theorem: $PH = P^{\#P}$*

The $GapP$ version is a little different in which the result below is with respect a random oracle.

Property 7 *With respect to a random oracle, PH is low for GapP, i.e.,*

$$Pr_R[GapP^{PH^R} = GapP^R] = 1$$

Note the above results use the so-called "oracle" technique. An *oracle Turing machine* is a Turing machine with an additional "oracle" to query for the answer to a decisional problem. The query always return in a single step. Often a computational problem is very hard to find an exact solution or algorithm to it, so in times we manage to construct an useful oracle and with the help of this oracle we might be able to solve the problem. This kind of technique is called "relativization". It does not really solve the problem, because an oracle might be physically realizable, but it do help us in determining the relative difficulty between problems. The oracle we choose should not be too powerful. Generic oracle [7] is a good choice. Several computational limitations of quantum computation were shown in this method.

1.2.3 Counting Complexity on Quantum Computation

We here use a simplified model of quantum computation due to [2]. While simple, this model captures all of the power of quantum computation.

Consider the transition function of a Turing machine that maps current state and content s under the tape heads to a new state, new values to write under the tape heads and a direction to move the heads. A deterministic Turing machine's transition function has a single output. A probabilistic Turing machines' transition maps to a distribution on outputs with nonnegative probabilities that add up to one.

A quantum Turing machine's transition function maps to a *superposition* of the outputs where each output gets an *amplitude* which may be a complex value. We can assume these amplitudes take on of the values in $\{-1, -\frac{4}{5}, -\frac{3}{5}, 0, \frac{3}{5}, \frac{4}{5}, 1\}$, this is proved by [SY96]. The quantum Turing machines we consider here all run in polynomial time and thus have an exponential number of possible configurations. Suppose that before a transition each configuration C_i has a real amplitude α_i. Consider the L_2 norm of the amplitudes

$$\sqrt{\sum_i \alpha_i^2}$$

A quantum Turing machine is required to preserve this L_2 norm. This is equivalent to the transition matrix U of the configurations being unitary. For real U, U is unitary if the transpose of U is the inverse of U.

To compute the probability of acceptance consider an initial configuration amplitude vector $\vec{\alpha}$ where $\alpha_0 = 1$ for the initial configuration C_0 and $\alpha_i = 0$ for every other configuration. Let $\vec{\beta} = U^t \cdot \vec{\alpha}$ where t is the running time of the Turing machine. The probability of acceptance is β_i^2 where C_i is the accepting configuration.

We can now define several complexity classes of quantum computation.

Definition 6 *A language L is in BQP if there is a quantum Turing machine M such that for all $x \in \Sigma^*$,*

- *If $x \in L$, then $M(x)$ accepts with probability at least two-thirds.*
- *If $x \notin L$, then $M(x)$ accepts with probability at most one-third.*

The class EQP has the same definition as BQP except that we require zero error. The class NQP has the definition the same definition as BQP except that we require the accept probability be larger than zero iff the input is in the language.

In [12], several complexity theoretic results were given:

Theorem 2 *$BQP \subseteq AWPP$, while $AWPP$ is defined as those languages L such that for all polynomials q there is a GapP function f and a polynomial-time computable function g such that for all $x \in \Sigma^*$ and $m \geq |x|$, $0 < f(x, 1^m) < g(1^m)$ and*

- *If $x \in L$, then $f(x, 1^m) \geq (1 - 2^{-q(m)})g(1^m)$*
- *If $x \notin L$, then $f(x, 1^m) \leq 2^{-q(m)}g(1^m)$*

Theorem 3 *BQP is low for PP.*

After applying techniques of generic oracles, we have:

Theorem 4 *There exists a relativized world where $P = BQP$ and the polynomial-time hierarchy is infinite.*

In [10], the following results were given:

Theorem 5 *$NQP = coC_{=}P$.*

Theorem 6 *For any $f \in GapP$, there is a polynomial-time quantum Turing machine Q and a polynomial p such that for all x of length n,*

$$Pr[Q(x)\ accepts] = \frac{f(x)^2}{2^{p(x)}}$$

Since $coC_P \subseteq GapP$, by applying Toda's Theorem, [10] concludes that determining acceptance probability for a quantum computation is hard for the polynomial hierarchy. On the quantum computation models above, we only consider the transition matrix with rational numbers. However, in [22], this restriction was broadened to the set of complex numbers.

In summary, [8] [10] [12] [22] gave different results on the problem of $BPP = BQP$?. [8] summarizes these papers and gave two opposite conclusions. In one positive way, quantum computation is more powerful than classical computers under relativization. On the negative side, the complexity classes of quantum computation is a subset of $AWPP$ and PP, which were defined in probabilistic computation. The conclusions hint that the current mathmetical proof methods of theoretical computer science seem not being able to solve the problem directly. Therefore we might need some new non-relativized proof methods.

1.3 Decision Tree complexity

A decision tree of a Boolean function f is binary tree whose internal vertices are labeled by variables, leaves are labeled by 0 and 1, and edges are labeled also by 0 and 1 such that

- every pair of edges from an internal vertex to its two children are labeled by 0 and 1, respectively; and
- any variable appears at most once in any path from the root to a leaf.

The goal here is to compute a Boolean function $f : \{0,1\}^n \to \{0,1\}$ using queries to the input. In the most simple form a query asks for the value of the bit x_i and the answer is the value. The algorithm is adaptive, that is the, the kth query may depend on the answers of the $k-1$ previous queries. The algorithm can therefore be described by a binary tree, whence its name "decision tree". A Boolean function may have a number of different decision trees, each defining a different procedure to compute the function. We denote $D(f)$ the minimum depth of decision tree computing f. $D(f)$ is called the *decision tree complexity* of f. A sample decision tree is shown in the figure above. This measures correspondes to the minimum number of queries that an optimal determinsitic algorithm for f need to make on any input. We can extend this notion to more powerful modes of query algorithms. This results in *randomized* and even *quantum* decision trees.

