INDIAN STATISTICAL INSTITUTE

KOLKATA

Facets of Quantum Computation – Practice and Theory

Submitted in partial fulfillment of requirements

for the degree of

M. Tech. in Computer Science

by

Chandra Sekhar Mukherjee

Roll No: CS1908

Guide:

Subhamoy Maitra

Indian Statistical Institute, Kolkata

Batch: 2019-2021

Declaration

I declare that this thesis has been composed by myself and that this work has not been submitted for any other qualification. I confirm that the work submitted is my own, or part of jointly authored publications that has been included. My contribution and those of the other authors to this work have been explicitly indicated in the "Publications from the Dissertation" page. I further confirm that appropriate credit has been given where reference has been made to the work of others.

Chandra Sekhar Mukherjee,

M. Tech. CS, 2nd year,

Roll: CS1908,

Indian Statistical Institute.

Certificate

This is to certify that the dissertation titled "*Facets of Quantum Computation – Practice and Theory*" submitted by *Chandra Sekhar Mukherjee* to Indian Statistical Institute, Kolkata in partial fulfilment for the award of the degree of *Master of Technology in Computer Science* is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per regulation of this institute and, in my opinion, has reached the standard needed for submission.

Subhamoy Maitra,

Professor, Applied Statistics Unit,

Indian Statistical Institute.

Acknowledgment

I am very fortunate to have Professor Subhamoy Maitra as my advisor, who continually supported me and pushed me throughout my time in Indian Statistical Institute and always encouraged me to trot into uncharted territories. His support in academics and beyond, and availability in helping me with doubts and ideas has been central to completion of this work.

I am also thankful to my collaborators Dibyendu Roy and Vineet Gaurav for the many valuable discussions.

Most importantly I would like to acknowledge my friends and family, especially my mother, for believing in me through varying times.

Dedication

I dedicate this to my father, as a step towards the dream we shared.

Abstract

Quantum computation has seen rapid development both in theory as well as hardware and implementation in the past three decades. Against this backdrop we study both the implementational and theoretical aspects of quantum computation.

In Part I we approach the problem of efficient implementation of quantum algorithm. The exact requirement of CNOT and single qubit gates needed for optimal implementation in a given architecture is one of the central problems in this computational paradigm. Specifically, we study preparation of Dicke states ($|D_k^n\rangle$) using concise realizations of partially defined unitary transformations. We further optimize the state of the art deterministic Dicke state preparation circuit in terms of CNOT and single qubit gates. We explain theoretical ideas in reducing the gate counts and observe how these improvements are reflected in actual implementation of the circuits. To emphasize the advantages, we describe the circuit for preparing $|D_2^4\rangle$ on the "ibmqx2" machine of the IBM QX service. Our approach shows that the error induced due to noise in the system is lesser in comparison to the existing works. We conclude by describing the CNOT map of the generic $|D_k^n\rangle$ preparation circuit and analyze different ways of distributing the CNOT gates in the circuit and its effect on the induced error in the Noisy Intermediate Scale Quantum (NISQ) environment.

In Part II we concentrate on the query complexity model of evaluating functions for unknown inputs in a setup where the value of the input variables can only be accessed by making query to an oracle. First we study the separation between the deterministic query complexity ($D(f)$) and exact quantum query complexity $Q_E(f)$ of query friendly functions (QF). These are functions on n variables with minimum $D(f)$ value. We observe that there is a non separable QF function f ($D(f) = Q_E(f)$) for all values of n. Additionally, for some values of n all QF functions are non-

separable. Finally we construct separable QF functions for some other values of n via the parity method, which is the most well known exact quantum query algorithm. We further show that for rest of the values of n no separation can be obtained through parity decision tree model.

Next we look into obtaining separation between $D(f)$ and $Q_E(f)$ that is not reachable via any parity method ($D^{(2)}_{\oplus}(f)$). In this direction we study the algebraic normal form of Boolean (ANF) functions to obtain separation between $D(f)$ and $Q_E(f)$ beyond parity for classes of non-symmetric functions. We combine existing results on Fourier analysis along with novel exact quantum algorithmic techniques that we design to obtain a class of direct sum based functions of size $\Omega\left(\sqrt{2^{\sqrt{n}}}\right)$ functions for which we have $Q_E(f) < D^{(2)}_{\oplus}(f) < D(f)$ for which we design optimal quantum algorithm. In fact our algorithms are more efficient than any possible general parity method, a model in which one can obtain parity of any number of variables in one query. To the best of our knowledge, this is the first family of algorithms beyond generalized parity (and thus parity) for a large class of non-symmetric functions.

Finally we use our novel ANF based algorithmic techniques to obtain a $\lceil \frac{5n}{8} \rceil$-query exact quantum algorithm for a set of Maiorana-McFarland type bent functions of size $2^{2^{\frac{n}{4}}}$ functions. This is better than the best parity method we could obtain for these functions, which has a query complexity of $\lceil \frac{3n}{4} \rceil$.

Publications from the Dissertation

The following papers have been communicated as outcomes of this dissertation work:

1. C. S. Mukherjee, S. Maitra, V. Gaurav and D. Roy, "Preparing Dicke States on a Quantum Computer," in IEEE Transactions on Quantum Engineering, DOI: 10.1109/TQE/2020.3041479 (2020).

2. C. S. Mukherjee and S. Maitra, " Parity decision tree in classical–quantum separations for certain classes of Boolean functions," in Quantum Information Processing, DOI: 10.1007/s11128-021-03158-1 (2021).

3. C. S. Mukherjee and S. Maitra, " Exact Quantum Query Algorithms Outperforming Parity – Beyond The Symmetric functions, " ArXiv: 2008.06317

Contents

Layout

Quantum Computation is one of the most fundamental aspects of quantum mechanics. Coined by the renowned Richard Feynman, quantum computation has been one of the most widely studied and fast developing field in computer science. It began to gain attention with the famous Shor's algorithm for factoring and Simon's algorithm for period finding in the early 90's. Shor's algorithm showed how prime factorization can be done in the bounded error quantum model in time polynomial in the number of bits of the number in question, whereas till date we only have super polynomial solutions in the classical computation model. Since then the domain of quantum computation has seen tremendous progress both in theory and in realization towards practical quantum computers. In this direction we look into two of the main aspects of quantum computation, that is hardware optimization and analysis of the query complexity model.

In terms of realization of quantum computers, today we have noisy quantum computers of upto as many as 100 qubits (qubits are the classical counterpart of bits) and have cloud access to quantum computers with 5 to 20 qubits via different platforms such as IBM-Q and Rigetti computing. Despite this progress, the existing technologies on which quantum computers are built are noisy by nature, which makes any computation erroneous, even in the very small scale. In this direction we look into the superconducting model of quantum hardware. In this model any unitary applied on the system (gate) increases error induced into the system. Moreover, two-qubit CNOT gates, which has the highest error induction rate of the ones in the universal gate sets, cannot be applied to any pairs of qubits due to hardware limitations. Part I is dedicated towards showing how optimizing circuits in terms of both architecture demands and gate count improves the error in the outcome. Specifically, we study the "Dicke State Preparation" problem and in this direction further optimize the state of the art deterministic algorithm, and in doing so also relax its architectural constraints. We observe these improvements by implementing and comparing our

circuit in a publicly available quantum computer.

Although Shor's algorithm has an asymptotically better time complexity than all known classical algorithms, one can not yet rule out the existence of a classical algorithm of similar efficiency. Overall understanding whether quantum computation is inherently a more powerful form of computation has been a very challenging problem. In this regard certain restricted models of computation (also called the black-box model) have been used to show separation between classical and quantum computation in a complexity theoretic fashion. In fact Shor's algorithm and the famous Grover's search algorithm are all set up in this model. The query complexity model we discuss in Part II is one such model, although it differs from the model in which the aforementioned algorithms were setup. Here the goal is to evaluate a function given its description for any inputs. Our functions of interest are Boolean functions, one of the most centrally studied discrete functions. Here the restriction is that one can only access the inputs to the functions my making queries to an oracle, whose internal functioning is unknown to us. In the classical model given a function f on n variables $x_1, \ldots, x_n$, if one queries the oracle with i, they get back the value of x_i. Similarly in case of the quantum model one can make superposition of queries, and attempt to evaluate the function making lesser number of queries. We are interested in the difference in query complexity between the deterministic classical ($D(f)$)and exact quantum query models ($Q_E(f)$), the two error free computational models. We are interested in evaluating the separations in the two models in different circumstances, that shall allow to us to understand how structure of Boolean functions can be exploited in the quantum computational model, as well as possibly lead us to new results in quantum computation.

In Chapter 2, Part II we analyze the functions on n variables with least possible deterministic query complexity. We term them as query friendly functions. We observe that for certain values of n all query friendly functions have same $Q_E(f)$ and $D(f)$ values. For certain other values of n we obtain separation via the most well known exact quantum algorithmic technique, known as the parity decision tree. For the rest of the cases we show that no separation can be achieved via parity decision tree technique.

Next in Chapter 3 we analyze the exact quantum query complexity of functions through their algebraic normal form (ANF). ANFs are widely studied in the domain of cryptography but is not widely explored in the query complexity model. Using our analytical techniques we obtain

large classes of functions for which we are able to show separation between $D(f)$ and $Q_E(f)$ as well as design novel exact quantum query algorithms. We further show that the well known parity decision tree techniques would not work equally efficiently for these functions. In doing so we design a novel exact quantum algorithmic protocol that untangles a system under certain conditions more efficiently.

Finally we finish our work in Chapter 4 by studying the Maiorana-McFarland (MM) type bent functions, a class of size doubly exponential in n for any even n. Deterministic query complexity of all functions in this class is n. We first obtain a $\lceil \frac{3n}{4} \rceil$-query parity decision tree technique for any functions in this class by exploiting its ANF structure. Next, we obtain a more efficient $\lceil \frac{5n}{8} \rceil$-query algorithm for a large subclass $\left(2^{2^{\frac{n}{4}}}\right)$ of MM type bent functions using our algorithmic techniques of Chapter 3. We conclude each chapter with related future directions and open problems.

Finally, we conclude this document with an ending note.

Part I

Quantum Circuits and Architecture Dependence

Chapter 1

Optimizing Quantum Hardware – Dicke State Preparation

1.1 Introduction

Quantum Computers enable Quantum Algorithms that can perform operations with even super exponential speed-ups in time over the best known classical algorithms. Any quantum algorithm can be defined as a series of unitary transformations and can be implemented as a Quantum Circuit. A quantum circuit has a discrete set of gates such that their combinations can express any unitary transformation with any desired accuracy. Such a set of gates is called a universal set of gates. We know from the fundamental work by Barenco et.al [1] that single qubit gates and the controlled NOT (CNOT) gate form a universal set of gates. In this work, we call these gates elementary gates.

Quantum State Preparation is a topic within Quantum Computation that has garnered interest in the past two decades due to applications of special quantum states in several fields of Quantum Information Theory. A n-qubit quantum state $|\psi_n\rangle$ can be expressed as the superposition of 2^n orthonormal basis states. In this work we look at n qubit states as super position of the computational basis states $|x_1x_2\dots x_n\rangle, x_i \in \{0,1\},\ 1 \leq i \leq n$. The basis states in the expression of $|\psi_n\rangle$ with non zero amplitude are called the active basis states. Starting from the state $|0\rangle^{\otimes n}$ any arbitrary quantum state can be formed using $\mathcal{O}(2^n)$ elementary gates, although for many n qubit states preparation circuits with polynomial (in n) number of elementary gates is possible. The family of Dicke States $|D^n_k\rangle$ is one such example. $|D^n_k\rangle$ is the n-qubit state which is the equal superposition state of all $\binom{n}{k}$ basis states of weight k. For example $|D^3_1\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)$. Dicke states are an interesting family of states due to the fact that they have $\binom{n}{k}$ active basis states, which can be exponential in n when $k = \Theta(n)$ but need only polynomial number of elementary gates to prepare. Dicke states also have applications in the areas of Quantum Game Theory, Quantum Networking, among others. One can refer to [2] for getting a more in-depth view of these applications.

There has been several probabilistic and deterministic Dicke state algorithms designed in the last two decades [3, 9, 4]. In this document we focus on the algorithm described by Bärtschi et.al [2] which gives a deterministic algorithm that takes $\mathcal{O}(kn)$ CNOT gates and $\mathcal{O}(n)$ depth to prepare the state $|D^n_k\rangle$. To the best of our knowledge this circuit description has the best gate count among the deterministic algorithms. Here it is important to note that the paper by Cruz et.al [5] describes two algorithms for preparing the $|D^n_1\rangle$ states, also known as W_n states. Both the algorithms have better gate count than the description by Bärtschi et.al [2] and one of the

algorithms has logarithmic depth. However, their work is restricted to $|D_1^n\rangle$ and has no implication on the circuits for $|D_k^n\rangle$, $2 \leq k \leq n-2$. We further observe in Section 4.3 that the circuit obtained by us after the improvements for $|D_1^n\rangle$ is same as the linear W_n circuit described in [5].

Because of the noisy behavior of current generation Quantum Computers the exact number of elementary gates needed and the distribution of the gates over the corresponding circuit become crucial issues which need to be optimized in order to prepare a state with high fidelity. An example of a very recent work done in this area is [8] which reduces the gate count of AES implementation. In this regard we discuss the following important problems in the domain of Quantum Circuit Design.

A unitary transformation acting on n qubits can be expressed as a $2^n \times 2^n$ unitary matrix and can be decomposed into elementary gates in several ways. Therefore finding the decomposition that needs the least amount of elementary gates is a very fundamental problem, with [6], [10] being examples of work done in this area. There has also been work on obtaining lower bounds on approximately preparing states using quantum circuits [7] and also the work of [12] that gives some upper-bounds on number of CNOT gates needed to implement any arbitrary n qubit unitary matrix. However, the result by [12] is general and therefore the number of CNOT is exponential in the number of qubits, where as in case of Dicke state preparation circuit we have deterministic circuits with only polynomial in n CNOT gates.

It is crucial to minimize the number of gates while decomposing a unitary matrix as every gate induces some amount of error into the result. Especially reducing the number of CNOT gates is of importance due to the well known fact that it induces more error compared to single qubit gates.

Here we first describe a fundamental problem that decomposition of matrix using a universal set of gates poses. Let there be a unitary transformation that is to be performed on a system of n qubits. This task can be represented as a unitary matrix U_n that works on the Hilbert Space H_n of dimension 2^n. If we know the intended transformation for all the states of any orthonormal basis of H_n, that completely defines the unitary matrix U_n. Let us consider such a transformation for $n = 1$. If the transformation is defined for the two states in the computational basis $|0\rangle$ and $|1\rangle$ then the corresponding unitary matrix is completely defined. If the transformation is defined as $|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ then the corresponding matrix is the Hadamard

matrix, expressed as $\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$. However if the transformation is only defined for one state, $|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and not defined for $|1\rangle$ then there can be uncountably many unitary matrices that can perform the said transformation. Specifically, any matrix of the form $\begin{bmatrix} \frac{1}{\sqrt{2}} & \alpha \\ \frac{1}{\sqrt{2}} & -\alpha \end{bmatrix}$ can perform this task, where $\alpha \in \mathbb{C}$, $|\alpha|^2 = \frac{1}{2}$.

There exists many quantum algorithms where at a step a particular transformation on n qubits is defined only for a a subset of the states of a orthonormal basis. This creates the possibility of there being uncountably many unitary matrices capable of such a transformation. The algorithm described in [2] contains such transformations that are not completely defined for all basis states. Let such a transformation be defined as a partially defined unitary transformation on n qubits. There are possibly multiple unitary matrices that can perform this transformation. In that case it becomes an important problem to find out which candidate unitary matrix can be decomposed using the minimal number of elementary gates.

Furthermore, the number of elementary gates needed to implement a well defined Quantum Circuit also varies with the architecture of the actual Quantum Computer. The architectures of current generation Quantum Computers do not allow for CNOT gates to be implemented between any two arbitrary qubits. This CNOT constraint may further increase the total number of CNOT and single qubit gates needed to implement a Quantum Circuit on a specific Quantum Architecture. Therefore the total number of CNOT gates needed to implement a circuit on an architecture maybe more than the theoretical value if the logical CNOT map of the circuit is not a subgraph of the CNOT mapping of the architecture. There has been different optimization techniques developed for efficient implementation such as the work in [15] but ensuring any sort of optimality is in this scenario.