In order to get a handle on the computational power of decision trees, other measures of the complexity of Boolean functions have been defined and studied. Some prime examples are *certificate complexity*, *sensitivity*, *block sensitivity*, the *degree of a representing polynomial*, and the *degree of an approximating polynomial.* In [4] and [1]

1.3.1 Decision Tree Models

A *deterministic decision tree* is defined above. $D(f)$, the decision tree complexity of f, is the depth of an optimal decision tree that computes f.

In *randomized decision tree*, the tree may contain internal nodes by a bias $p \in [0,1]$, and when the evaluation procedure reaches such a node, it will flip a coin with bias p and will go to the left child on outcome 'heads' and to the right child on 'tails'. A probability distribution is induced over the set of all leaves. We say a randomized decision tree *computes f with bounded-error* if its output equals $f(x)$ with probability at least 2/3, for all $x \in \{0,1\}^n$. $R_2(f)$ denotes the complexity of the optimal randomized decision tree that computes f with bounded error.

A *quantum decision tree* has the following form: we start with an m-qubit state $|\vec{0}\rangle$ where every bit is 0. Then we apply a unitary transformation U_0 to the state, then we apply a query O, then another unitary transformation U_1, etc. A T-query quantum decision tree thus corresponds to a big unitary transformation $A = U_T O U_{T-1} \dots O U_1 O U_0$. U_i are fixed unitary transformations, independent of the input x. The final state $A|\vec{0}\rangle$ depends on the input x only via the T applications of O. The output isobtained by measuring the final state and outputting the rightmost bit of the observed biasis state.

We say that a quantum decision tree *computes f exactly* if the output equals $f(x)$ with probability 1, for all $x \in \{0,1\}^n$. The tree *computes f with bounded-error* if the output equals $f(x)$ with probability at leasta 2/3, for all $x \in \{0,1\}^n$. $Q_E(f)$ denotes the number of queries of an optimal quantum decision tree that computes f exactly, $Q_2(f)$ is the number of queries of an optimal quantum decision tree that computes f with bounded-error. Note we just count the number of queries.

1.3.2 Complexity Measures

Certificate complexity measures how many of the n variables have to be given a value in order to fix the value of f.

Definition 7 *Let C be an assignment $C : S \to \{0,1\}$ of values to some subset S of the n variables. We say that C is* consistent *with $x \in \{0,1\}^n$ if $x_i = C(i)$ for all $i \in S$.*

For $b \in \{0,1\}$, a b-certificate for f is an assignment C such that $f(x) = b$ whenever x is consistent with C. The size of C is $|S|$, the cardinality of S.

The certificate complexity $C_x(f)$ *of f on x is the size of a smallest $f(x)$-certificate that is consistent with x. The* certificate complexity *of f is $C(f) = \max_x C_x(f)$. The* 1-certificate complexity *of f is $C^{(1)}(f) = \max_{x|f(x)=1} C_x(f)$, and similarly we define $C^{(0)}(f)$.*

Sensitivity and block sensitivity measure how sensitive the value of f is to changes in the input.

Definition 8 *The* sensitivity $s_x(f)$ *on x is the maximum number of variables x_i for which $f(x) \neq f(x^i)$. The* sentivity of f *is $s(f) = \max_x s_x(f)$.*

The block sensitivity $bs_x(f)$ *of f on x is the maximum number b such that there are disjoint sets $B_1, \dots, B_b$ for which $f(x) \neq f(x^{B_i})$. The* block sensitivity of f *is $bs(f) = \max_x bs_x(f)$. (If f is contant, we define $s(f) = bs(f) = 0$.)*

Each Boolean function, there are some ways to view them as polynomials in terms of their inputs. We're especially interested in understanding the degree of these polynomials.

Definition 9 *A polynomial* $p : \mathbb{R}^n \to \mathbb{R}$ represents f *if* $p(x) = f(x)$ *for all* $x \in \{0,1\}^n$*.*

The degree $\deg(f)$ *of* f *is the degree of the multilinear polynomial that represents* f*.*

Definition 10 *A polynomial* $p : \mathbb{R}^n \to \mathbb{R}$ approximates f *if* $|p(x) - f(x)| \leq 1/3$ *for all* $x \in \{0,1\}^n$*.*

The approximate degree $\widetilde{\deg}(f)$ *of* f *is the minimum degree among all multilinear polynomials that approximate* f*.*

1.3.3 Relationships for Classical and Quantum Complexity

In [1] (and the later survey [4]), the authors found that the above measures are all polynomially related. This implies that for every Boolean *total function*, exponential speedup doesn't exist in quantum computers computing these problems. To be more specific:

Theorem 7 *If* f *is a Boolean function, then* $D(f) \leq 4096 Q_2(f)^6$*.*

Theorem 8 *If* f *is a Boolean function, then* $D(f) \leq 32 Q_E(f)^4$*.*

Grover's search algorithm [14], which could be viewed as computing an OR function on the quantum computer, is only quadratically faster than a classical counterpart, with the theorem above, we could understand that there won't be much improvement on this algorithm.

1.4 Communication complexity

The need for communication arises whenever two or more computers, components, systems, or humans need to jointly perform a task that none of them can perform alone. The increasing importance of distributed computing, networking, VLSI and the use of computers in telecommunication have pointed out the significance of communication as a resource. In many devices, communication is significantly slower and costlier than local computation, and it is the real bottleneck in solving certain problems. What we will be studying here is how much communication is necessary to solve a given problem. The amount of communication needed is what we will call the communication complexity of the problem. Please refer to [15] and [17] for more complete survey.

1.4.1 Basic Model, Simultaneous Message Model

Let X, Y, Z be arbitrary finite sets and let $f : X \times Y \to Z$ be an arbitrary function. There are two players, Alice and Bob, who wish to evaluate $f(x, y)$, for some inputs $x \in X$ and $y \in Y$. Alice only knows x and Bob only knows y. Thus to evaluate the function, they will need to communicate with each other. The communication will be carried out according to some fixed protocol $\mathtt{P}$ (which depends only on f). The protocol consists of the players sending bits to each other until the value of f can be determined.

At each stage, the protocol $\mathtt{P}$ (for the function f) must determine whether the run terminates; if the run has terminated, the protocol mush specify the answer given by the protocol (that is, $f(x, y)$); and if the run has not terminated, the protocol must specify which player sends a bit of communication next. This information must depend solely on the bits communicated so far during this run of the protocol, because this is the only knowledge common to both Alice and Bob. In addition, if it is Alice's turn to speak, the protocol must specify what she sends; this depends on the communication so far as well as on x, the input visible to Alice. The similar things goes to Bob, too. We are only interested in the amount of communication between Alice and Bob, and we wish to ignore the question of the internal computations each of them makes. Thus, we allow Bob and Alice to have unlimited computation power. The cost of a protocl $\mathtt{P}$ on input (x, y) is the number of bits communicated by $\mathtt{P}$ on input (x, y). The cost of a protocol $\mathtt{P}$ is the *worst* case (maximal) cost of $\mathtt{P}$ over all inputs (x, y). The complexity of f is the minimum cost of a protocol that computes f.

On using communication complexity to measure the resources required for quantum computers to compute a particular function, we are more interested in a more restricted model, the *simultaneous message model*. In this model, two parties, Alice and Bob, receive inputs x and y respectively, and are not permitted to communicate with one another directly. Rather they each send a message to a third party, called the *referee*, who determines the output of the protocol based solely on the messages sent by Alice and Bob. The collective goal the three parties is to cause the protocol to output correct value of some function $f(x, y)$ while minimizing the amount of information that Alice and Bob send to the referee. Denote by $D^{\|}(f)$ the deterministic communication complexity of computing the function f using simultaneous protocols and $R^{\|}(f)$ the randomized communication complexity of computing the function f using simultaneous protocols.

1.4.2 Quantum Fingerprinting

As described with more detail in [18], we understand that quantum entanglement is a invaluable tool for usage in communication. As was demonstrated in [20], the game "Guess my number", which cannot be solved classically, could be easily solved in a quantum fashion withe the help of GHZ triplets $|000\rangle + |111\rangle$.

For the *equality* problem, the function is simply asking if $x = y$, and return 1 if so, 0 or else. The problem can be trivially solved if Alice sends x and Bob sends y to the referee, who can then simply compute $f(x, y)$. However, the cost of this protocol is high; if x and y are n-bit strings, then a total of $2n$ bits are communicated. If Alice and Bob instead send *fingerprints* of x and y, which may each be considerably shorter than x and y, the cost can be reduced significantly.

If Alice and Bob share a random $O(\log n)$-bit key then the fingerprints need only be of *constant* length. But the disadvantage of the above scheme is that it requires overhead in creating and maintaining a shared key. In [3], the authors gave the remarkable result that, under the setting of no shared key (or quantum entanglement) between Alice and Bob but the fingerprints can consist of quantum information, $O(\log n)$-qubit fingerprints are sufficient to solve the equality problem. This is an exponential improvement over the $\sqrt{n}$-bound for the comparable classical case. The method proposed by the authors was to set the 2^n fingerprints to quantum states whose pairwise inner-products are bounded below 1 in absolute value and to use a test that identifies identical fingerprints and distinguishes distinct fingerprints with good probability.

In [23], the author further showed that short quantum fingerprints can be used to solve the problem for a much larger class of functions. Let $R^{\|,pub}(f)$ denote the number of bits needed in the classical case, assuming in addition a common sequence of random bits known to all parties (the *public coin* model). Yao showed that, if $R^{\|,pub}(f) = O(1)$, then there exists a quantum protocol for f using only $O(\log n)$ bits.

2 Research Methodology and Agenda

Doing research of theoretical computer sciences involves mainly on reading research papers and giving mathematical proofs. This research propose a term of three years. We propose the following agenda:

1. In the first year, we will put focus on using methods from communication complexity to design protocols or find lower/upper bounds for

computational problems on quantum computers.

2. In the second year, we will put focus on counting complexity to view computational problems as Boolean functions and discuss their computational bounds.

3. In the third year, we will put focus on decision tree complexity and try to find relativized worlds (oracles) that could help us gaining more insights of quantum computation. And we would collect the research results of these three years and compile a complete report for lecture uses.

3 Results

The original proposal is a three-year project. However, we only get a one-year support. In this year, we spend most time in reading related papers. Despite the shortage of time, we still have some results about lower bounds for permutation arrays.

3.1 Lower Bound for Permutation Arrays

3.1.1 Introduction

In this note we study the construction of mapping from binary vectors of dimension n to permutations of $\{1, 2, \cdots, n+1\}$ that increase the minimum Hamming distance at least 2, for $n \geq 7$. A permutation array is a subset of these permutations that satisfies some distance constraints. With our construction, we improve the lower bounds on the size of permutation arrays. For the motivation for studying related mappings, we refer to the paper [16].

A systematic study of DPMs (distance-preserving mappings) can be found in [16]. Then, Chang [5] improved a result by Lee [?] and gave a construction of DIMs (distance-increasing mappings). In both of the papers [16, 5], a lower bound on the size of permutation arrays is given, i.e. $P(n, r) \geq A(n, r-1)$, where $P(n, r)$ denotes the maximal size among all permutation arrays of length n with minimum distance r, and $A(n, r)$ denotes the maximal size among all binary codes of length n and minimum distance r . Here, we prove that $P(n, r) \geq A(n-1, r-2)$. It is well-known that for all possible n and r, $A(n-1, r-2) \geq A(n, r-1)$. Furthermore $A(n-1, r-2)$ is much larger than $A(n, r-1)$ when n is even.