Against this backdrop, let us draw out the organization of the rest of the document.

1.1.1 Organization

In Section 4.2 we first define the concept of maximally partial unitary transformation. We then describe the the circuit in [2] for preparing Dicke States. We denote the circuit described in [2] for preparing $|D_k^n\rangle$ as $\mathcal{C}_{n,k}$.

We start Section 1.3 by showing that a transformation implemented in $\mathcal{C}_{n,k}$ is in fact a partially

defined construction. We then show that the unitary matrix used to represent the transformation is not optimal in terms of number of elementary gates needed to decompose it. We propose a different construction that indeed requires lesser number of elementary gates and we also argue its optimality w.r.t the Universal gate set.

In Section 4.3 we propose additional methods to reduce the gate count of the implementation shown in [2], centered around different partially defined unitary transformations used in the circuit. We remove the redundant gates in the circuit and analyze the different partially defined transformations implemented in the circuit to further reduce the gate counts of the circuit. We denote the improved circuit for preparing any Dicke State $|D_k^n\rangle$ as $\widehat{\mathcal{C}}_{n,k}$.

Next in Section 4.4 we discuss the architectural constraints posed by the current generation Quantum Computers that are available for public use through different cloud services. We discuss the restrictions in terms of implementing CNOT gates between two qubits in an architecture and how it increases the number of CNOT gates needed to implement a circuit in an architecture. In this regard we show that the improvements described by us in Section 4.3 not only reduces gate counts but also reduces architectural constraints.

We implement the circuits $\mathcal{C}_{4,2}$ and $\widehat{\mathcal{C}}_{4,2}$ on the IBM-QX machine "ibmqx2"[16] and calculate the deviation in each case from ideal measurement statistics using a simple error measure, and we also describe the results obtained by full tomography. Next we show how two circuits with the same number of CNOT gates and the same architectural restrictions can lead to different expected error due to different CNOT distribution across the qubits. We analyze this by proposing modifications in the circuit $\widehat{\mathcal{C}}_{4,2}$ possible because partial nature of certain transformations and how it reduces the number of CNOT gates functioning erroneously on expectation in a fairly generalized error model. We finish this section by drawing out the general CNOT map of $\widehat{\mathcal{C}}_{n,k}$, shown as the graph $G^{n,k}$ and observing that there in fact exists $n-k-1$ independent modifications each leading to a different CNOT distribution.

We conclude the document in Section 1.6 by noting down open problems in this area that we feel will improve our understanding both in the domains of partially defined transformations and architectural constraints.

1.2 Preliminaries

Let us first define some terminologies that we shall frequently use before moving onto some definitions and the preliminaries.

1.2.1 Notations

1. $|v_{(2)}\rangle$: If we look at a system with n qubits then all the 2^n orthogonal states in the computational basis can be expressed as $|b_1 b_2 \dots b_n\rangle\,,\ b_i \in \{0,1\}, 1 \leq i \leq n$.

   In that case for representing the state $|b_1 b_2 \dots b_n\rangle$ we treat it as a binary string and express it as $|v_{(2)}\rangle$ where $v = \sum_{i=1}^{n} b_i 2^{n-i}$.

2. $R_y(\theta)$: The R_y gate is a single qubit gate defined as follows. $R_y(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$.

3. X: This is a single qubit gate defined as $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

4. CU^i_j: While implementing a controlled unitary on a two qubit subsystem we use the following notations. Let there be a n-qubit system. CU^i_j represents a two qubit controlled unitary operation where the i-th qubit is the control qubit and the j-th qubit is the target qubit.

1.2.2 Maximally Partial Unitary Transformation

Let there be a unitary transformation that acts on n qubits. To perform this transformation we have to create a corresponding unitary matrix. If the transformation is defined for all 2^n states of some orthonormal basis then the unitary matrix is completely defined. On the other hand if the transformation is defined for a single state belonging to the computational basis, only a single column of the corresponding $2^n \times 2^n$ matrix is filled. The rest can be filled up conveniently, provided its unitary property is satisfied. In this regard we call a unitary transformation on n qubits to be maximally partial if it is defined for $2^n - 1$ states of some orthonormal basis. That implies only a column of the matrix is not defined. This tutorial is centered around the observation that corresponding to a maximally partial unitary transformation there can be multiple unitary

matrices and the number of elementary gates needed to implement each of these matrices may not be same.

Next we discuss the structure of Dicke states and the current state of the art Dicke state protocol [2], for which we show methods for efficient implementations in the later sections.

1.2.3 The Dicke State Preparation Circuit $\mathcal{C}_{n,k}$

The circuit $\mathcal{C}_{n,k}$ as described in [2] works on the n qubit system $|q_1 q_2 \dots q_n\rangle$. The circuit $\mathcal{C}_{n,k}$ is broken into $n-1$ blocks of the form SCS_y^x of which the first $n-k$ blocks are of the form $SCS_k^{n-t},\ n-t>k$ which is then followed by $k-1$ blocks of the form $SCS_{i-1}^i, k \geq i \geq 2$.

A block SCS_k^n consists of a two qubit transformation and $k-1$ three qubit transformations. The two qubit transformation works on the $n-1$ and n-th qubits and we denote it as μ_n. We describe the overall structure of the circuit again in Section 4.4. The basic intuition behind the construction is as follows. Starting from the computational basis state $\otimes_{i=1}^n |0\rangle$, the state preparation circuit of $|D_k^n\rangle$ can be designed as the combination of SCS_k^n acting on the last $k+1$-qubits and the the state preparation circuit of $|D_k^{n-1}\rangle$ acting on the first $n-1$ qubits. One can refer to [2] to get a more in-depth view of this construction.

The three qubit transformations are of the form $\mathcal{M}_n^l, n-1 \leq i \leq n-k+1$ where $\mathcal{M}_l^n$ works on the qubits $l-1, l$ and n. This construction is interesting in how the transformations μ and $\mathcal{M}$ are partially defined which raises different implementation choices, with possibly different number of gates needed for elemental decomposition. We now describe these two transformations for reference. We denote by $|ab\rangle_x$ the qubits in the $x-1$ and x-th position in a system.

$$
\begin{aligned}
\mu_n: \quad & |00\rangle_n \rightarrow |00\rangle_n \\
& |11\rangle_n \rightarrow |11\rangle_n \\
& |01\rangle_n \rightarrow \sqrt{\frac{1}{n}}\,|01\rangle_n + \sqrt{\frac{n-1}{n}}\,|10\rangle_n
\end{aligned}
$$

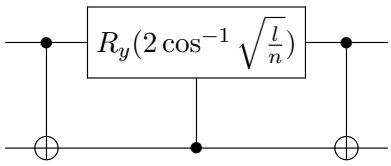

Figure 1.1: Implementation of μ_n

$$
\begin{aligned}
\mathcal{M}_n^l: \quad & |00\rangle_l |0\rangle_n \to |00\rangle_l |0\rangle_n \\
& |01\rangle_l |0\rangle_n \to |01\rangle_l |0\rangle_n \\
& |00\rangle_l |1\rangle_n \to |00\rangle_l |1\rangle_n \\
& |11\rangle_l |1\rangle_n \to |11\rangle_l |1\rangle_n \\
& |01\rangle_l |1\rangle_n \to \sqrt{\frac{n-l+1}{n}} |01\rangle_l |1\rangle_n \\
& \qquad\qquad + \sqrt{\frac{l-1}{n}} |11\rangle_l |0\rangle_n
\end{aligned}
$$

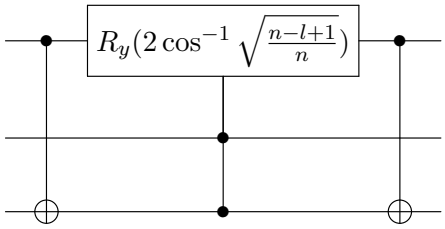

Figure 1.2: Implementation of $\mathcal{M}_n^l$

The implementations of these transformations in [2] is shown in Figure 1.1 and 1.2 respectively. The first transformation, μ_n is in fact a maximally partial unitary transform. Because of the partially defined nature of the transformation the CR_y and CCR_y gates are also not fed all possible inputs. Instead the input to the CR_y gates is only from the subspace spanned by the computational basis states $|00\rangle, |10\rangle$ and $|01\rangle$. Similarly the input to the CCR_y gate is only from the subspace spanned by the states $|000\rangle, |010\rangle, |001\rangle, |011\rangle,$ and $|110\rangle$.

Next in Section 1.3 we look how partially defined transformations can be implemented more efficiently, and argue the optimality of this improvement with respect to this particular building block. Then in Section 4.3 we study how the gate count of the circuit $\mathcal{C}_{n,k}$ can be reduced by removing redundancies and analyzing how the μ and $\mathcal{M}$ transformations act only on a subset of

the defined computational basis states in specific cases.

1.3 Example of Optimality for a Maximally Partial Unitary Transformation

We have already described the two partially defined unitary transformations used in the circuit $\mathcal{C}_{n,k}$. The implementation of the first transformation, μ_n is done using a controlled CR_y gate and two CNOT gates in [2]. This CR_y gate only acts on the states $|00\rangle, |10\rangle, |01\rangle$ and their superpositions and the transformation never acts on the $|11\rangle$ state. If we take $\theta = 2\cos^{-1}\left(\sqrt{\frac{1}{n}}\right)$ and denote the transformation implemented by the $CR_y(\theta)$ gate on the defined basis states as $T_1(\theta)$, then it can be expressed as follows:

$$\begin{aligned} T_1(\theta): \quad & |00\rangle \rightarrow |00\rangle \\ & |10\rangle \rightarrow |10\rangle \\ & |01\rangle \rightarrow \left(\cos(\frac{\theta}{2})\,|0\rangle + \sin(\frac{\theta}{2})\,|1\rangle\right)|1\rangle \end{aligned} \tag{1.1}$$

This is in fact a maximally defined partial unitary transformation. While the gate $CR_Y(\theta)$ can perform this transformation, it needs at least 4 elementary gates to implement. We first show this necessary requirement using an important result from [6, Theorem B], which we note down for reference.

Theorem 1. *[6]*

1. *For a controlled gate CU if $tr(UX) = 0,\ tr(U) \neq 0,\ \det U = 1,\ U \neq \pm I$ then the minimal number of elementary gates needed to implement CU is 4.*

2. *For a controlled gate CU if $tr(U) = 0,\ \det U = -1,\ U \neq \pm X$ then the minimal number of elementary gates needed to implement CU is 3.*

3. *For a controlled gate CU the minimal number of number of elementary gates needed to implement CU is less than three 3 iff $U \in \{e^{i\phi}I, e^{i\phi}X, e^{i\phi}Z\},\ 0 \leq \phi \leq 2\pi$.*

Our lemma follows immediately.

Lemma 1. *It takes minimum 4 elementary gates to implement the $CR_y(\theta)$ gate.*

Proof. We calculate the values of $\det R_y(\theta)$ and $tr(R_y(\theta)X)$ to confirm the minimal number of gates needed to decompose $CR_y(\theta)$.

$$\det R_y(\theta) = \sin^2(\frac{\theta}{2}) + \cos^2(\frac{\theta}{2}) = 1$$
$$R_y(\theta)X = \begin{bmatrix} -\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \\ \cos(\frac{\theta}{2}) & \sin(\frac{\theta}{2}) \end{bmatrix} \implies tr(R_y(\theta)X) = 0$$

The result (1) of Theorem 1 concludes the proof. □

However the transformation $T_1(\theta)$ can in fact be implemented using three elementary gates as follows.

$$T_1(\theta) \equiv \Big(R_y(\frac{-\alpha}{2}) \otimes I_2\Big)\mathsf{CNOT}_1^2\Big(R_y(\frac{\alpha}{2}) \otimes I_2\Big),\ \frac{\alpha}{2} = \frac{\pi}{2} - \frac{\theta}{2}$$

This decomposition has also been used by Cruz et.al [5] in the W_n (D_1^n) state preparation algorithm. However, the corresponding transformation there is defined only for the states $|00\rangle$ and $|01\rangle$ and no insight into the optimality of the implementation is given.

We first derive the underlying 4×4 unitary matrix $U^0(\alpha)$ that describes this three gate transformation. Next we show that $U^0(\alpha)$ needs at least three gates to be implemented by verifying the conditions of result (2) of Theorem 1. We end this section by showing that the transformation $T_1(\theta)$ needs at least three elementary gates (including one CNOT) to be implemented, proving the optimality of the $U^0(\alpha)$ implementation.

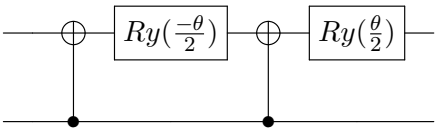

Figure 1.3: Implementation of $CR_y(\theta)$

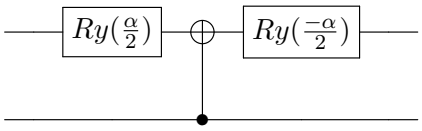

Figure 1.4: Implementation of $U^o(\alpha)$

Theorem 2. *The gate $U^0(\alpha)$ performs the partially defined unitary transformation $T_1(\theta)$ where $\alpha = \pi - \theta$ and needs minimum three elementary gates to be implemented.*

Proof. We first study the transformation carried out by U^0 in the subspace of T_1.

$$|00\rangle$$
$$\xrightarrow{R_y(\frac{\alpha}{2}) \text{ on } q1} \Big(\cos(\frac{\alpha}{4})\,|0\rangle + \sin(\frac{\alpha}{4})\,|1\rangle\Big)\,|0\rangle$$

$$\xrightarrow{\mathsf{CNOT}_1^2} \Big(\cos(\frac{\alpha}{4}) \left|0\right\rangle + \sin(\frac{\alpha}{4}) \left|1\right\rangle \Big) \left|0\right\rangle$$
$$\xrightarrow{R_y(-\frac{\alpha}{2}) \text{ on } q1} \Big(\cos(\frac{\alpha}{4}) \big(\cos(\frac{\alpha}{4}) \left|0\right\rangle - \sin(\frac{\alpha}{4})) \left|1\right\rangle \big) +$$
$$\sin(\frac{\alpha}{4})) \big(\sin(\frac{\alpha}{4})) \left|0\right\rangle + \cos(\frac{\alpha}{4})) \left|1\right\rangle \big) \Big) \left|0\right\rangle$$
$$= \big(\cos^2(\frac{\alpha}{4}) + \sin^2(\frac{\alpha}{4}) \big) \left|00\right\rangle = \left|00\right\rangle$$

$\left|10\right\rangle$

$$\xrightarrow{R_y(\frac{\alpha}{2}) \text{ on } q1} \Big(-\sin(\frac{\alpha}{4}) \left|0\right\rangle + \cos(\frac{\alpha}{4}) \left|1\right\rangle \Big) \left|0\right\rangle$$
$$\xrightarrow{\mathsf{CNOT}_1^2} \Big(-\sin(\frac{\alpha}{4}) \left|0\right\rangle + \cos(\frac{\alpha}{4}) \left|1\right\rangle \Big) \left|0\right\rangle$$
$$\xrightarrow{R_y(-\frac{\alpha}{2}) \text{ on } q1} \Big(-\sin(\frac{\alpha}{4}) \big(\cos(\frac{\alpha}{4}) \left|0\right\rangle - \sin(\frac{\alpha}{4})) \left|1\right\rangle \big) +$$
$$\cos(\frac{\alpha}{4})) \big(\sin(\frac{\alpha}{4})) \left|0\right\rangle + \cos(\frac{\alpha}{4})) \left|1\right\rangle \big) \Big) \left|0\right\rangle$$
$$= \big(\cos^2(\frac{\alpha}{4}) + \sin^2(\frac{\alpha}{4}) \big) \left|10\right\rangle = \left|10\right\rangle$$

$\left|01\right\rangle$

$$\xrightarrow{R_y(\frac{\alpha}{2}) \text{ on } q1} \Big(\cos(\frac{\alpha}{4}) \left|0\right\rangle + \sin(\frac{\alpha}{4}) \left|1\right\rangle \Big) \left|1\right\rangle$$
$$\xrightarrow{\mathsf{CNOT}_1^2} \Big(\sin(\frac{\alpha}{4}) \left|0\right\rangle + \cos(\frac{\alpha}{4}) \left|1\right\rangle \Big) \left|1\right\rangle$$
$$\xrightarrow{R_y(-\frac{\alpha}{2}) \text{ on } q1} \Big(\sin(\frac{\alpha}{4}) \big(\cos(\frac{\alpha}{4}) \left|0\right\rangle - \sin(\frac{\alpha}{4})) \left|1\right\rangle \big) +$$
$$\cos(\frac{\alpha}{4})) \big(\sin(\frac{\alpha}{4})) \left|0\right\rangle + \cos(\frac{\alpha}{4})) \left|1\right\rangle \big) \Big) \left|1\right\rangle$$
$$= \big(2\cos(\frac{\alpha}{4}) \sin(\frac{\alpha}{4}) \left|0\right\rangle + (\cos^2(\frac{\alpha}{4}) - \sin^2(\frac{\alpha}{4}) \left|1\right\rangle \big) \left|1\right\rangle$$
$$= \Big(\sin(\frac{\alpha}{2}) \left|0\right\rangle + \cos(\frac{\alpha}{2}) \left|1\right\rangle \Big) \left|1\right\rangle$$

Setting $\alpha = \pi - \theta$ gives us the same transformation as defined by $T_1(\theta)$.