In this note, we give explicit constructions for $n = 7, 8, 9$ and 10. Then, for $n \geq 11$, we show how to construct from a mapping for $n-4$.

3.1.2 Notations

Let S_n denote the set of all permutations of $Z_n = \{1, 2, \cdots, n\}$ and the set Z_q^n denote the set of all q-ary vectors of length n. For a permutation $\pi = (\pi_1, \cdots, \pi_n) \in S_n$, let $\pi(i) = \pi_i$ and $\pi_{[i..j]}$ denote that sub-array $(\pi_i, \cdots, \pi_j)$ of π. For $i \in \{1, 2, \cdots, n\}$, $\pi^{-1}(i)$ denotes the position of i in π, i.e. if $\pi(j) = i$ then $\pi^{-1}(i) = j$. The Hamming distance $d_H(a, b)$ between two n-tuples $a = (a_1, a_2, \cdots, a_n)$ and $b = (b_1, b_2, \cdots, b_n)$ is the number of positions where they differ, i.e.

$$d_H(a, b) = |\{j | a_j \neq b_j\}|.$$

We now define a class of distance-increasing mapping from binary vectors to permutations. For $d \leq n + k$, let $f : Z_2^n \to S_{n+k}$ be a mapping such that for all $x, y \in Z_2^n$, $d_H(f(x), f(y)) \geq d_H(x, y) + d$, if $d_H(x, y) \leq (n + k) - d$; $d_H(f(x), f(y)) = n + k$, if $d_H(x, y) > (n + k) - d$.

Let $\mathcal{F}(d, n, k)$ denote the collection of all such function f. Clearly, the collection of DPMs equals to $\mathcal{F}(0, n, 0)$ and the collection of DIMs in [5] is $\mathcal{F}(1, n, 0)$.

3.1.3 Construction of $\mathcal{F}(2, n, 1)$

We give a systematic study on the construction of $\mathcal{F}(2, n, 1)$. Once we have a mapping in $\mathcal{F}(2, n, 1)$, the corresponding permutation arrays would be retrieved easily. First we will give the basic constructions: $g_7 \in \mathcal{F}(2, 7, 1)$, $g_8 \in \mathcal{F}(2, 8, 1)$, $g_9 \in \mathcal{F}(2, 9, 1)$, $g_{10} \in \mathcal{F}(2, 10, 1)$. Then , we can inductively construct $g_{n+4} \in \mathcal{F}(2, n + 4, 1)$ from a map $g_n \in \mathcal{F}(2, n, 1)$. Thus finally we have a family of mappings that increase the minimum distance at least 2, for $n \geq 7$.

We will first show the construction of g_7 as in the following and the other three constructions in the appendix. Consider two auxiliary mappings $A_7 \in \mathcal{F}(2, 3, 2)$ and $B_7 \in \mathcal{F}(2, 4, 2)$. We construct g_7 with these two mappings. Similarly, for each of g_8, g_9 and g_{10}, we will use two auxiliary mappings for the constructions. Note that in the image of A_7, 5 only appears in coordinate 1 or 2 . Similarly, in the image of B_7 the value 1 only appears in coordinate 1 or 2, and the value 2 only appears in coordinate 3 or 4. With this observation, we can construct a mapping $g_7 \in \mathcal{F}(2, 7, 1)$ by the following algorithm:

```
Algorithm $g_7$:
Input: $(x_1, x_2, \cdots, x_7) \in Z_2^7$
Output: $(\pi_1, \cdots, \pi_8) = g_7(x_1, \cdots, x_7)$
begin
0   $\rho = A_7(x_1, x_2, x_3); \tau = B_7(x_4, x_5, x_6, x_7)$;
1   $\tau_i = \tau_i + 2$ for $1 \le i \le 6$;
2   $\rho_{\rho^{-1}(5)} = \tau_6$;
3   $\tau_{\tau^{-1}(3)} = \rho_4$;
4   $\tau_{\tau^{-1}(4)} = \rho_5$;
5   $(\pi_1, \pi_2, \pi_3) = \rho_{[1..3]}$;
6   $(\pi_4, \pi_5, \pi_6, \pi_7, \pi_8) = \tau_{[1..5]}$;
7   if $x_1 = 1$ then swap $(\pi_3, \pi_8)$;
end
```

x	$A_7(x)$	x	$A_7(x)$
000	$(1,5,3,4,2)$	100	$(5,2,1,4,3)$
001	$(1,5,4,2,3)$	101	$(5,3,2,4,1)$
010	$(2,5,3,1,4)$	110	$(5,4,1,3,2)$
011	$(2,5,4,3,1)$	111	$(5,1,2,3,4)$

x	$B_7(x)$	x	$B_7(x)$
0000	$(1,3,2,4,5,6)$	1000	$(3,1,2,5,4,6)$
0001	$(1,3,2,5,6,4)$	1001	$(3,1,2,4,6,5)$
0010	$(1,3,5,2,4,6)$	1010	$(3,1,4,2,5,6)$
0011	$(1,3,4,2,6,5)$	1011	$(3,1,5,2,6,4)$
0100	$(1,4,2,6,5,3)$	1100	$(5,1,2,6,4,3)$
0101	$(1,5,2,6,3,4)$	1101	$(4,1,2,6,3,5)$
0110	$(1,5,6,2,4,3)$	1110	$(4,1,6,2,5,3)$
0111	$(1,4,6,2,3,5)$	1111	$(5,1,6,2,3,4)$

Similar to [5], given $g \in \mathcal{F}(2, n, 1)$, let $D_{n\times(n+1)}$ denote the distance expansion matrix where D_{ij} represents the number of all unordered pairs $\{x, y\}$, $x, y \in Z_2^n$ such that $d_H(x, y) = i$ and $d_H(g(x), g(y)) = j$. Our D is a little bit different from those in previous works. Our D is an $n \times (n + 1)$ matrix, instead of $n \times n$ matrix. Since the permutations in the range of g is one dimension larger than the domain of g. We show the distance expansion matrix D for g_7 as follows:

```
**Algorithm** $g_{n+4}$:
**Input:** $(x_1, \cdots, x_n, \cdots, x_{n+4}) \in Z_2^{n+4}$
**Output:** $(\pi_1, \cdots, \pi_{n+5}) = g_{n+4}(x_1, \cdots, x_{n+4})$
**begin**
**0** $\rho = g_n(x_1, \cdots, x_n); \tau = B_7(x_1, x_2, x_3, x_4)$;
**1** $\tau_i = \tau_i + n - 1$, **for** $1 \le i \le 6$;
**2** $\tau_{\tau^{-1}(n)} = \rho_n$;
**3** $\tau_{\tau^{-1}(n+1)} = \rho_{n+1}$;
**4** $(\pi_1, \cdots, \pi_{n-1}) = \rho_{[1..n-1]}$;
**5** $(\pi_n, \cdots, \pi_{n+5}) = \tau_{[1..6]}$;
**6** **if** $x_1 = 1$ **then swap** $(\pi_1, \pi_{n+4})$;
**7** **if** $x_2 = 1$ **then swap** $(\pi_2, \pi_{n+5})$;
**end**
```

0	0	320	64	32	32	0	0
	0	0	320	336	464	144	80
		0	0	288	624	992	336
			0	0	320	992	928
				0	0	384	960
					0	0	448
						0	64

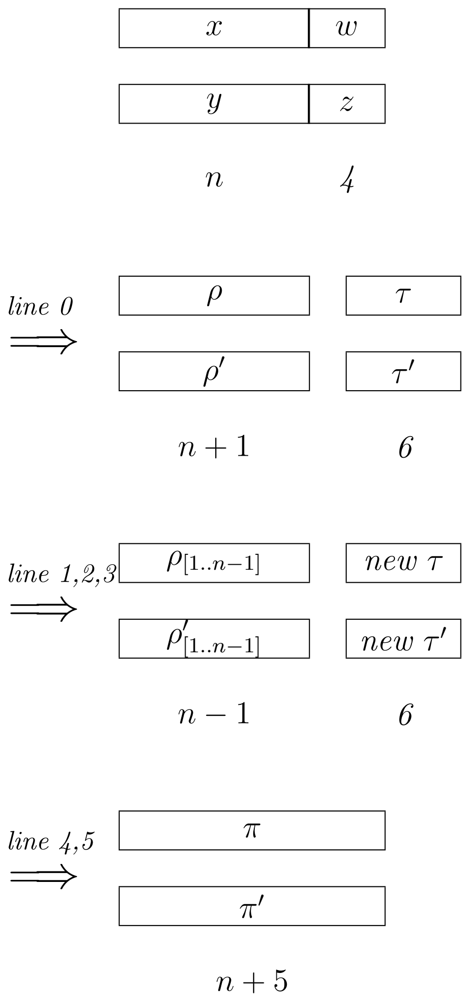
For g_8, g_9 and g_{10}, the constructions are similar and we reuse some of the auxiliary mappings. We show the details in the appendix.

Next we show how to construct a mapping $g_{n+4} \in \mathcal{F}(2, n+4, 1)$ inductively from a mapping $g_n \in \mathcal{F}(2, n, 1)$. To do that, we reapply the auxiliary mapping $B_7 : Z_2^4 \to S_6$. The construction is shown in Algorithm g_{n+4}. We prove the correctness as follows.

Theorem 1 *$g_{n+4} \in \mathcal{F}(2, n+4, 1)$, for $n \ge 7$.*

Proof 1 *First note that after line 1, $\rho_i \in \{1, \cdots, n+1\}$, for $i = 1$ to $n+1$ and $\tau_i \in \{n, \cdots, n+5\}$, for $i = 1$ to 6. But at line 2 and 3, the value n in τ is replaced by ρ_n and the value $n+1$ in τ by ρ_{n+1}. The other values in τ range from $n+2$ to $n+5$.*

Let (x, w) and $(y, z) \in Z_2^{n+4}$, where x, $y \in Z_2^n$, and w, $z \in Z_2^4$. Let $g_n(x) = \rho = (\rho_1, \cdots, \rho_{n+1})$, $g_n(y) = \rho' = (\rho'_1, \cdots, \rho'_{n+1})$, $B_7(w) = \tau = (\tau_1, \cdots, \tau_6)$, and $B_7(z) = \tau' = (\tau'_1, \cdots, \tau'_6)$. And $g_{n+4}(x, w) = \pi = (\pi_1, \cdots, \pi_{n+5})$, $g_{n+4}(y, z) = \pi' = (\pi'_1, \cdots, \pi'_{n+5})$. We illustrate the transforms of these two strings in the following diagram.



Let's first observe the change of the distance due to the swap step in line 6. If both $x_1 = 1$ and $y_1 = 1$ or both $x_1 = 0$ and $y_1 = 0$, the distance of these two coordinates remains the same. If exact one of x_1 and y_1 equals to 1, then the distance of these two coordinates won't decrease, since we know that the range of the values of π_1 and π'_1 is from $\{1, \cdots, n+1\}$ and the values of π_{n+4} and π'_{n+4} is from $\{n+2, \cdots, n+5\}$. Similarly, it holds for the swap step at line 7. Therefore after the swap steps the distance of these four coordinates does not decrease. While in some cases, the distance does increase.