Now we completely define the gate U^0 by studying the transformation acted on the state $\left|11\right\rangle$.

$\left|11\right\rangle$

$$\xrightarrow{R_y(\frac{\alpha}{2}) \text{ on } q1} \Big(-\sin(\frac{\alpha}{4}) \left|0\right\rangle + \cos(\frac{\alpha}{4}) \left|1\right\rangle \Big) \left|1\right\rangle$$
$$\xrightarrow{\mathsf{CNOT}_1^2} \Big(\cos(\frac{\alpha}{4}) \left|0\right\rangle - \sin(\frac{\alpha}{4}) \left|1\right\rangle \Big) \left|1\right\rangle$$
$$\xrightarrow{R_y(-\frac{\alpha}{2}) \text{ on } q1} \Big(\cos(\frac{\alpha}{4}) \big(\cos(\frac{\alpha}{4}) \left|0\right\rangle - \sin(\frac{\alpha}{4})) \left|1\right\rangle \big) -$$

$$
\begin{aligned}
&\sin(\frac{\alpha}{4}))\big(\sin(\frac{\alpha}{4}))\,|0\rangle + \cos(\frac{\alpha}{4}))\,|1\rangle\,)\Big)\,|1\rangle \\
&= \big((\cos^2(\frac{\alpha}{4}) - \sin^2(\frac{\alpha}{4})\,|0\rangle - 2\cos(\frac{\alpha}{4})\sin(\frac{\alpha}{4})\,|0\rangle\,)\,|1\rangle \\
&= \Big(\cos(\frac{\alpha}{2})\,|0\rangle - \sin(\frac{\alpha}{2})\,|1\rangle\,\Big)\,|1\rangle
\end{aligned}
$$

So the overall transformation provided by $U^0(\alpha)$ is:

$$
\begin{aligned}
&|00\rangle \to |00\rangle \\
&|10\rangle \to |10\rangle \\
&|01\rangle \to \Big(\sin(\frac{\alpha}{2})\,|0\rangle + \cos(\frac{\alpha}{2})\,|1\rangle\,\Big)\,|1\rangle \\
&|11\rangle \to \Big(\cos(\frac{\alpha}{2})\,|0\rangle - \sin(\frac{\alpha}{2})\,|1\rangle\,\Big)\,|1\rangle
\end{aligned}
$$

Therefore the gate $U^0(\alpha)$ is a two qubit gate which can be expressed as a controlled gate $CU(\alpha)$ gate where $U(\alpha) = \begin{bmatrix} \sin(\frac{\alpha}{2}) & \cos(\frac{\alpha}{2}) \\ \cos(\frac{\alpha}{2}) & -\sin(\frac{\alpha}{2}) \end{bmatrix}$. Now $trU(\alpha) = 0$ and $\det U(\alpha) = -1$ for all α. Therefore we can conclude from the result (2) in Theorem 1 that this gate requires at least three gates to be implemented. □

We finally show the optimality of this implementation for implementing the two qubit transformation $T_1(\theta)$.

Lemma 2. *The transformation $T_1(\theta)$ needs at least one CNOT and two single qubit gates to be implemented for $0 < \theta < \pi$.*

Proof. The transformation $T_1(\theta)$ is only defined for the basis states $|00\rangle, |01\rangle$ and $|10\rangle$. Any matrix $M(\theta)$ that can carry out the transformation is of the form $\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & \cos(\frac{\theta}{2}) & 0 & b \\ 0 & 0 & 1 & c \\ 0 & \sin(\frac{\theta}{2}) & 0 & d \end{bmatrix}$ where a, b, c, d are complex unknowns. However since $M(\theta)$ is unitary we have $M(\theta)M^\dagger(\theta) = I$. Therefore $1 + aa^* = 1 \implies a = 0$ and $1 + cc^* = 1 \implies c = 0$. That is the matrix M_θ is a controlled unitary $CM_1(\theta)$ and $M_1(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & b \\ \sin(\frac{\theta}{2}) & d \end{bmatrix}$.

Now for $0 < \theta < \pi$ both $\cos(\frac{\theta}{2})$ and $\sin(\frac{\theta}{2})$ are non zero. This implies that the matrix cannot fulfill the necessary conditions defined in result (3) of Theorem 1 and therefore cannot be expressed with less than three elementary gates. □

The code snippets in Listing 1.1 and Listing 1.2 show the two implementations for $T_1(\theta)$ where $\theta = 2\cos^{-1}(\frac{1}{\sqrt{3}})$ in the QASM language.

```
cx q[2],q[1];
ry(-pi/3.2885) q[1];
cx q[2],q[1];
ry(pi/3.2885) q[1];
```

Listing 1.1: $T_1(\theta)$ implemented as CR_y

```
ry ((pi/2)-pi/3.2885) q[1];
cx q[2],q[1];
ry (-((pi/2)-pi/3.2885)) q[1];
```

Listing 1.2: Modified implementation as $U^0(\theta)$

Now we use these observations to show how efficient implementation of partially defined unitary transformations can lead to better implementation of the Dicke state preparation circuit $\mathcal{C}_{n,k}$.

1.4 Improving the implementation of $|D_k^n\rangle$

We first count the number of CNOT and single qubit gates in $\mathcal{C}_{n,k}$ by reviewing the circuit. The circuit is composed of $n-1$ blocks of gates called SCS. There are $n-k-1$ blocks of the form $SCS_k^t,\ k < t \leq n$ and $k-1$ blocks of the form $SCS_i^{i+1},\ 1 \leq i \leq k-1$.

Each block SCS_k^t consists of one two qubit transformation μ_t which is implemented on the qubits $t-1$ and t and $k-1$ three qubit transformations of the type $\mathcal{M}_t^l,\ t-1 \leq l \leq t-k-2$. Here μ_t is implemented on the $t-1$ and t-th qubit and $\mathcal{M}_t^l$ is implemented on the $l-1$, l and t-th qubit, as described in Section 4.2. Each transformation of type μ is decomposed into two CNOT and a $T_1(\theta)$ transformation which is implemented as a CR_y gate by adjusting the value of θ. One can refer to Section 4.2 for the functioning of the CR_y gate. We have shown in Lemma 1 that a CR_y transformation needs minimum 4 gates to implement. In fact it needs at least two CNOT gates. Therefore each μ transformation needs four CNOT and two single qubit gate. The number

of transformations of type $\mathcal{M}_n^l$ is

$$\begin{aligned}
&(n-k)(k-1) + \sum_{i=1}^{k-2} i \\
&= nk - n + k - k^2 + \frac{(k-1)(k-2)}{2} \\
&= nk - \frac{k(k+1)}{2} - n + 1.
\end{aligned}$$

Each $\mathcal{M}_n^l$ transformation is shown to require six CNOT and four single qubit gates. However one CNOT gate of for each $\mathcal{M}_n^l$ transformation can be canceled by rearranging the first two CNOT gates of the next transformation.

The total number of CNOT gates and single qubit gates used to prepare the state $|D_k^n\rangle$ is shown in Table 1.1.

CNOT gates	$5(nk - \frac{k(k+1)}{2} - n + 1) + 4(n-1)$
single qubit gates	$4(nk - \frac{k(k+1)}{2} - n + 1) + 2(n-1)$

Table 1.1: Gates needed to prepare $|D_k^n\rangle$ as in [2]

Figure 1.5 shows the circuit $\mathcal{C}_{6,3}$ in terms of CNOT, CR_y and CCR_y gates.

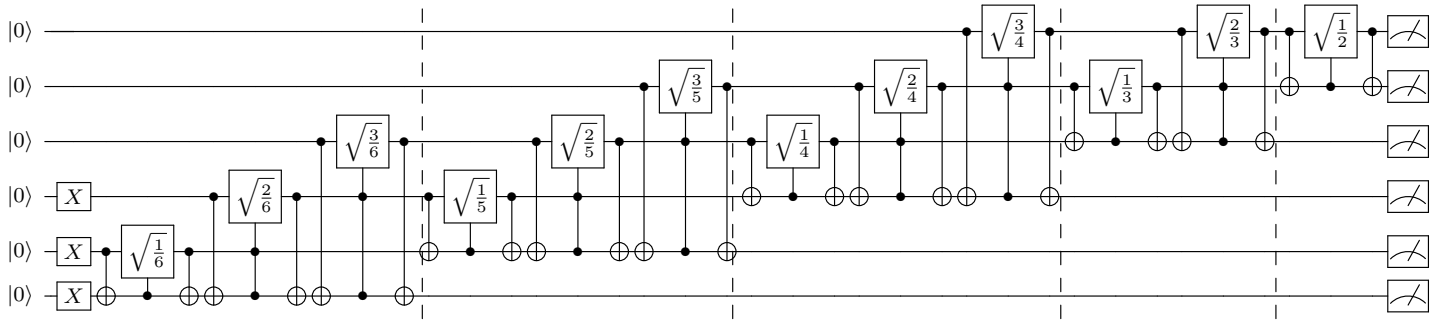

Figure 1.5: Description of the circuit $\mathcal{C}_{6,3}$

Having discussed the gate count of the original circuit description, let us now explore methods through which this gate count can be reduced. Here it should be noted that both CNOT and single gate count of the original circuit description is $\mathcal{O}(nk)$ for implementing $|D_k^n\rangle$. If one applies the implementation that we shall suggest, the gate counts will still remain $\mathcal{O}(nk)$. However, the reduction in the number of gates has direct consequence in terms of the error induced in the circuit,

even for as low value of n as 4.

Here, note that Figure 1.5 contains of the blocks SCS_3^6, SCS_3^5, SCS_3^4 , SCS_2^3 and SCS_1^2 and we have separated these blocks with dotted line. The same partitioning is continued in the improved circuit Figure 1.6 to help understand the block-wise reduction in gate count.

1.4.1 Replacing CR_y with CU

If one uses the CU gate shown in Section 1.3 to implement the transformation T_1 corresponding to each μ transformation, then that requires one CNOT and two single qubit gates to be implemented. Therefore each μ transformation needs three CNOT and two single qubit gates. Since there are $n-1$ μ transformations this reduces the number of CNOT by $n-1$ for any $|D_k^n\rangle$. In this regard one can also refer to the work in [13] which shows that any two qubit unitary can be implemented with 3 CNOT gates, but our approach leaves the CNOT gates available for forward cancellation with the next $\mathcal{M}$ block.

One of the more easily identifiable redundancy in the circuit is of gates that do not actually have any impact on the state. That is there are controlled gates in the circuit that always have the qubit of the control in state $|0\rangle$ in all the superposition states that is fed to it. These kinds of redundancies occur as a result of direct implementation of a theoretical argument. Next we observe that some of the μ_n and $\mathcal{M}_n^l$ transformations act as identity transformation, which we count as a function of k for any $|D_k^n\rangle$.

1.4.2 The μ and $\mathcal{M}$ transformations that act like Identity

Let there be a n qubit system in some state $|\phi\rangle$. This state can be uniquely represented as a superposition of all 2^n (computational) basis state. The amplitude of a particular basis state may or may not be zero depending on the description of $|\phi\rangle$. Let us call a basis state with non zero amplitude an active basis state. The affect of a unitary transformation T on this state can be completely described by observing how it transforms the active basis states of $|\phi\rangle$. If the k-th qubit is in the zero (one) state in all the active basis states and the transformation T doesn't act on the k-th qubit in non trivial way on any of those basis states then the k-th qubit in all the active basis states of $T|\phi\rangle$ will also be in the zero(one) state. Here we observe that in the Dicke state preparation circuit we have many CNOT gates where the control is always in the $|0\rangle$ state and all

such gates can be removed. In this regard we prove the following theorem using induction.

Theorem 3. *If the n qubit system is expressed as superposition of computational basis states after the block SCS_k^{n-t} has acted then it can be expressed as*

$$\sum_{a=0}^{2^{t+1}-1} \sum_{b=0}^{2^{t+1}-1} \alpha_{a,b} \Big(|0\rangle^{\otimes n-k-1-t} \big(\bigotimes_{i=1}^{t+1} |a_i^{bin}\rangle \big) \\ |1\rangle^{\otimes k-1-t} \big(\bigotimes_{j=1}^{t+1} |b_j^{bin}\rangle \big) \Big).$$

Proof. The statement implies that the first $n-k-t-1$ qubits are all in the state $|0\rangle$ and the $(n-k+1)$-th qubit and the next $k-t-1$ qubits are all in the state $|1\rangle$ in all active basis states of the n qubit system after the block SCS_k^{n-t} has acted on it.

We first prove the statement for $t=0$. The n qubit system is at first in the state $|\psi_0\rangle = |0\rangle^{\otimes(n-k)} |1\rangle^{\otimes k}$ and the block to be applied is SCS_k^n. This block consists of the gates μ_n, $\mathcal{M}_n^i, n-1 \leq i \leq n-k+1$. We know that the transformation μ_n affects the $(n-1)$ and the n-th qubit and the transformation $\mathcal{M}_n^l$ affects the $(l-1)$ and n th qubit. Additionally μ_n acts as identity on a basis state if the $n-q$th qubit of the basis state is in the $|1\rangle$ state. Similarly $\mathcal{M}_n^l$ acts as identity on a basis state if the $(n-l-1)$-th qubit is in the $|1\rangle$ state.

The last k qubits of $|\psi_0\rangle$ are in the state $|1\rangle$ and therefore μ_n and $\mathcal{M}_n^{n-1}, \mathcal{M}_n^{n-2}, \ldots \mathcal{M}_n^{n-k+2}$ act as identity transformations. Finally the transformation $\mathcal{M}_n^{n-k+2}$ is applied. The first qubit to this transformation is in the state $|0\rangle$ and therefore this transformation may lead to basis states with either $|0\rangle$ or $|1\rangle$ in the $(n-k)$-th and n-th positions. Therefore the resultant state can be written as

$$\sum_{a_1,a_2 \in \{0,1\}} \alpha_{a_1 a_2} |0\rangle^{\otimes n-k-1} |a_1\rangle |1\rangle^{\otimes k-1} |a_2\rangle .$$

Thus the first $n-k-1$ qubits are all in the state $|0\rangle$ and the $(n-k+1)$-th qubit and the next $k-1$ qubits are all in the $|1\rangle$ state in all active basis states of the system.This concludes the base case.

Now assuming that our statement holds true for some $t-1 < k-2$ we show that the statement also holds for t. The SCS_k^{n-t} block is composed of the transformations μ_{n-t}, $\mathcal{M}_{n-t}^i, n-t-1 \leq$

$t \leq n-t-k+1$. The n qubit system is in the state

$$|\psi_{t-1}\rangle = \sum_{a=0}^{2^t-1} \sum_{b=0}^{2^t-1} \alpha_{a,b} \Big(|0\rangle^{\otimes n-k-t} \big(\bigotimes_{i=1}^{t} |a_i^{bin}\rangle \big) \\ |1\rangle^{\otimes k-t} \big(\bigotimes_{j=1}^{t} |b_j^{bin}\rangle \big) \Big).$$

That is, the first $n-k-t$ qubits are in the state $|0\rangle$ in all active basis states and the $n-k+1$ and the next $k-t-1$ qubits are in the state $|1\rangle$. These are the first qubits to the transformations $\mu_{n-t}, \mathcal{M}_{n-t}^i, n-t-1 \leq i \leq n-k$. This implies the μ transformation and the $(k-2-t)$ $\mathcal{M}$ transformations act as identity transformations on all active basis states.

The next t $\mathcal{M}$ transformations may get the $|0\rangle$ state as the first qubit and therefore the n-qubit system before the last $\mathcal{M}$ has been applied is in the state

$$|\psi'_{t-1}\rangle = \sum_{a=0}^{2^t-1} \sum_{b=0}^{2^{t+1}-1} \alpha_{a,b} \Big(|0\rangle^{\otimes n-k-t} \big(\bigotimes_{i=1}^{t} |a_i^{bin}\rangle \big) \\ |1\rangle^{\otimes k-1-t} \big(\bigotimes_{j=1}^{t+1} |b_j^{bin}\rangle \big) \Big).$$

Finally, the last three qubit transformation of the block SCS_k^{n-t} $\mathcal{M}_{n-t}^{n-t-k+1}$ acts on the system. Now since the $(n-t-k)$-th qubit is in the state $|0\rangle$ in all active basis states, the $\mathcal{M}$ gate may non trivially act on it and the $(n-t)$-th qubit. This results in the state

$$|\psi_t\rangle = \sum_{a=0}^{2^{t+1}-1} \sum_{b=0}^{2^{t+1}-1} \alpha_{a,b} \Big(|0\rangle^{\otimes n-k-t-1} \big(\bigotimes_{i=1}^{t+1} |a_i^{bin}\rangle \big) \\ |1\rangle^{\otimes k-1-t} \big(\bigotimes_{j=1}^{t+1} |b_j^{bin}\rangle \big) \Big).$$

This completes the proof. It is important to note that there may be many basis states in the expression of $|\psi_t\rangle$ with zero amplitude. However our focus is on qubits that are definitely going to be either in the zero state or in the one state in all active basis states. □

This proof also shows that the μ transformation and the $k-2-t$ $\mathcal{M}$ transformations in the block $SCS_k^{n-t}, t < k-1$ act as identity transformations and therefore can be removed from the circuit.

Therefore the number of μ transformations omitted is $k-1$ and the number of $\mathcal{M}$ transformation omitted are $\frac{(k-2)(k-1)}{2}$. This removes $3(k-1)+\frac{5(k-2)(k-1)}{2}$ CNOT and $2(k-1)+\frac{4(k-2)(k-1)}{2}$ single qubit gates.

1.4.3 The first non identity $\mathcal{M}$ transformation in SCS_k^n

Once the μ and $\mathcal{M}$ transformations are removed, we are essentially left with no redundant transformation. But that does not imply this circuit is optimal. For some controlled unitary operations, the control is always in the state $|0\rangle$ in all active basis state, in which case the whole operation can be removed. In some other cases the controls to a controlled unitary is always in the $|1\rangle$ states in all active basis states, and sometimes some of the controls are always in the $|1\rangle$ state, in which case these controls can be removed, which reduces the number of CNOT gates in the circuit. Here we analyze the first transformation of the block $SCS_k^{n-t}, t < k-1$ after the identity transformations are removed, which is $\mathcal{M}_{n-t}^{n-k+1}$. This transformation depends on the state of the $n-k, n-k+1$ and $(n-t)$-th qubits and affects the state of the $(n-k)$-th and the $(n-t)$-th qubit. At this stage the $n=$ qubit system is at the state

$$\sum_{a=0}^{2^t-1}\sum_{b=0}^{2^t-1}\alpha_{a,b}\,|0\rangle^{\otimes n-k-t}\left(\bigotimes_{i=1}^{t}|a_i^{bin}\rangle\right)|1\rangle^{\otimes k-t}\left(\bigotimes_{j=1}^{t}|b_j^{bin}\rangle\right).$$

Therefore in all the active basis states both the $n-k$th and the $n-t$th qubits are in the state $|1\rangle$. Therefore the three qubit transformation applied by $\mathcal{M}_{n-t}^{n-k+1}$ can be expressed as follows, substituting $l=n-k+1$:

$$\begin{aligned}|11\rangle_l\,|1\rangle_{n-t} &\to |11\rangle_l\,|1\rangle_{n-t}\\ |01\rangle_l\,|1\rangle_{n-t} &\to \sqrt{\frac{n-t-l+1}{n-t}}\,|01\rangle_l\,|1\rangle_{n-t}\\ &\qquad + \sqrt{\frac{l-1}{n-t}}\,|11\rangle_l\,|0\rangle_{n-t}\,.\end{aligned}$$

This is in-fact can be implemented as a a two qubit transformation of the type μ. as the $(n-k+1)$-th qubit is in the $|1\rangle$ state in all the active basis states.

The transformation acts on the $(n-k)$-th and $(n-t)$-th qubits as

$\mathcal{M}_{n-t}^{n-k+1} \equiv (\mathsf{CNOT}_{n-t}^{n-k})(CU_{n-k}^{n-t}(\theta))(\mathsf{CNOT}_{n-t}^{n-k})$ where $\theta = 2\cos^{-1}\left(\sqrt{\frac{n-t-l+1}{n-t}}\right)$. We know that the CU gate requires one CNOT and two R_y gates to be implemented therefore $\mathcal{M}_{n-t}^{n-k+1}$ requires

only three CNOT and two R_y gates. This improvement is reflected for all SCS_k^{n-t} such that $n-t \geq n-k+2$ that is for $0 \leq t \leq k-2$. Therefore it reduces the number of CNOT gate in the circuit by further $2(k-1)$ and the number of single qubit gates by $2(k-1)$ as well.

Additionally for $|D_k^n\rangle$, $k > 1$ when SCS_k^n is applied the n-qubit system is in the state $|0\rangle^{\otimes n-k}|1\rangle^{\otimes k}$ and therefore the transformation $\mathcal{M}_{n-t}^{n-k+1}$ only acts on the basis state $|011\rangle$. The corresponding transformation is

$$|01\rangle_{n-k+1}|1\rangle_n \rightarrow \sqrt{\frac{k}{n}}|01\rangle_{n-k+1}|1\rangle_n + \sqrt{\frac{n-k}{n}}|11\rangle_{n-k+1}|0\rangle_n .$$

This can be implemented using a $R_y(\cos^{-1}\sqrt{\frac{k}{n}})$ on the $(n-k)$-th qubit followed by a CNOT gate CNOT_n^{n-k} which removes further two CNOT and one single qubit gate.

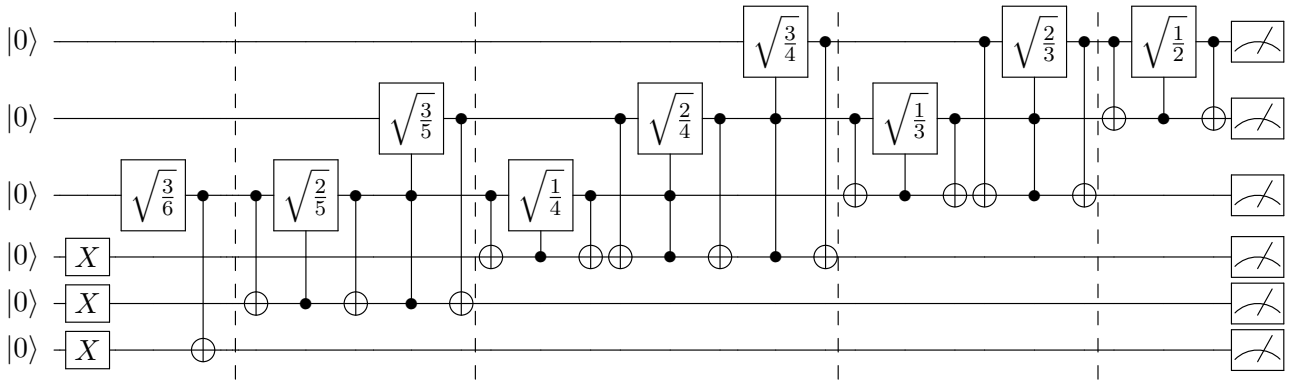

Figure 1.6: Description of the Circuit $\widehat{\mathcal{C}}_{6,3}$

We denote this circuit by $\widehat{\mathcal{C}}_{n,k}$. Figure 1.6 shows the structure of $\widehat{\mathcal{C}}_{6,3}$. Combining these modifications one can get the following count of CNOT and single qubit gates in the improved circuit.

- The total number of CNOT gates removed $= n-1+3(k-1)+\frac{5(k-2)(k-1)}{2}+2(k-1)+2$.
- The total number of single qubit gates removed $= 2(k-1)+\frac{4(k-2)(k-1)}{2}+2(k-1)+1$.

Therefore the total number of CNOT gates present in the circuit is

$$\begin{aligned}
&5(nk - \frac{k(k+1)}{2}) - n + 1 \\
&- \Big(n - 1 + 3(k-1) + \frac{5(k-2)(k-1)}{2} + 2(k-1) + 2\Big) \\
&= 5nk - 5k^2 - 2n
\end{aligned}$$

The number of single qubit gates present in the circuit is

$$\begin{aligned}
&4(nk - \frac{k(k+1)}{2} - n + 1) + 2(n-1) \\
&- \Big(2(k-1) + \frac{4(k-2)(k-1)}{2} + 2(k-1) + 1\Big) \\
&= 4nk - 4k^2 - 2n + 1
\end{aligned}$$

For $k = 1$ the number of CNOT gates is $3n - 3$ (from $n - 1$ μ transformations) and the number of single qubits gate as $2n - 2$. However, one CNOT gate can be further removed from each μ gate as the active basis states in input to the μ transformations are only $|00\rangle$ and $|01\rangle$. The resultant circuit is identical to the linear W_n preparation circuit in [5] and contains $2n - 2$ CNOT and $2n - 2$ single qubit gates and thus we don't elaborate it further.

We know that the state $|D_k^n\rangle$ can be prepared by first forming the state $|D_{n-k}^n\rangle$ and then applying a X gate to each qubit. On that note it is interesting to observe that after these modifications the circuits for $|D_k^n\rangle$ and $|D_{n-k}^n\rangle$ require the same number of CNOT gates.

k \ n	4	5	6	7	8
2	22,12	31,20	40,28	49,36	58,44
3	27,7	41,20	55,33	69,46	83,59
4		46,10	65,28	84,46	103,64
5			70,13	94,36	118,59
6				99,16	128,44
7					133,19

Table 1.2: CNOT gate count of the pair $\mathcal{C}_{n,k}, \widehat{\mathcal{C}}_{n,k}$

k \ n	4	5	6	7	8
2	14,9	20,15	26,21	32,27	38,33
3	18,5	28,15	38,25	48,35	58,45
4		32,7	46,21	60,35	74,49
5			50,9	68,27	86,45
6				72,11	94,33
7					98,13

Table 1.3: Single qubit gate count of the pair $\mathcal{C}_{n,k}, \widehat{\mathcal{C}}_{n,k}$

Table 1.2 and 1.3 show the number of CNOT and single qubit gates needed to implement the states $|D_k^n\rangle$ for $4 \leq n \leq 8, 1 \leq k \leq n-1$, respectively.

In the next section we discuss how these modifications not only reduces the gate counts of the circuit but also relaxes its architectural constraints.

1.5 Actual Implementation and architectural constraints

1.5.1 Architectural Constraints

We are at the stage where quantum circuits can be implemented on actual quantum computers using cloud services, such as IBM Quantum Experience, also known as IQX [16]. However the architecture of the individual back-end quantum machines pose restrictions to implementation of a particular circuit. The most prominent constraint is that of the CNOT implementation. In this regard we use the terms architectural constraint and CNOT constraint interchangeably. Every quantum system Q with n (physical) qubits has a CNOT map, which we express as $G_A^Q(V^Q, E^Q)$ where $V^Q = \{q_1, q_2, \ldots q_n\}$. In this graph the nodes represent the qubits and the edges represent CNOT implementability. A directed edge $q_i \rightarrow q_j$ implies that a CNOT can be implemented with q_i as control and q_j as target in the system Q. The edges in the CNOT maps of all the publicly available IQX machines are bidirectional.

Let there be a circuit $\mathcal{C}$ on n qubits. We denote the (logical) qubits $c_1, c_2, \ldots c_n$. We also have a CNOT map corresponding to the circuit, which describes the CNOT gates used in the circuit. We

describe this as the directed graph $G_{\mathcal{C}}(V^{\mathcal{C}}, E^{\mathcal{C}})$ where $V^{\mathcal{C}} = \{c_1, c_2, \ldots c_m\}$ and there is a directed edge $c_i \rightarrow c_j$ if there is one or more CNOT with c_i as control and c_j as target.

Therefore if the graph $G_{\mathcal{C}}$ can shown to be the subgraph of G_A^Q by some mapping of the logical qubits to the physical qubits then the circuit $\mathcal{C}$ can be implemented on the architecture with the same number of CNOT gates. However, if such a map is not possible, then the circuit can be implemented on that architecture, either by using swap gates which require additional CNOT gates or changing the construction of the circuit. There are mapping solutions such as the one applied by IQX which dynamically changes the structure of the circuit to implement a circuit in an architecture that does not meet the circuit's CNOT constraints. Similarly in their paper Zulehner et.al [15] have also proposed an efficient mapping solution. However it is not always possible to avoid an increase in the number of CNOT gates. Given a circuit it is crucial to find it's minimal architectural needs in terms of the CNOT map without increasing the CNOT gate count. The IBM-Q systems mapping solutions show the modified circuit as the transpiled circuit given any circuit as input, although its solutions are not always optimal. In this chapter we first consider the system "ibmqx2" (Q_1) of IQX. The CNOT map of Q_1 is shown in the Figure 1.7.

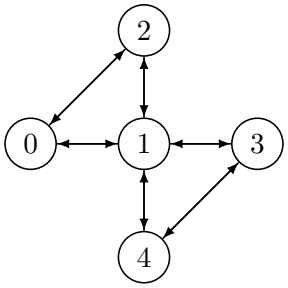

Figure 1.7: The CNOT map of Q_1 represented as $G_A^{Q_1}$

Against this backdrop we first observe the CNOT constraints of the circuit $\mathcal{C}_{4,2}$, implemented to prepare $|D_2^4\rangle$. Then we implement the circuit $\widehat{\mathcal{C}}_{4,2}$ which is the result of the improvements shown in Section 4.3. We shall see that the modifications that we have discussed in the previous sections not only reduces gate counts but also reduces CNOT constraints. We shall implement these circuits in the system Q_1 and compare the measurement statistics of the two circuits by measuring the deviation from the ideal measurement statistics and find that the results of $\widehat{\mathcal{C}}_{4,2}$ is much more closely aligned with the ideal results. To solidify the argument of reduced error, we will perform

full state tomography, which simulates the state prepared by the system in form of a density matrix.

We end this section by discussing how some changes in the circuit $\widehat{\mathcal{C}}_{4,2}$ possible because of partially defined transformations can lower the error in the circuit due to CNOT on expectation without a reduction in number of CNOT gates or change in the CNOT constraints.

1.5.2 Implementation and Improvement for $|D_2^4\rangle$

Let us first construct the circuit $\mathcal{C}_{4,2}$. We shall implement every CR_y gate using two CNOT gates and two R_y gate as we know that the CR_y gate needs at least 4 gates to be implemented and every three qubit $\mathcal{M}_n^l$ transformation using five CNOT and four R_y gates (as given in the description of [2]). The resultant circuit $\mathcal{C}_{4,2}$ is shown in Figure 1.8. This circuit contains 22 CNOT gates. The corresponding QASM code for $\mathcal{C}_{4,2}$ and $\widehat{\mathcal{C}}_{4,2}$ have been attached in the appendix as Listing 1.3 and Listing 1.4 respectively.