Now we explain the effect of the operation at line 2. Since the values of ρ_n and ρ'_n are from $\{1, \cdots, n+1\}$, after the substitution, if $\tau^{-1}(n) = \tau'^{-1}(n)$ then ρ_n and ρ'_n are still in the same coordinate and the distance of this coordinate is preserved, else ρ_n and ρ'_n correspond to a value from $\{n+2, \cdots, n+5\}$ (note that $n+1$ is impossible, since value $n+1$ is in coordinate 3 or 4).

Thus after substituting operation at line 2, the distance won't decrease. Same argument holds for the operation at line 3. Therefore, after line 5, we have $d_H(\pi_{[n..n+5]}, \pi'_{[n..n+5]}) \geq d_H(\tau, \tau')$.

Next we consider the following cases:

- *Case [*$d_H(x, y) = 0$*]: We know that* $d_H(w, z) \neq 0$*, otherwise* (x, w) *and* (y, z) *are identical. Let* $d_H(w, z) = t \leq 4$*. Since* $B_7 \in \mathcal{F}(2, 4, 2)$*, we have* $d_H(\tau, \tau') \geq t+2$*. Therefore* $d(\pi, \pi')) \geq t+2 = d_H((x, w), (y, z)) + 2$.

- *Case [*$0 < d_H(x, y) = s < n$*]: It is clear that* $d_H(\rho, \rho') \geq s + 2$*. If* $0 < d_H(w, z) = t$*, then* $d_H(\tau, \tau') \geq t + 2$*. Thus, by the above mentioned observation, we have* $d_H(\pi, \pi') = d_H(\pi_{[1..n-1]}, \pi'_{[1..n-1]}) + d_H(\pi_{[n..n+5]}, \pi'_{[n..n+5]}) \geq s+(t+2) = d_H((x, w), (y, z))+2$*. For* $d_H(w, z) = 0$*, it is easy to see* $d_H(\pi, \pi') \geq s + 2$.

- *Case [*$d_H(x, y) = n$*]: In this case, it is clear that* $d_H(\rho, \rho') = n + 1$*. Let* $d_H(w, z) = t$*. Again by earlier observation, we know that* $d_H(\pi_{[1..n-1]}, \pi'_{[1..n-1]}) = d_H(\rho_{[1..n-1]}, \rho'_{[1..n-1]}) = n-1$ *and* $d_H(\pi_{[n..n+5]}, \pi'_{[n..n+5]}) \geq d_H(\tau, \tau') \geq t+2$ *(even when* $t = 0$*). Thus* $d_H(\pi, \pi') \geq n+t+1$*. We argue that this lower bound is indeed at least* $n+t+2$*, except when* $t = 4$*. There are two subcases on the value of* $d_H(w, z)$*, which is denoted as* t*.*

  1. *Subcase [*$t = 4$*]: Then* $d_H(\tau, \tau') = 6$*. It is easy to see* $d_H(\pi, \pi') = n + 5$.

  2. *Subcase [*$0 \leq t \leq 3$*]: If* $d_H(\tau, \tau') = 6$*, then* $d_H(\pi, \pi') = n + 5 \geq n + t + 2 = d_H((x, w), (y, z)) + 2$*. If* $d_H(\tau, \tau') \leq 5$*, there must be one coordinate* i *such that* $\tau_i = \tau'_i$*. Note that* $x_1 \neq y_1$ *and* $x_2 \neq y_2$*, since* $d_H(x, y) = n$*. If* $\tau_5 = \tau'_5$ *or* $\tau_6 = \tau'_6$*, then after the swap steps in line 6 and line 7,* $d_H(\pi_{[n..n+5]}, \pi'_{[n..n+5]}) \geq t + 3$*. So* $d_H(\pi, \pi') \geq n+t+2$*. If* $\tau_1 = \tau'_1$ *(or* $\tau_2 = \tau'_2$*), then after line 2 (line 3) the difference between* ρ_n *and* ρ'_n *(*ρ_{n+1} *and* ρ'_{n+1}*) is preserved respectively. Thus we have* $d_H(\pi_{[n..n+5]}, \pi'_{[n..n+5]}) \geq t + 3$*. Same argument holds for* $\tau_3 = \tau'_3$ *or* $\tau_4 = \tau'_4$*.*

This completes our proof on the correctness of construction.

3.1.4 Applications to Permutation Arrays

As shown in [16] and [5], we know that distance-increasing mappings are quite helpful for constructing permutation arrays. Let $P(n,r)$ denote the maximal size among all permutation codes of length n and minimum distance r, and $A(n,r)$ the maximal size among all binary codes of length n and minimum distance r.

Theorem 2 *For $n \geq 8$ and $3 \leq r \leq n$, $P(n,r) \geq A(n-1,r-2)$.*

Proof 2 *Let C be a binary code of length $n-1$ with minimum distance $r-2$. By the construction in Section 3, we have a mapping $g_{n-1} \in \mathcal{F}(2,n-1,1)$. From the definition, we know that $g_{n-1}(C)$ is a permutation array of length n with minimum distance r. Thus $P(n,r) \geq |C|$. Therefore $P(n,r) \geq A(n-1,r-2)$.*

Note that when r is odd, $A(n-1,r-2) = A(n,r-1)$. But when r is even, $A(n-1,r-2) > A(n,r-1)$. Thus, we improve the bound for permutation arrays when n is even.