We use the notation θ_y^x to denote the angle $2\cos^{-1}(\sqrt{\frac{x}{y}})$.

The CNOT map of the circuit is shown in Figure 1.9.

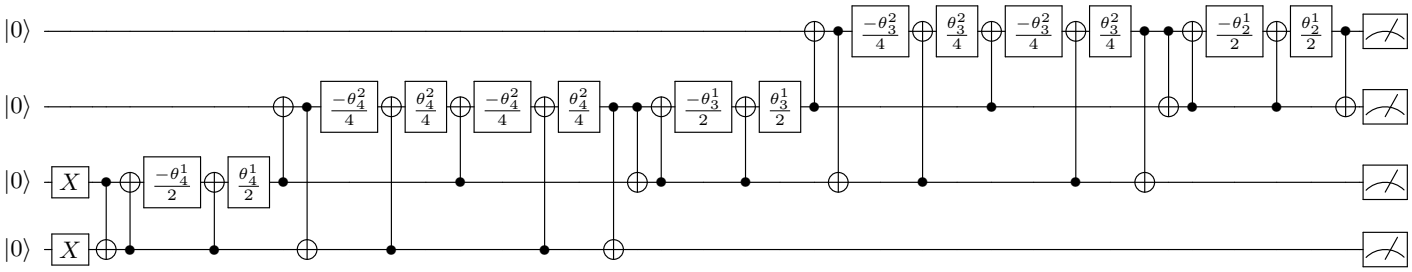

Figure 1.8: The circuit $\mathcal{C}_{4,2}$ due to [2]

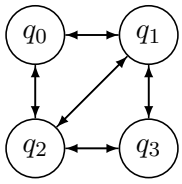

Figure 1.9: The CNOT map of $\mathcal{C}_{4,2}$

Then the modified circuit $\widehat{\mathcal{C}}_{4,2}$ can be implemented by making the following changes to $\mathcal{C}_{4,2}$.

1. Implement the CU^o gate instead of CR_y gates.
2. Remove the Redundant μ and $\mathcal{M}$ transformation.

3. Reduce the gate count in implementation of $\mathcal{M}_{n-t}^{n-k+1}$ type transformations.

This brings the total number of CNOT gates in the circuit to 12. We name the circuit at this stage $\widehat{\mathcal{C}}_{4,2}$.

These steps not only reduce the CNOT gates in the circuit but also reduces the CNOT constraints of the circuit. Figure 1.10 shows the circuit at this stage and the reduced CNOT map $G_{\widehat{\mathcal{C}}_{4,2}}$ as shown in the Figure 1.11.

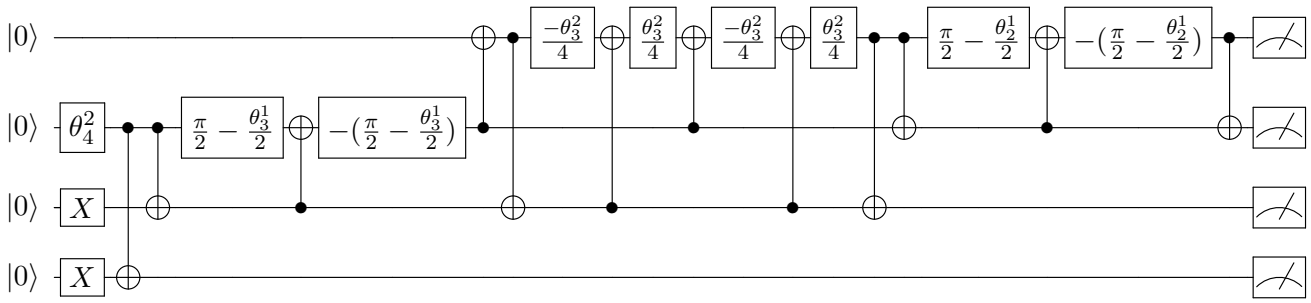

Figure 1.10: The Circuit $\widehat{\mathcal{C}}_{4,2}$

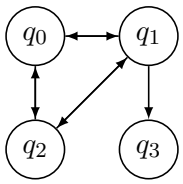

Figure 1.11: The CNOT map $G^{\widehat{\mathcal{C}}_{4,2}}$ corresponding to the circuit $\widehat{\mathcal{C}}_{4,2}$

In fact the Graph $G^{\widehat{\mathcal{C}}_{4,2}}$ can be shown to be a subgraph of $G_A^{Q_1}$ under several mappings of qubits. Therefore this circuit can be implemented in the "ibmqx2" (Q_1) machine with 12 CNOT gates. Here it should be noted that any 4 qubit state can be prepared using a maximum of 9-CNOT gates. However the corresponding CNOT map is a 4-cycle [11], which is also not available in the "ibmqx2" machine.

However the CNOT constraints of the circuits corresponding to even $D_k^5, k > 1$ cannot be met by any IBM-Q architecture at this stage. Now let us compare the results of the circuits $\mathcal{C}_{4,2}$ which is due to [2] and $\widehat{\mathcal{C}}_{4,2}$ which is what we obtained after the reductions and modifications.

Comparison of Measurement Statistics of $\mathcal{C}_{4,2}$ and $\widehat{\mathcal{C}}4,2$

The output by an ideal Quantum Computer would produce the state $\sqrt{\frac{1}{\binom{n}{k}}} \sum_{wt(i)=k} |i_{(2)}\rangle$ on a correct $|D_k^n\rangle$ preparation circuit. One can verify the resultant state vectors of the two circuits to see that they both ideally produce $\sqrt{\frac{1}{6}}\Big(|0011\rangle + |0101\rangle + |0110\rangle + |1100\rangle + |1010\rangle + |1001\rangle \Big)$ using a simulator.

Here we first discuss a primary error measure based on measurement in computational basis to estimate the closeness of the states formed by the two circuits from the ideal state $|D_2^4\rangle$. Let us run both the circuits for the maximum possible shots (8192) and use the measurement statistics to estimate the closeness to the desired state using the following error measure. We define our error measure $EM_{n,k}$ for the Dicke state $|D_k^n\rangle$ as follows. Let p_i be the percentage of times the measurement of the circuit $\mathcal{C}$ yields the result $i_{(2)}$. Then we have

$$EM_{n,k}(\mathcal{C}) = \frac{1}{2}\Big(\sum_{j,wt(j)=k} |*|\, p_j - \frac{1}{\binom{n}{k}} + \sum_{j,wt(j)\neq k} p_j \Big)$$

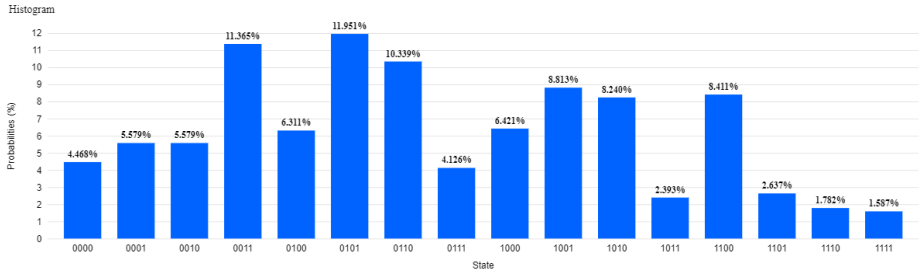

Figure 1.12: Measurement statistics for $\mathcal{C}_{4,2}$ with 25 CNOT in transpiled circuit

An EM value of 0 signifies that the measurement statistics are exactly aligned with the expected ideal results while the EM value can at maximum be 1. We have calculated the EM values for results of different mappings for the circuit $\mathcal{C}_{4,2}$. The transpiled circuits for $\mathcal{C}_{4,2}$ had a minimum of 25 CNOT gates and were as high as 31 in some cases. The corresponding transpiled circuit contains 25 CNOT gates which is the least of all the transpiled circuits. Figure 1.12 shows the measurement statistics corresponding to the circuit with the minimum EM value, which is equal to 0.4088.

Next we look at the measurement statistics of the circuit $\widehat{\mathcal{C}}_{4,2}$. There are many mappings between logical and physical qubits in this case such that the CNOT constraint of the circuit is

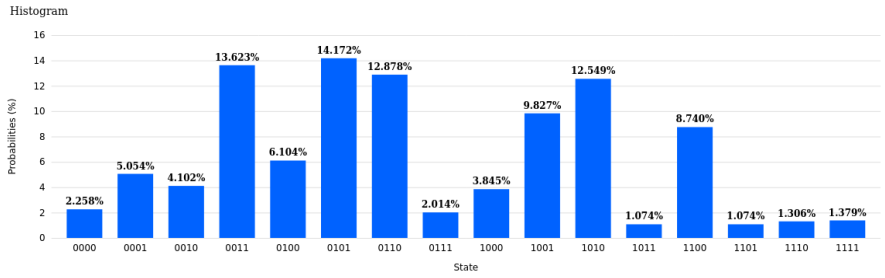

Figure 1.13: Measurement statistics for $\widehat{\mathcal{C}}_{4,2}$ corresponding to the map M_1

met. Let such a map be $M : \{q_0, q_2, q_3, q_4\} \to \{0, 1, 2, 3, 4\}$. Then if there is a CNOT between q_i and q_j then there is an edge $M(q_i) \leftrightarrow M(q_j)$ in graph $G_A^{Q_1}$. In such mappings the IQX mapping solution didn't implement any modification in the transpiled circuit as expected.

Here we present the result for the following map M_1

$$M1: \quad q_0 \to 3, \quad q_1 \to 2, \quad q_2 \to 4, \quad q_3 \to 0.$$

Figure 1.13 shows the measurement statistics corresponding to this mapping and the resultant EM value is 0.282103. These results show that the circuit $\mathcal{C}_{4,2}$ needs more than the specified number of CNOT while being implemented on "ibmqx2" and the measurement statistics of $\widehat{\mathcal{C}}_{4,2}$ is much more closely aligned with the ideal measurement statistics compared to $\mathcal{C}_{4,2}$.

Different Optimization Levels and Tomography results

The results that we have portrayed so far are obtained without exhausting the ability of the QISKIT compiler. There are four optimization levels that the QISKIT compiler of IBM-QX provides, with 0 being the least and 3 being the maximum possible optimization level, where the default level is 1. Compiling the circuit with different optimization level results in different transpiled circuits. We know that the number of CNOT gates in $\mathcal{C}_{4,2}$ is originally 22. The best that we could achieve using the different optimization levels is 19. This is better than the original count of 22, but still quite higher than the number of CNOT gates in $\widehat{\mathcal{C}}_{4,2}$ which is 12. We now also discuss the quality of the states prepared using these circuits via full state tomography.

Although the measurement EM gives us an idea about the error induced in the circuits, it does not account for the full picture. For example, we cannot identify the imaginary amplitude

that the circuits induce separately, rather we only get the real norm value. To understand the complete structure of the state prepared for these circuits, full state tomography needs to be done. Since the circuit consists of 4 qubits, we need to perform measurements in $3^4 = 81$ different ways. Tomography is used to measure the fidelity of the obtained state with the ideal state which is well known measure of closeness. A fidelity of 1(maximum value) implies that the two states are identical and a fidelity of 0(minimum value) implies the two states are completely distinguishable. Let us now measure the fidelity of the state prepared by the following circuits and the ideal $|D_2^4\rangle$ state.

1. The original circuit description, $\mathcal{C}_{4,2}$

2. The circuit compiled by the transpiler using maximum possible optimization, which we denote as $trans(\mathcal{C}_{4,2})$.

3. The improved circuit due to our modifications, $\widehat{\mathcal{C}}_{4,2}$.

Here one should note that, the error rates in the IBMQ systems vary significantly and therefore the results of tomography would be different if recorded on different time. The error rates in the IBMQ machines are calibrated roughly once every 24 hours and can be found at [16]. We present two such instances of results and find the relation between the fidelity of the three circuits remain unchanged. We now note down the results in Table 1.4 and Table 1.5.

We have plotted the density matrices corresponding to the simulator and the result of $\widehat{\mathcal{C}}_{4,2}$ described on Table 1.5 in Figure 1.14 and Figure 1.15 respectively. Each density matrix consists of two plots, one depicting the real and the other depicting the complex components of the density matrix.

Thus we can see that even when we account for full tomography, the modifications that we have discussed forms better state than the circuit formed by the transpiler using maximum optimization level.

1.5.3 Modifications leading to different CNOT error distributions

We have discussed possible modifications to reduce gate count of the circuit for preparing the Dicke state $|D_k^n\rangle$ and have also seen how these modifications affect the error induced in the circuit with

Circuit	Backend	Fidelity
$\mathcal{C}_{4,2}, \widehat{\mathcal{C}}_{4,2}$	Simulator	1
$\mathcal{C}_{4,2}$	"ibmqx2"	≈ 0.40
$trans(\mathcal{C}_{4,2})$	"ibmqx2"	≈ 0.43
$\widehat{\mathcal{C}}_{4,2}$	"ibmqx2"	≈ 0.53

Table 1.4: Fidelity of $|D_2^4\rangle$ prepared using different circuits on 20/09/2021

Circuit	Backend	Fidelity
$\mathcal{C}_{4,2}, \widehat{\mathcal{C}}_{4,2}$	Simulator	1
$\mathcal{C}_{4,2}$	"ibmqx2"	≈ 0.25
$trans(\mathcal{C}_{4,2})$	"ibmqx2"	≈ 0.35
$\widehat{\mathcal{C}}_{4,2}$	"ibmqx2"	≈ 0.40

Table 1.5: Fidelity of $|D_2^4\rangle$ prepared using different circuits on 04/11/2021

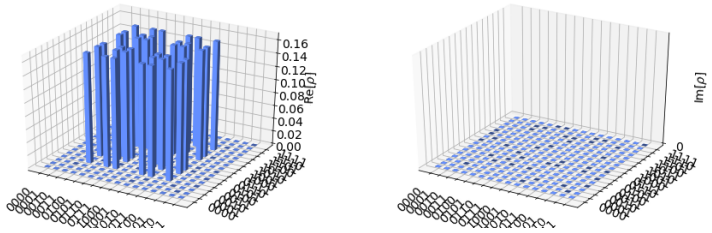

Figure 1.14: Density matrix corresponding to $|D_2^4\rangle$ when run in simulator

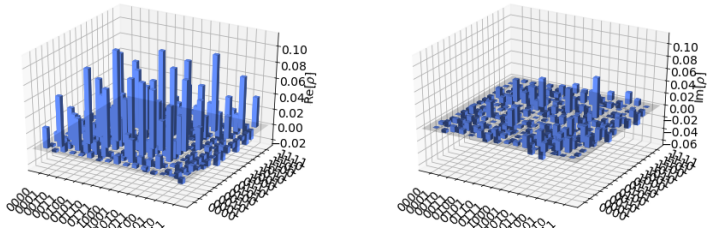

Figure 1.15: Density matrix corresponding to $\widehat{\mathcal{C}}_{4,2}$ corresponding to the result in Table 1.5

respect to the " imbqx2" machine provided by IBM-QX. The main tool behind these modifications were identification and analysis of different partially defined unitary transformations.

Let us now shift our focus to observing how these partially defined transformations can also

come in handy when analyzing the expected amount of noise affecting a circuit. We describe this scenario with the case study of $\widehat{\mathcal{C}}_{4,2}$. Consider a four qubit architecture A_4 that has the same CNOT connectivity as $G^{\widehat{\mathcal{C}}_{4,2}}$ and only differs in CNOT error distribution. We will now observe how further modifying the circuit $\widehat{\mathcal{C}}_{4,2}$ can lead to lower CNOT error on expectation against some error distributions in the architecture A_4. We assume every edge in the CNOT map of A_4 is bidirectional, as is the case with all currently publicly available IBM-Q machines. The CNOT error rate when applying a CNOT between qubits i and j (such that the edge $i \leftrightarrow j$ is present in G_A) is denoted as e_{ij}.Figure 1.17 shows the CNOT map of the architecture.

CNOT error model

In this regard let us define a plausible error model to calculate CNOT error on expectation of a circuit implemented in the architecture A_4. The probability of a CNOT placed between qubits i and j acting erroneously in a circuit is dependent on the error rate of the corresponding CNOT coupling in the architecture. We call this CNOT error. We denote this probability with $f_e : [0, 1] \rightarrow [0, 1]$. We do not assume the exact nature of f_e, but only that it is a monotonically increasing function w.r.t to CNOT error rate which is by definition.