Appendix

3.1.5 Construction of g_8

For g_8 we use A_8 and B_8 as the auxiliary mappings, where $A_8 : Z_2^4 \rightarrow S_6$ is defined as follows and B_8 is the same as B_7. The mapping $g_8 \in \mathcal{F}(2,8,1)$ is constructed by algorithm g_8.

x	$A_8(x)$	x	$A_8(x)$
0000	$(1,6,3,4,5,2)$	1000	$(6,2,1,4,5,3)$
0001	$(1,6,3,5,2,4)$	1001	$(6,2,3,1,5,4)$
0010	$(1,6,4,2,5,3)$	1010	$(6,4,5,1,2,3)$
0011	$(1,6,4,3,2,5)$	1011	$(6,2,4,3,1,5)$
0100	$(2,6,5,4,3,1)$	1100	$(6,3,2,4,5,1)$
0101	$(2,6,3,5,4,1)$	1101	$(6,3,2,5,1,4)$
0110	$(3,6,1,2,4,5)$	1110	$(6,4,1,2,3,5)$
0111	$(3,6,5,2,1,4)$	1111	$(6,1,2,3,4,5)$

```
Algorithm $g_8$:
Input: $(x_1, x_2, \cdots, x_8) \in Z_2^8$
Output: $(\pi_1, \cdots, \pi_9) = g_8(x_1, \cdots, x_8)$
begin
0  $\rho = A_8(x_1, \cdots, x_4), \tau = B_8(x_5, \cdots, x_8)$;
1  $\tau_i = \tau_i + 3$, for $1 \le i \le 6$;
2  $\rho_{\rho^{-1}(6)} = \tau_6$;
3  $\tau_{\tau^{-1}(4)} = \rho_5$;
4  $\tau_{\tau^{-1}(5)} = \rho_6$;
5  $(\pi_1, \cdots, \pi_4) = \rho_{[1..4]}$;
6  $(\pi_5, \cdots, \pi_9) = \tau_{[1..5]}$;
7  if $x_1 = 1$ then swap $(\pi_3, \pi_9)$;
end
```

3.1.6 Construction of g_9

For g_9 we use two auxiliary mappings A_9 and B_9, where A_9 is the same as A_8 and $B_9 : Z_2^5 \to S_7$ is defined as follows. We construct a mapping $g_9 \in \mathcal{F}(2, 9, 1)$. It follows from algorithm g_9. We show the algorithm as follows.

x	$B_9(x)$	x	$B_9(x)$
00000	$(1,3,2,4,5,6,7)$	10000	$(3,1,2,4,6,5,7)$
00001	$(1,3,2,4,6,7,5)$	10001	$(7,1,2,5,3,4,6)$
00010	$(1,3,2,5,7,6,4)$	10010	$(4,1,2,3,7,5,6)$
00011	$(1,3,2,5,4,7,6)$	10011	$(5,1,2,4,3,7,6)$
00100	$(1,3,4,2,6,5,7)$	10100	$(3,1,4,2,5,6,7)$
00101	$(1,3,7,2,5,4,6)$	10101	$(4,1,7,2,3,6,5)$
00110	$(1,3,6,2,7,5,4)$	10110	$(7,1,3,2,5,6,4)$
00111	$(1,4,3,2,5,7,6)$	10111	$(7,1,3,2,4,5,6)$
01000	$(1,5,2,3,6,4,7)$	11000	$(3,1,2,6,7,4,5)$
01001	$(1,4,2,7,6,3,5)$	11001	$(6,1,2,7,3,4,5)$
01010	$(1,4,2,6,7,5,3)$	11010	$(5,1,2,6,7,3,4)$
01011	$(1,5,2,6,3,7,4)$	11011	$(5,1,2,7,4,3,6)$
01100	$(1,6,4,2,7,3,5)$	11100	$(4,1,5,2,6,3,7)$
01101	$(1,6,5,2,3,4,7)$	11101	$(5,1,7,2,6,4,3)$
01110	$(1,5,6,2,4,3,7)$	11110	$(6,1,5,2,7,3,4)$
01111	$(1,6,5,2,4,7,3)$	11111	$(5,1,6,2,4,7,3)$

```
Algorithm $g_9$:
Input: $(x_1, x_2, \cdots, x_9) \in Z_2^9$
Output: $(\pi_1, \cdots, \pi_{10}) = g_9(x_1, \cdots, x_9)$
begin
0   $\rho = A_9(x_1, \cdots, x_4); \tau = B_9(x_5, \cdots, x_9)$;
1   $\tau_i = \tau_i + 3$, for $1 \leq i \leq 6$;
2   $\rho_{\rho^{-1}(6)} = \tau_7$;
3   $\tau_{\tau^{-1}(4)} = \rho_5$;
4   $\tau_{\tau^{-1}(5)} = \rho_6$;
5   $(\pi_1, \cdots, \pi_4) = \rho_{[1..4]}$;
6   $(\pi_5, \cdots, \pi_{10}) = \tau_{[1..6]}$;
7   if $x_1 = 1$ then swap $(\pi_3, \pi_9)$;
8   if $x_5 = 1$ then swap $(\pi_4, \pi_{10})$;
end
```

3.1.7 Construction for g_{10}

In this construction, we use A_{10} and B_{10} as the auxiliary mappings, where $A_{10} : Z_2^5 \to S_7$ is defined as follows and B_{10} is the same B_9. We construct a mapping $g_{10} \in \mathcal{F}(2, 10, 1)$ follows from algorithm g_{10}.