Next we define the following Bernoulli random variables to calculate the the number of CNOT acting erroneously on expectation when a circuit is applied on this architecture. We define a variable x_k corresponding to each CNOT used in a circuit. The variable is assigned zero if the k-th CNOT is applied correctly while executing a circuit, and one otherwise.

Let us suppose the k-th CNOT is applied between qubits i and j. Then we have $Pr(x_k = 1) = f_e(e_{ij})$ and The expected error while applying the CNOT is $\mathbb{E}(x_k) = f_e(e_{ij})$. Therefore the CNOT error on expectation while implementing a circuit $\mathcal{C}$ on the architecture is

$$\mathbb{E}(\mathcal{C}) = \sum v_{ij} f_e(e_{ij}).$$

Having described the error model we look at the CNOT distribution of the circuit $\widehat{\mathcal{C}}_{4,2}$ as a weighted graph G_f. The vertices and the edges of this graph is same as that of $G^{\widehat{\mathcal{C}}_{4,2}}$. The weight of an edge $q_i \leftrightarrow q_j$ is the number of CNOT gates applied between the two qubits in the circuit. The graph G_f is shown in Figure 1.16.

Now we implement the circuit $\widehat{\mathcal{C}}_{4,2}$ on the architecture A_4 so that all CNOT constraints can be

met. We observe that only the qubit 1 has degree 3 and therefore q_1 is mapped to the physical qubit 1. Then we can have the following maps which satisfies all the CNOT constraints.

1. $q_1 \to 1$, $q_0 \to 0$, $q_2 \to 2$, $q_3 \to 3$.
2. $q_1 \to 1$, $q_0 \to 2$, $q_2 \to 0$, $q_3 \to 3$.

Then the expected CNOT error of the circuit $\widehat{\mathcal{C}}_{4,2}$ when applied on the architecture A_4 is

$$\mathbb{E}(\widehat{\mathcal{C}}_{4,2}) = 5f_e(e_{01}) + 3f_e(e_{02}) + 3f_e(e_{12}) + f_e(e_{13}).$$

Now let us observe how the circuit $\widehat{\mathcal{C}}_{4,2}$ (described in Figure 1.10) can be further modified using partially defined transformations so that the CNOT error in the circuit on this architecture will reduce on expectation under some error distribution conditions.

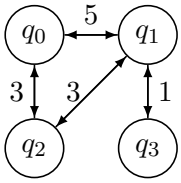

Figure 1.16: Graph G_f corresponding to $G^{\widehat{\mathcal{C}}_{4,2}}$.

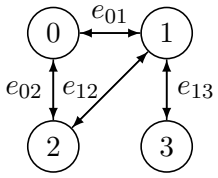

Figure 1.17: CNOT map of the architecture A.

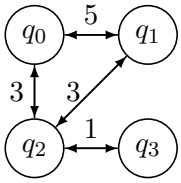

Figure 1.18: Graph G'_f corresponding to $\widehat{\mathcal{C}}'_{4,2}$

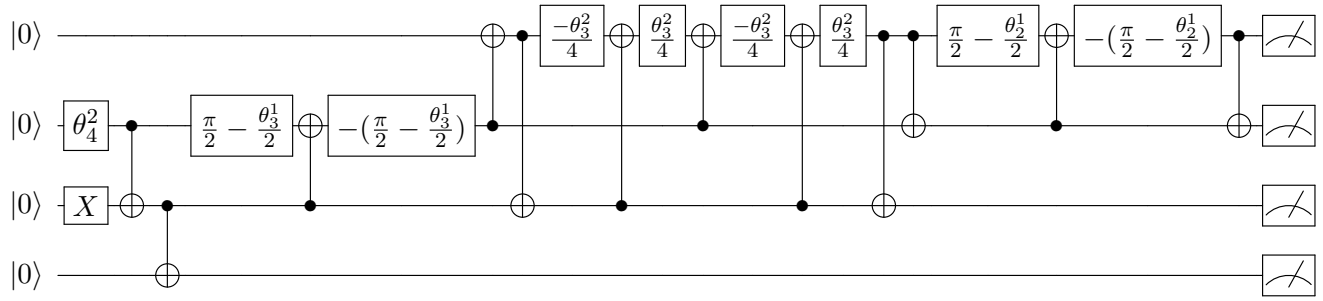

Figure 1.19: Circuit description of $\widehat{\mathcal{C}}'_{4,2}$

The first R_y gate that acts on the second qubit of $\widehat{\mathcal{C}}_{4,2}$ is followed by the CNOT gates CNOT_3^2 and CNOT_4^2. The combined transformation T_4 of these two CNOT is defined only for two basis states on 4 qubits $|0011\rangle \to |0011\rangle$ and $|0111\rangle \to |0100\rangle$.

We can use the partial nature of the transformation to modify the circuit as follows. Note that transformation T_4 is the first transformation that acts on q_3. Then if we start the circuit from the

state $|0010\rangle$ instead of $|0011\rangle$ then we can define the transformation T_4^1 such that

$$\begin{aligned} T_4^1 &\equiv (I_2 \otimes \mathsf{CNOT}_2^3 \otimes I_2)(I_2 \otimes I_2 \otimes \mathsf{CNOT}_1^2) \\ \implies T_4^1 |0010\rangle &= |0011\rangle\,, \quad T_4^1 |0110\rangle = |0100\rangle \end{aligned}$$

resulting in the same output states as T_4 for all the computational basis states for which T_4 is defined. It is important to note that this implementation would not have been possible if the transformation was defined for all the 8 basis states of the second third and fourth qubits.

Let's denote this circuit as $\widehat{\mathcal{C}}'_{4,2}$ and it is drawn in Figure 1.19. The weighted CNOT map of the circuit $\widehat{\mathcal{C}}'_{4,2}$ is denoted as as G'_f and it is shown in Figure 1.18.

Now observe that in the graph G'_f ,q_2 has degree three and therefore any map that meets all the CNOT constraints will have $q_2 \to 1$. Therefore we can have the following maps that satisfies all the CNOT constraints.

1. $q_2 \to 1, \quad q_0 \to 0, \quad q_1 \to 2, \quad q_3 \to 3.$
2. $q_2 \to 1, \quad q_0 \to 2, \quad q_1 \to 0, \quad q_3 \to 3.$

The CNOT error on expectation for both the circuits is

$$\mathbb{E}(\widehat{\mathcal{C}}'_{4,2}) = 5f_e(e_{02}) + 3f_e(e_{01}) + 3f_e(e_{12}) + f_e(e_{13}).$$

Now we calculate the conditions when $\mathbb{E}(\widehat{\mathcal{C}}'_{4,2})$ is less than $\mathbb{E}(\widehat{\mathcal{C}}'_{4,2})$.

$$\begin{aligned} &\mathbb{E}(\widehat{\mathcal{C}}'_{4,2}) < \mathbb{E}(\widehat{\mathcal{C}}_{4,2}) \\ \implies &5f_e(e_{02}) + 3f_e(e_{01}) + 3f_e(e_{12}) + f_e(e_{13}) \\ &\quad < 5f_e(e_{01}) + 3f_e(e_{02}) + 3f_e(e_{12}) + f_e(e_{13}) \\ \implies &f_e(e_{02}) < f_e(e_{01}) \\ \implies &e_{02} < e_{01}. \end{aligned}$$

This gives us an insight into how different CNOT distributions in a circuit may lead to better results without reduction in the number of CNOT gates or a reduction in the architectural constraints. We conclude this section by describing the architectural constraint of the circuit $\widehat{\mathcal{C}}_{n,k}$. Here note that this CNOT error model is rather simplistic and we design this model to emphasize how the

distribution of the CNOT gates in the circuit is also of importance in trying to reduce the error due to the noise in the system. One can refer to [14] to get a more formal approach towards noise aware circuit compilation.

1.5.4 The CNOT map of $\widehat{\mathcal{C}}_{n,k}$

The CNOT gates in the circuit $\widehat{\mathcal{C}}_{n,k}$ are due to implementation of the μ and $\mathcal{M}$ transformation of the different SCS_k^n blocks. μ_n forms an edge in the CNOT map of the form $n-1 \leftrightarrow n$. where as $\mathcal{M}_n^l$ forms the edges $(l-1) \leftrightarrow n$ and $l \rightarrow (l-1)$. However in the circuit $\widehat{\mathcal{C}}_{n,k}$ the transformations $\mathcal{M}_t^{n-k+1}$ do not have a CNOT between the neighboring qubits $n-k$ and $n-k+1$.

We divide the edges into two groups. One corresponding to CNOT gates between neighboring qubits and one where the positions of the qubits differ at least by two. We calculate the edges of each of these types.

- The neighboring qubits with CNOT connections are the qubits $(n-k+1-i)$ and $(n-k-i)$ where i varies from 0 to $n-k-1$. The other neighboring qubits do not have CNOT connections due to removal of identity transformations form those qubits. This results in $n-k$ edges.

- Now consider the second kind of connections. These connections are formed between $l-1$ and t th qubit for any $\mathcal{M}_t^l$ transformation.

  There are $n-k$ SCS_K^n blocks with originally $k-1$ $\mathcal{M}$ transformations in $\mathcal{C}_{n,k}$ which forms the edges:

  $$(n-t) \leftrightarrow (n-t-2-i), \quad 0 \leq i \leq k-2, 0 \leq t \leq n-k-1.$$

  Then there are $k-1$ blocks of SCS_i^{i+1} with $i-1$ $\mathcal{M}$ transformations which forms the edges:

  $$(k-t) \leftrightarrow (k-t-2-i), \quad 0 \leq i \leq k-t-2, 0 \leq t \leq k-2.$$

  However in $\widehat{\mathcal{C}}_{n,k}$ there are no $\mathcal{M}$ transformations of the type $\mathcal{M}_y^{n-k+1+x}$, $x > 0$. Removing such edges $n-k+x \leftrightarrow y$ gives us the complete description of the CNOT map of $\widehat{\mathcal{C}}_{n,k}$, which we denote by $G^{n,k}$. Additionally, in the transformation $\mathcal{M}_n^{n-k+1}$ the edge $n \rightarrow (n-k)$ is not present.

Figure 1.20 and 1.21 show the CNOT maps $G^{6,2}$ and $G^{6,3}$ respectively.

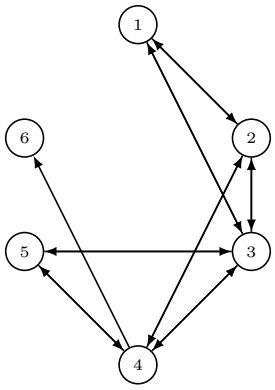

Figure 1.20: The CNOT map $G^{6,2}$

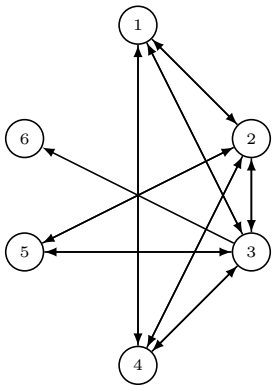

Figure 1.21: The CNOT map $G^{6,3}$

The number of edges present due to $\mathcal{M}$ transformations is $nk - \frac{k(k+1)}{2} - n + 1 - \frac{(k-1)(k-2)}{2} = nk - n - k^2 + k$. There are further $n-k$ edges due to the μ transformations. Which brings the total number of edges in $G^{n,k}$ to $nk - k^2$. It is important to note that although the number of edges in $G^{n,k}$ and $G^{n,n-k}$ are same they are not isomorphic. Moreover the Graph $G^{n,i}$ is not a subgraph of $G^{n,i+1}$. This is an interesting implication of the improvements that we have discussed and further analyzing the relation between the different $G^{n,i}$ graphs is an intriguing combinatorial prospect.

The final point of discussion in this document is the fact that circuit $\widehat{\mathcal{C}}_{n,k}$ can be modified so that the number of CNOT gates between the qubits change for certain cases, although the total number of CNOT gates and the overall CNOT map does not change. We call these different instances as different CNOT distributions of $\widehat{\mathcal{C}}_{n,k}$.

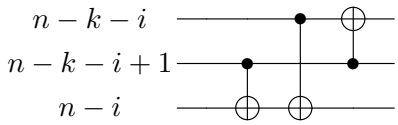

Figure 1.22: Initial Implementation

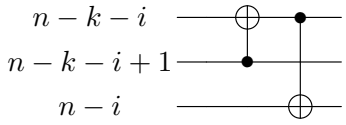

Figure 1.23: Modification in $\mathcal{C}_{n,k}$

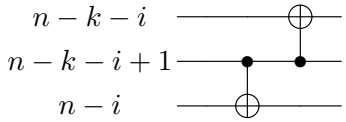

Figure 1.24: Alternate Implementation

Different CNOT distributions for $\widehat{\mathcal{C}}_{n,k}$

We know from the description of $\mathcal{C}_{n,k}$ [2] that the number of CNOT gates in the three qubit transformation $\mathcal{M}$ is reduced from 6 to 5 by canceling the last CNOT of every transformation by rearranging the first two CNOT gates of the next transformation. Figure 1.22 shows the original layout as per the algorithm and Figure 1.23 shows the reduction due to [2].

Now let us consider the last transformation (k-th) of each $SCS_k^{n-i}, 0 < i < n-k$ block, $\mathcal{M}_{n-i}^{n-i-k+1}$. This transformation acts on the qubits $n-i-k, n-i-k+1$ and $(n-i)$. This is in fact the first transformation that affects the qubit $(n-i-k)$ and thus the qubit is in the state $|0\rangle$. If we do not cancel the last CNOT of the preceding $\mathcal{M}$ transformation ($CNOT_{n-i}^{n-i-k+1}$) this then enables us to remove of the CNOT gate ($CNOT_{n-i}^{n-i-k}$), changing the CNOT distribution of the circuit without a change in CNOT map or number of CNOT gates. This leads to the implementation shown in Figure 1.24. Since there are $n-k-1$ such transformations, this leads to a total of 2^{n-k-1} different CNOT distributions. However, these modifications do not alter the CNOT map of the circuit due to the fact that there are other CNOT gates applied between these qubits, which is evident from the circuit description. As we have observed in Section 1.5.3 such different CNOT distributions may lead to different number of CNOT gates acting erroneously on expectation and thus affect the overall error induced in the circuit.

1.6 Conclusion

In this chapter we have explored the domain of optimal circuit implementation in terms of CNOT and single qubit gates. In this regard we have shown how concise realization of partially defined unitary transformations can improve the gate count of the current state of the art deterministic Dicke state ($|D_k^n\rangle$) preparation circuit ($\mathcal{C}_{n,k}$). We have reduced the gate count of one such implementation and have also proven the optimality of our implementation. We have further shown ways in which one can further reduce the gate count of the said Dicke State preparation circuit by removing redundant gates and modifying implementations of certain partially defined unitary transformations depending on the active basis states that that act as input to these transformations. We have then discussed how these improvements not only reduce the number of CNOT and single qubit gates but also reduces the architectural constraints of the circuit using the case of $|D_2^4\rangle$. The resultant circuit is the deterministic Dicke State ($|D_k^n\rangle\,,\ 2 \leq k \leq n-1$) has the same asymptotic gate count as the original circuit $\mathcal{O}(nk)$, but we see the CNOT and single qubit gate count being reduced by $\mathcal{O}(k^2)$. We have then shown how to implement the circuit $\mathcal{C}_{4,2}$ and the improved circuit $\widehat{\mathcal{C}}_{4,2}$ on the IBM-Q machine "ibmqx2" and observed that the deviation from ideal measurement statistics is significantly lesser in case of $\widehat{\mathcal{C}}_{4,2}$. We have also discussed that the full tomography results also support this observation. Furthermore, we have discussed how different CNOT distributions can help a circuit without changing the number of gates or the architectural constraints by comparing the expected CNOT error of two such distributions against a fairly generalized error model. We have concluded by describing the CNOT map of the circuit $\widehat{\mathcal{C}}_{n,k}$ and observe the exponential number of different CNOT distributions that can be derived by modifying the circuit to complete our generalization.

We observed that even the circuits for $|D_2^5\rangle$ could not be implemented in the IBM back end machines without adding further CNOT gates to our description. This is because of incompatibility of the architecture and circuit CNOT maps. Therefore it is of all the more importance to form the circuit for an algorithm in the most concise way possible. In conclusion we note down the following optimization problems that should help the readers implement algorithms more efficiently in the current scenario.

1. Given a maximally partial unitary transformation what is the corresponding unitary matrix that can be decomposed using the least number of elementary gates?

2. Given two circuits corresponding to an algorithm with isomorphic CNOT maps and the same number of CNOT gates, but different CNOT distribution across the qubits, which circuit will produce less erroneous outcome?

Bibliography

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. W. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. 1995. Elementary gates for quantum computation. Phys. Rev. A, 52:3457-3467. DOI: 10.1103/PhysRevA.52.3457.

[2] A. Bärtschi and S. Eidenbenz. 2019. Deterministic Preparation of Dicke States. Fundamentals of Computation Theory, 126-139. DOI: 10.1007/978-3-030-25027-0_9.

[3] K. Chakraborty, B. Choi, A. Maitra and S. Maitra. 2014. Efficient quantum algorithms to construct arbitrary Dicke states. Quantum Inf Process 13, 2049–2069. DOI: 10.1007/s11128-014-0797-8.

[4] A. M. Childs, E. Farhi, J. Goldstone, and S. Gutmann. 2002. Finding cliques by quantum adiabatic evolution. Quantum Information & Computation, 2(3):181–191, Apr 2002. DOI: 10.26421/QIC2.3.

[5] D. Cruz, R. Fournier, F. Gremion, A. Jeannerot, K. Komagata, T. Tosic, J. Thiesbrummel, C.L. Chan, N. Macris,M.-A. Dupertuis and C. Javerzac-Galy. 2019. Efficient Quantum Algorithms for GHZ and W States, and Implementation on the IBM Quantum Computer. Adv. Quantum Technol., 2: 1900015. DOI: 10.1002/qute.201900015.

[6] G. Song and A. Klappenecker. 2003. Optimal realizations of controlled unitary gates. Quantum Info. Comput. 3, 2 (March 2003), 139–156.

[7] E. Knill. 1995. Approximation by Quantum Circuits.

[8] B. Langenberg, H. Pham and R. Steinwandt. 2020. Reducing the Cost of Implementing the Advanced Encryption Standard as a Quantum Circuit. IEEE Transactions on Quantum Engineering, vol. 1, pp. 1-12, 2020, 2500112. DOI: 10.1109/TQE.2020.2965697.

[9] M. Mosca and P. Kaye. 2001. Quantum Networks for Generating Arbitrary Quantum States. Optical Fiber Communication Conference and International Conference on Quantum Information ICQI, page PB28, Jun 2001. DOI: 10.1364/ICQI.2001.PB28.

[10] M. Möttönen, J.J Vartiainen, V. Bergholm, and M.M Salomaa. 2004. Quantum Circuits for General Multiqubit Gates. Phys. Rev. Lett, 93, 130502. DOI: 10.1103/PhysRevLett.93.130502.

[11] Martin Plesch and Časlav Brukner. 2011. Quantum-state preparation with universal gate decompositions. Phys. Rev. A 83, 032302. DOI: 10.1103/PhysRevA.83.032302.

[12] V. V. Shende, S. S. Bullock and I. L. Markov. 2006. Synthesis of quantum-logic circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 6, pp. 1000-1010. DOI: 10.1109/TCAD.2005.855930.

[13] G. Vidal and C. M. Dawson. 2004. Universal quantum circuit for two-qubit transformations with three controlled-NOT gates. Phys. Rev. A 69, 010301(R). DOI: 10.1103/PhysRevA.69.010301.

[14] E. Wilson, S. Singh and F. Mueller. 2020. Just-in-time quantum circuit transpilation reduces noise. In proceedings of IEEE International Conference on Quantum Computing & Engineering (QCE20).

[15] A. Zulehner,A. Paler, and R. Wille. 2019. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 38, 1226-1236. DOI: 10.1109/TCAD.2018.2846658.

[16] IBM Q Experience Website, `https://quantum-computing.ibm.com`

1.7 Code for $\mathcal{C}_{4,2}$

```
OPENQASM 2.0;
include "qelib1.inc";
```

```
qreg q[4];

x q[2];
x q[3];
cx q[2],q[3];
cx q[3],q[2];
ry(-pi/3) q[2];
cx q[3],q[2];
ry(pi/3) q[2];
cx q[2],q[1];
cx q[1],q[3];
ry(-pi/8) q[1];
cx q[3],q[1];
ry(pi/8) q[1];
cx q[2],q[1];
ry(-pi/8) q[1];
cx q[3],q[1];
ry(pi/8) q[1];
cx q[1],q[3];
cx q[1],q[2];
cx q[2],q[1];
ry(-pi/3.2885) q[1];
cx q[2],q[1];
ry(pi/3.2885) q[1];
cx q[1],q[0];
cx q[0],q[2];
ry(-pi/10.208) q[0];
cx q[2],q[0];
ry(pi/10.208) q[0];
cx q[1],q[0];
ry(-pi/10.208) q[0];
cx q[2],q[0];
ry(pi/10.208) q[0];
cx q[0],q[2];
cx q[0],q[1];
cx q[1],q[0];
ry(-pi/4) q[0];
cx q[1],q[0];
```

```
ry(pi/4) q[0];
cx q[0],q[1];
```

Listing 1.3: The preparation circuit for $|D_2^4\rangle$

1.8 Code for $\widehat{\mathcal{C}}_{4,2}$

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[4];