x	$A_{10}(x)$	x	$A_{10}(x)$
00000	$(1,7,3,4,5,6,2)$	10000	$(7,2,3,4,6,5,1)$
00001	$(1,7,3,4,6,2,5)$	10001	$(7,2,3,4,5,1,6)$
00010	$(1,7,3,5,2,6,4)$	10010	$(7,3,2,4,1,5,6)$
00011	$(1,7,3,5,4,2,6)$	10011	$(7,2,4,5,1,3,6)$
00100	$(1,7,4,3,6,5,2)$	10100	$(7,3,4,2,6,5,1)$
00101	$(1,7,4,6,5,2,3)$	10101	$(7,2,1,6,5,3,4)$
00110	$(1,7,4,2,3,5,6)$	10110	$(7,3,4,1,2,6,5)$
00111	$(1,7,4,6,2,3,5)$	10111	$(7,2,4,3,1,6,5)$
01000	$(2,7,3,1,6,5,4)$	11000	$(7,3,2,5,6,1,4)$
01001	$(2,7,3,6,5,4,1)$	11001	$(7,4,6,5,1,2,3)$
01010	$(2,7,5,4,3,6,1)$	11010	$(7,4,2,5,3,6,1)$
01011	$(2,7,3,5,1,4,6)$	11011	$(7,5,2,1,4,3,6)$
01100	$(2,7,5,3,6,1,4)$	11100	$(7,3,6,1,5,4,2)$
01101	$(2,7,6,3,4,1,5)$	11101	$(7,4,5,2,6,1,3)$
01110	$(2,7,5,1,4,6,3)$	11110	$(7,3,5,2,1,6,4)$
01111	$(2,7,4,6,3,1,5)$	11111	$(7,4,5,1,3,2,6)$

```
Algorithm $g_{10}$:
Input: $(x_1, x_2, \cdots, x_{10}) \in Z_2^{10}$
Output: $(\pi_1, \cdots, \pi_{11}) = g_{10}(x_1, \cdots, x_{10})$
begin
0   $\rho = A_{10}(x_1, \cdots, x_5); \tau = B_{10}(x_6, \cdots, x_{10})$;
1   $\tau_i = \tau_i + 4$, for $1 \leq i \leq 7$;
2   $\rho_{\rho^{-1}(7)} = \tau_7$;
3   $\tau_{\tau^{-1}(5)} = \rho_6$;
4   $\tau_{\tau^{-1}(6)} = \rho_7$;
5   $(\pi_1, \cdots, \pi_5) = \rho_{[1..5]}$;
6   $(\pi_6, \cdots, \pi_{11}) = \tau_{[1..6]}$;
7   if $x_1 = 1$ then swap $(\pi_3, \pi_{10})$;
8   if $x_6 = 1$ then swap $(\pi_4, \pi_{11})$;
end
```

3.1.8 Distance expansion matrices

g_8:

0	0	560	256	144	48	16	0	0
	0	0	752	624	872	864	424	48
		0	0	592	1032	2704	2088	752
			0	0	400	2080	4016	2464
				0	0	496	2928	3744
					0	0	720	2864
						0	0	1024
							0	128

g_9:

0	0	672	704	560	240	128	0	0	0
	0	0	1088	1428	1480	2122	1628	1094	376
		0	0	944	1402	4370	6478	5590	2720
			0	0	270	2522	8390	11998	9076
				0	0	134	4284	12118	15720
					0	0	474	5884	15146
						0	0	976	8240
							0	0	2304
								0	256

g_{10}:

0	0	1184	1472	1248	672	416	128	0	0	0
	0	0	2656	2924	3054	3854	4152	3454	2230	716
		0	0	2784	2976	8208	14300	15108	12388	5676
			0	0	1028	4764	18704	31298	32360	19366
				0	0	104	9554	31850	49270	38246
					0	0	774	15388	42838	48520
						0	0	2136	19760	39544
							0	0	3424	19616
								0	0	5120
									0	512

4 Bibliography

References

[1] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. In *IEEE Symposium on Foundations of Computer Science*, pages 352–361, 1998.

[2] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Computing*, 26(5):1411–1473, oct 1997.

[3] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf. Quantum fingerprinting. *Los Alamos Lab preprint library (http://xxx.lanl.gov)*, 2001(quant-ph/0102001), 2001.

[4] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288:21–43, 2002.

[5] J. Chang. Distance-increasing mappings from binary vectors from binary vectors to permuations. *IEEE Transactions on Information Theory*, 51(1).

[6] D.-Z. Du and K.-I. Ko. *Theory of Computational Complexity*. Wiley Interscience, 2000.

[7] S. Fenner, L. Fortnow, S. A. Kurtz, and L. Li. An oracle builder's toolkit. In {*SCT*}*: Annual Conference on Structure in Complexity Theory*, 1993.

[8] S. A. Fenner. *Counting Complexity and Quantum Computation*, chapter 8: Mathematics of Quantum Computation, pages 171–219. CRC Press, 2002.

[9] S. A. Fenner, L. Fortnow, and S. A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[10] S. A. Fenner, F. Green, S. Homer, and R. Pruim. Determining acceptance possibility for a quantum computation is hard for the polynomial hierarchy. *Electronic Colloquium on Computational Complexity (ECCC)*, 6(03), 1999.

[11] L. Fortnow. *Counting Complexity*, chapter 2: Complexity Theory Retrospective II, pages 81–107. Springer Verlag, 1997.

[12] L. Fortnow and J. D. Rogers. Complexity limitations on quantum computation. In *IEEE Conference on Computational Complexity*, pages 202–209, 1998.

[13] L. A. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion.* Springer Verlag, 2002.

[14] M. Hirvensalo. *Quantum Computing.* Springer Verlag, 2001.

[15] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[16] K. Lee. New distance-preserving maps of odd length. *IEEE Transactions on Information Theory*, 50(10).

[17] L. Lovasz. Communication complexity: A survey, 1989.

[18] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[19] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[20] A. M. Steane and W. van Dam. Physicists triumph at guess my number. *Physics Today*, pages 35–39, February 2000.

[21] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, pages 189–201, 2001.

[22] T. Yamakami and A. C.-C. Yao. $nqp = co - c_{=}p$. *Electronic Colloquium on Computational Complexity (ECCC)*, 5(073), 1998.

[23] A. C.-C. Yao. On the power of quantum fingerprinting. *Electronic Colloquium on Computational Complexity (ECCC)*, 9(02), 2002.