x q[2];
x q[3];
ry(pi/2) q[1];
cx q[1],q[3];
cx q[1],q[2];
ry((pi/2)-pi/3.2885) q[1];
cx q[2],q[1];
ry(-((pi/2)-pi/3.2885)) q[1];
cx q[1],q[0];
cx q[0],q[2];
ry(-pi/10.208) q[0];
cx q[2],q[0];
ry(pi/10.208) q[0];
cx q[1],q[0];
ry(-pi/10.208) q[0];
cx q[2],q[0];
ry(pi/10.208) q[0];
cx q[0],q[2];
cx q[0],q[1];
ry((pi/2)-pi/4) q[0];
cx q[1],q[0];
ry(-((pi/2)-pi/4)) q[0];
cx q[0],q[1];
```

Listing 1.4: The post modification preparation circuit for $|D_2^4\rangle$

Part II

The Query Complexity Model

Chapter 1

Background and Organization

Ever since the inception of quantum computation, one of the central questions has been that of understanding its quantitative difference from classical computation. Quantum computation is driven by qubits and has mysterious properties like superposition and entanglement, properties with seemingly no classical counterpart. However, despite these striking advantages, any asymptotic comparison between the power of quantum and classical computation in the general framework has been elusive. One can arguably say that the interest in quantum computation began to grow in the computer science community with the Shor's algorithm and Simon's algorithm in the late 1990's. Shor's algorithm showed that the factoring problem can be solved in a quantum computer in only polynomial (in number of bits needed to express the number to be factored) time, where as the best known classical algorithm has exponential time requirement. Although this algorithm implied that if large scale quantum computers could break the famous RSA encryption, one of the most commercially used public encryption models, it did not have any impact in separating the general power of classical and quantum computation. This is because although there is no polynomial time algorithm for the factoring problem, it has not been possible yet to prove that there can never be one.

Now, both the Shor's algorithm and Simon's algorithm are developed in a particular model, known as the query model. We describe this model briefly using Simon's algorithm. There are two different kinds of query model, and although the model we concentrate on is different from the one used in the aforementioned algorithm, it is useful to have an idea about both, since the underlying idea is the same. First we define the period of a function. A function f with a domain X is said to have a period α if for all $x \in X$ we have $f(x) = f(x + \alpha)$. The problem of interest for Simon's algorithm is to find the period of a function given "black box access" to it. This "black box access", popularly called oracle access is the fundamental characteristics and restriction of the query model. In this model the an oracle U_f (which can viewed as a quantum operator) corresponding to a function f is given, so that for any $x \in X$ $U_f |x\rangle = (-1)^{f(x)} |x\rangle$ where $|x\rangle$ is a quantum state defined so that there is a known bijection from x to $|x\rangle$. Here the restriction is that the underlying functioning of U_f is not revealed. In the classical model similarly we could have a circuit $\hat{U}_f : \hat{U}_f(x) = f(x)$ to which we have black box access. In this model one can show that Simon's algorithm is more efficient than any possible classical algorithm for finding hidden period of functions. The different asymptotic separation known between quantum and classical

computation are in different variations of the query model. The model we discussed here is the most popular model, which we shall refer to as the U_f model. In the rest of this chapter and the coming chapters we discuss a different model, which we call the O_x model.

1.1 The O_x Query Model

The O_x query model can be defined for any kinds of functions. We are only interested in Boolean functions, both for their simplicity and the deep interest in them in different areas of mathematics and computer science, including but not restricted to combinatorics, coding theory and cryptography. From hereon we refer to Boolean functions simply as functions. The problem we are interested in O_x query model is as follows.

We are given the complete description of a function on n variables $f(x_1, \ldots x_n) : \{0,1\}^n \rightarrow \{0,1\}$. Now, given this information we are to evaluate the function f for any given input with the restriction that the inputs to the function can only be accessed by making queries to an oracle O_x. Before moving forward let us first formally describe this oracle in both classical and the quantum paradigm.

Classical and Quantum Oracle: In the query complexity model, the value of any variable can only be queried using an oracle. An oracle is a black-box which can perform a particular computation. In the classical model, an oracle accepts an input i $(1 \leq i \leq n)$ and output the value of the variable x_i. In the quantum model, the oracle needs to be reversible. It is represented as an unitary O_x which functions as follows.

$$O_x \left|i\right\rangle \left|\phi\right\rangle = \left|i\right\rangle \left|\phi \oplus x_i\right\rangle, \ 1 \leq i \leq n$$

Figure 1.1 represents the working of an oracle in the quantum complexity model.

In this framework, we are only interested in the minimum number of queries one would need to make to this oracle in the worst case scenario, which is called the query complexity of the function.

Definition 1. *The query complexity of a function is the maximum number of times this oracle needs to be used to evaluate the value of the function f for any value of the variables $x_1, x_2, \ldots, x_n$*

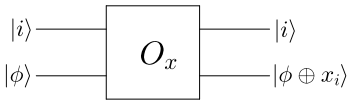

Figure 1.1: Working of a quantum oracle

in a given model.

The query complexity of a function can be defined in both deterministic and probabilistic classical models as well as exact and bounded error quantum models. The main advantage of quantum models over the classical model is that we can query the superposition of variables in the same query, which allows us to possibly extract more "global" information about the variables which could be intelligently used to reduce the query requirement to evaluate a function. Here we will be interested in the deterministic classical and the exact quantum query models, and the comparison of how efficiently different functions can be evaluated in these two models. Before proceeding, it is important to underline the motivation behind the work and results to follow. Like we discussed earlier the main driving force behind research in the query complexity model is understanding the difference in power of quantum and classical computation in a restricted model. Another very interesting problem is that of understanding how the structure of different Boolean functions can effect the difference in the query complexity for it between the classical and quantum query models. For there are functions for which the maximum number of queries are same in both the models, others for which it only differs in constant factors w.r.t n, and for others we have superlinear separation. Exploring and exploiting different Boolean functions properties and structures to observe these scenarios, and as a result laying out new separation results as well as new algorithmic techniques are our main motivations. Let us now formally define the terms deterministic classical and exact quantum query complexities.

Deterministic (Classical) Query Complexity: The minimum number of queries that a function f needs to be evaluated using a deterministic algorithm is called its Deterministic Query Complexity ($D(f)$). We generally omit the word 'classical'. A query based classical deterministic algorithm for evaluating a Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be expressed as a rooted decision tree as follows.

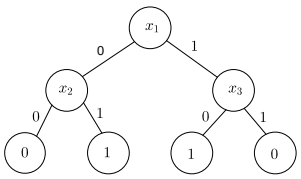

Figure 1.2: Example of a decision tree

In this model, every internal node corresponds to a query to a variable x_i $1 \leq i \leq n$. Each leaf is labeled as either 0 or 1. The tree is traversed from the root of the tree till it reaches a leaf in the following manner. Every internal node has exactly two children and depending on the outcome of the query (0 or 1 respectively), one of the two children are visited (left or right, respectively). That is this is a binary tree. The leaf nodes correspond to the output of f for different inputs. Every decision tree uniquely defines a Boolean function which we can obtain by deriving the Algebraic Normal Form (ANF) from a given tree. This can be obtained as follows. We start from the root node with the following recursion. Let T be the decision tree in question and let it's ANF be $ANF(T)$. Assume the root node is some x_j and T_j^0 and T_j^1 be the subtrees connected to x_j through the edged weighted 0 and 1 respectively. Then the ANF is $ANF(T) = x_j ANF(T_j^1) \oplus (x_j \oplus 1) ANF(T_j^0)$. For example, the ANF of the Boolean function corresponding to the tree shown in Figure 1.2 is $(x_1 \oplus 1)(x_2) \oplus x_1(x_3 \oplus 1) = x_1x_2 \oplus x_1x_3 \oplus x_1 \oplus x_2 \oplus x_3$. When evaluating the function for any input point, we start from the root and if the value of the variable at a node is 0, we traverse down the edge marked 0, and the edge marked 1 otherwise, till we reach a leaf node, which is the output of the function for that particular input.

Corresponding to a function, there can be many Deterministic Query Algorithms that can evaluate it. The depth of a decision tree is defined as the number of edges encountered in the longest root to leaf path. Given f, the shortest depth decision tree representing the function, is called the optimal decision tree of f and the corresponding depth is termed as the Deterministic classical complexity of f, denoted as $D(f)$.

Exact Quantum Query Complexity: A Quantum Query Algorithm is defined using a start state $|\psi_{start}\rangle$ and a series of unitary Transformations

$$U_0, O_x, U_1, O_x, \ldots, U_{t-1}, O_x, U_t,$$

where the unitary operations U_j are indifferent of the values of the variables x_i and O_x is the oracle as defined above. Therefore, the final state of the algorithm is

$$|\psi_{final}\rangle = U_t O_x U_{t-1} \dots U_1 O_x U_0 |\psi_{start}\rangle$$

and the output is decided by some measurement of the state $|\psi_{final}\rangle$. A quantum algorithm is said to exactly compute f if for all $(x_1, x_2, \dots, x_n)$ it outputs the value of the function correctly with probability 1. The minimum number of queries needed by a Quantum Algorithm to achieve this is called the Exact Quantum Query Complexity $Q_E(f)$ of the function. We also define the term separable as we often use it.

Definition 2. *A Boolean function f is called separable if $Q_E(f) < D(f)$ and non-separable otherwise.*

Now, it is easy to see that in both the models the maximum query complexity of a function can be n, in any model. One can also show that the least exact quantum query complexity of a non-degenerate function on n variables is $\Omega\left(\sqrt{\log n}\right)$. Let us now move to query friendly functions.

1.2 Outline

In chapter 2 we explore functions who have the lowest possible deterministic classical query complexity of all functions on n variables. We term these as "query friendly functions". With respect to these functions we have a very interesting observation, around which the chapter revolves. For certain values of n, all query friendly functions on n variables are non-separable. For certain other values of n we show that they are separable. For all other values of n we show that they cannot be separated using the arguably most popular algorithmic technique of the exact quantum query model, known as the parity decision tree technique.

In chapter 3 we aim to study separation between $D(f)$ and $Q_E(f)$ for non-symmetric functions using the properties of the algebraic normal form of certain Boolean functions. Our motivation for this chapter stems from the fact that there are not many techniques known for obtaining efficient exact quantum query algorithms and most functions that have been studied so far symmetric in

nature. In this regard we design a new algorithmic technique that gives us separation for a class of functions with $\Omega\left(2^{\sqrt{n}}\right)$ functions built using direct sum constructions.

Finally In Chapter 4 we study the $D(f)$ and $Q_E(f)$ of the Maiorana-Mcfarland (MM) type bent functions. This is a class of size super-exponential in n, with more than $2^{2^{\frac{n}{4}}}$ functions for any n. The motivation behind this study is the uniformity in the algebraic normal form of these functions. We obtain that $D(f) = n$ for functions in this class. Then we design a parity decision tree based technique with query complexity of $\lceil \frac{n}{4} \rceil$. We further reduce it to $\lceil \frac{5n}{8} \rceil$ using the techniques we have developed in Chapter 3. We finish the chapter with open questions in this area that we are yet to solve.

Chapter 2

Query Friendly Functions

2.1 Introduction

In this chapter we concentrate on the deterministic and exact quantum query complexity of different Boolean function classes. There are other computational models such as the classical randomized model and the bounded error quantum model [1] and there exists rich literature on work on these models as well. However, those are not in the scope of this work.

In this regard one may note that the work by Barnum et.al [5] can be used to find the exact quantum query complexity of any function on n variables by repetitively solving semi definite programs (SDP). Montanaro et.al [10] have used this method to find exact quantum query complexity of all Boolean functions upto four variables as well as describe a procedure of formulating the quantum algorithm to achieve the said exact quantum query complexity. This method is not yet found to be suitable for finding the exact quantum query complexity of a general classes of Boolean functions. Additionally, the SDP are resource intensive in nature and solving the SDP for large values of n is computationally challenging. But for the cases where the number of variables is low, this does offer an exhaustive view of the exact quantum query complexities of all Boolean functions.

As an example, in a very recent paper Chen et.al [8] have shown that $f(x) = x_i$ or $f(x) = x_{i_1} \oplus x_{i_2}$ are the only Boolean functions with $Q_E(f) = 1$. However the work of Montanaro et.al [10, Section 6.1] show that the Boolean functions f with 2 or lesser variables and $Q_E(f) = 1$ are

- The single variable function x_i.
- The two variable functions $x_{i_1} \oplus x_{i_2}$.

Then it is shown in [10, Section 6.2] that the minimum quantum exact quantum query complexity of any Boolean function with 3 or more influencing variables is 2. This essentially implies that the work of [8] is in fact a direct corollary of [10].

We now lay out of the structure of the rest of the chapter.

2.1.1 Organization & Contribution

In Section 2.2, we start by describing the fact that the maximum number of influencing variables that a function with k deterministic query complexity can have is $(2^k - 1)$. We first construct such a function using the decision tree model. The decision tree representation of such a function is a

k-depth fully-complete binary tree in which every internal node queries a unique variable. We first prove in Theorem 4 that any function with $2^k - 1$ influencing variables and k deterministic query complexity must have the same exact quantum query complexity (k).

Next, we define a special class of Boolean functions in Section 2.2.1, called the "Query Friendly" functions. A function f with n influencing variables is called query friendly if there does not exist any other function with n influencing variables with lesser deterministic query complexity than f. If n lies between 2^{k-1} and $2^k - 1$ (both inclusive) then all functions with deterministic query complexity k are called query friendly functions. The proof in Theorem 4 directly implies that all query friendly functions with $n = 2^k - 1$ influencing variables are non-separable.

Then in Section 2.2.2 we identify a class of non-separable query friendly functions for all values of n. We conclude this section by showing that all query friendly functions with $n = 2^k - 2$ ($k > 2$) influencing variables are non-separable as well.

In Section 2.3, we describe the parity decision tree model. We first discuss the simple result that a k-depth parity decision tree can describe functions with upto $2^{k+1} - 2$ influencing variables. In Section 2.3.1 we define another set of query friendly functions on n influencing variables that exhibit minimum separation (i.e., one) between deterministic and exact quantum query complexity for certain generalized values of n. We prove by construction that if $2^{k-1} \leq n < 2^{k-1} + 2^{k-2}$ then there exists a class of query friendly functions such that for any function f in that class we have $Q_E(f) = D(f) - 1$. We conclude the section by showing that for other values of n there does not exist separable query friendly functions that can be completely described by the parity decision tree model.

We conclude the study in Section 2.4 outlining the future direction of our work. We further state open problems that we have encountered in this work. Solution to these problems will help us better understand the limitations of the parity decision tree model.

2.2 Decision Trees and No-separation results

As we have discussed, query algorithms can be expressed as decision trees in the classical deterministic model. In this regard, let us present the two following simple technical results. These results

are well known in folklore and we present them for completeness.

Lemma 3. *There exists a Boolean function f_k with 2^k-1 influencing variables such that $D(f) \leq k$.*

Proof. We construct this function for any k as follows. We know that if a Boolean function f can be expressed as a decision tree of depth d, then $D(f) \leq d$. We now build a decision tree, which is a fully-complete binary tree of depth k. Each of the internal nodes in this tree is a unique variable, that is, no variable appears in the decision tree more than once. Since there are $2^k - 1$ internal nodes in such a tree, this decision tree represents a Boolean function f_k on $2^k - 1$ variables with $D(f_k) \leq k$.

Without loss of generality we can name the root variable of the corresponding decision tree as x_1 and label the variables from left to right at each level in ascending order. The resultant structure of the tree is shown in Figure 2.1. □

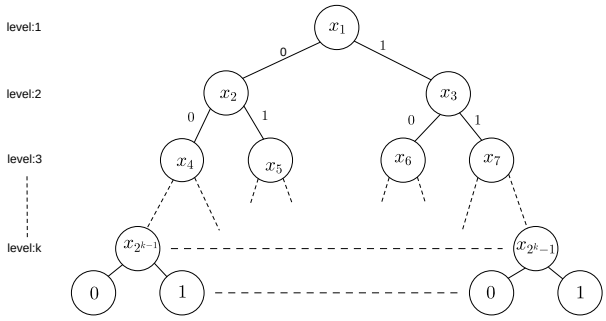

Figure 2.1: Decision Tree corresponding to function f with maximum influencing variables for $D(f) = k$

Having constructed such a Boolean function f_k, we now show that is indeed the function with the maximum number of influencing variables that can be evaluated using the deterministic computational model using k queries.

Lemma 4. *Given any integer k, the maximum number of influencing variables that a Boolean function f has such that $D(f) = k$ is $2^k - 1$.*

Proof. Suppose there exists a Boolean function with $n_1 (> 2^k - 1)$ influencing variables that can be evaluated using k queries. This implies that there exists a corresponding decision tree of depth k that expresses this function. However, in a decision tree corresponding to a Boolean function f, all

the influencing variables should be present as an internal node at least once in the decision tree. Otherwise,

$$f(x_1, x_2, \ldots, x_{i-1}, 0, \ldots, x_n)$$
$$= f(x_1, x_2, \ldots, x_{i-1}, 1, \ldots, x_n) \; \forall x_j \in \{0,1\} : j \neq i,$$

which implies that x_i is not an influencing variable of the function. Since there cannot exist a decision tree of depth k that has more than $2^k - 1$ internal nodes, such a function can not exist.

This implies that for any function f with $n = 2^k - 1$ influencing variables and $D(f) = k$, the corresponding decision tree is a k-depth complete tree where every variable is queried only once. □

It immediately follows that a function f with $n = 2^k - 1$ influencing variables has deterministic query complexity $D(f) \geq k$.

Theorem 4. *Given any Boolean function f with $2^k - 1$ influencing variables and $D(f) = k$ we have $Q_E(f) = k$.*

Proof. This is proven by showing that any function f characterized as above is at least as hard to evaluate as the function AND_k, which is AND of k variables.

Given such a function f, there exists a corresponding k-depth complete tree T_f. As we have shown in Lemma 4, in such a tree all internal nodes will query a variable and all the variables will appear in the tree exactly once.

Given the decision tree T_f corresponding to f let $x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_k}$ be a root to internal node path in the tree so that children of x_{i_k} are the leaf nodes. Here

$$val(x_{i_t}, 1) = x_{i_{t+1}}, \; 1 \leq t \leq k-1.$$

We call this set of variables s_{max}. We fix the values of the variables $\{x_1, x_2, x_3, \ldots, x_{2^k-1}\} \setminus s_{max}$ as follows. Each of the variables at a level less than or equal to $k-1$ is assigned either 0 or 1. Now either $val(x_{i_k}, 0) = 0$ and $val(x_{i_k}, 0) = 1$ or $val(x_{i_k}, 0) = 1$ and $val(x_{i_k}, 0) = 0$.

- In the first case, If a variable is at the k-th level, i.e., its children are the leaf nodes then each such variable y_i is fixed at the value c_i such that $val(y_i, c_i) = 0$. Then the function is reduced

to $\prod_{t=1}^{k} x_{i_k}$.

- In the second case, the values of variables y_i in the k-th level is fixed at the value e_i so that $val(y_i, c_i) = 1$. Then the function is reduced to $\left(\prod_{t=1}^{k} x_{i_k}\right) \oplus 1$.

The reduced function is AND_k in the first case and OR_k in the second case. In both the cases we have $Q_E(f) \geq k$, as $Q_E(AND_k) = Q_E(OR_k) = k$ [6, Table 1]. We also know that $Q_E(f) \leq D(f)$ for any Boolean function f and therefore $Q_E(f) \leq k$. Combining the two we get $Q_E(f) = k$. □

We reiterate the idea behind the proof to further simplify the argument. Reducing a function to AND_k essentially implies that there exists a set of variables $x_1, x_2, \ldots x_k$, such that if they are not all equal to 1, then the function outputs 0. In terms of the tree the implication is as follows. Let the path in the proof of Theorem 4 be $x_{i_1}, x_{i_2}, \ldots x_{i_l k}$ such that the function is reduced to AND_k by fixing values of the other variables. Then while the decision tree is traversed from the root, if any of these k variable's value is 0, we move to a node that is out of the path, and then the value of the other internal nodes should be so fixed that we always reach a 0-valued leaf node.

2.2.1 Query Friendly Functions

Having established these results, we characterize a special class of Boolean functions. Given any n, We call the Boolean functions with n influencing variables that have minimum deterministic query complexity as the query friendly functions on n variables. We denote the corresponding query complexity of this class of functions as DQ_n, and its value is calculated as follows.

Lemma 5. *The value of DQ_n is equal to $\lceil \log(n+1) \rceil$.*

Proof. We consider any n such that $2^{k-1} - 1 < n \leq 2^k - 1$. We have shown in lemma 4 that there cannot exist a Boolean function with n variables that can be evaluated with $k-1$ classical queries.

Since the maximum number of influencing variables that a Boolean function with k query complexity has is $2^k - 1$ as proven above, there exists a Boolean function with n variables with $D(f) = k$. Now $\lceil \log(n+1) \rceil = k$, which concludes the proof. □

Corollary 1. *For $n = 2^k - 1$, there does not exist any separable query friendly functions.*

Proof. For $n = 2^k - 1$, we have $DQ_n = k$. We have shown in Theorem 4 that any function f with $2^k - 1$ influencing variables and $D(f) = k$ has $Q_E(f) = k$. □

Now let us provide some examples of such functions where the deterministic classical and exact quantum query complexities are equal.

- $k = 2, n = 2^k - 1 = 3, Q_E(f) = D(f) = 2$: the function is $f = (x_1 \oplus 1)x_2 \oplus x_1x_3 = x_1x_2 \oplus x_1x_3 \oplus x_2$.
- $k = 3, n = 2^k - 1 = 7, Q_E(f) = D(f) = 3$: the function is $f = (x_1 \oplus 1)((x_2 \oplus 1)x_4 \oplus x_2x_5) \oplus x_1((x_3 \oplus 1)x_6 \oplus x_3x_7) = x_1x_2x_4 \oplus x_1x_2x_5 \oplus x_1x_4 \oplus x_2x_4 \oplus x_2x_5 \oplus x_4 \oplus x_1x_3x_6 \oplus x_1x_3x_7 \oplus x_1x_6$.

Next we move to a generalization when $n \neq 2^k - 1$.

2.2.2 Extending the result for $n \neq 2^k - 1$

We first identify a generic set of non-separable query friendly functions where $2^{k-1} - 1 < n < 2^k - 1$ and then show that no query friendly function on $n = 2^k - 2, k > 2$ influencing variables are separable. We define such a set of non-separable query friendly functions for $2^{k-1} - 1 < n < 2^k - 1$ using the decision tree model again. We construct a decision tree of depth k such that the first $k - 1$ levels are completely filled and every variable occurs exactly once in the decision tree. That implies there are $n - 2^{k-1} + 1$ nodes in the k-th level. Let us denote the corresponding function as $f_{(n,1)}$.

Theorem 5. *The Boolean function $f_{(n,1)}$ on n influencing variables has $D(f_{(n,1)}) = Q_E(f_{(n,1)})$.*

Proof. This k-depth decision tree constructed for any n such that $2^{k-1} - 1 < n < 2^k - 1$ has the following properties.

- The corresponding function has deterministic query complexity equal to k. This is because the number of influencing variables in the function is more than the number of variables that a Boolean function with deterministic query complexity $k - 1$ can have.
- There is at least one internal node at k-th level. let that node be called x_{i_k}. Let the root to x_{i_k} path be $x_1, x_{i_2}, x_{i_3}, \ldots, x_{i_{k-1}}, x_{i_k}$ such that $val(x_1, d_1) = x_{i_2}, val(x_{i_2}, d_2) = x_{i_3}$ and so on. Applying the reduction used in Theorem 4 the corresponding Boolean function can be

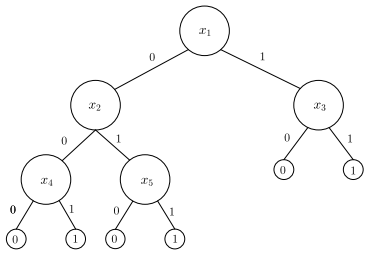

Figure 2.2: Decision Tree corresponding to $f_{(5,1)}$

reduced to the function $(x_1 \oplus \overline{d_1})(x_{i_2} \oplus \overline{d_2}) \ldots (x_{i_k} \oplus \overline{d_k})$ which is isomorphic to AND_k. (Note that $\overline{d_i} = 1 \oplus d_i$, i.e., the complement of d_i.) This implies that $Q_E(f_{n,1}) \geq k$. We also know $D(f_{n,1}) = k$, and therefore the exact quantum query complexity of the function is k. Figure 2.2 gives an example of a function in $f_{(5,1)}$.

□

The result is thus a generalization when $2^{k-1} - 1 < n < 2^k - 1$, in identifying a class of functions where the separation between classical and quantum domain is not possible.

We now show that in fact for $k > 2$, all query friendly functions with $2^k - 2$ variables are non-separable.

Theorem 6. *Let f be a query friendly function on $n = 2^k - 2$ variables, such that $k > 2$. Then $D(f) = Q_E(f)$.*

Proof. There exists a decision tree T_f of depth-k that evaluates f. Since f has $2^k - 2$ variables then T_f can be of the following forms:

1. T_f has $2^k - 2$ internal nodes and each of the nodes query a unique variable. Each tree of this type corresponds to a function of the type $f_{(n,1)}$ and therefore is non-separable.

2. T_f has $2^k - 1$ internal nodes and there exists two nodes in the tree which query the same variable.

We analyze the different structures of T_f corresponding to the second case. Let the root node queries a variable x_1. Then the following cases can occur.

Case 1: Both the children of x_1 query the same variable x_2:
Let the two nodes be represented by x_2^0 and x_2^1. We choose a k-depth path $x_1, x_2^1, x_3, \ldots x_k$ such

that $val(x_i, 1) = x_{i+1}, 1 \leq i \leq k-1$. Let us assume for simplicity $val(x_k, 0) = 0$ and $val(x_k, 1) = 1$. For all vertices x_t on the $k-th$ level such that $x_t \neq x_k$ we fix the value of the variable to d_t such that $val(x_t, d_t) = 0$. This construction reduces the function f to the AND_k function, implying $Q_E(f) \geq k$. As we know $D(f) = k$, this implies $Q_E(f) = k$.

Case 2: At most one of the children of x_1 query a variable that appears more than once in the decision tree:
In this case there exists a k-depth path consisting of nodes querying $x_1, x_2, \ldots x_k$ such that each of these variables appear only once in the tree such that

$$val(x_i, d_i) = x_{i+1}, 1 \leq i \leq k-1 \text{ and } val(x_k, d_k) = 1$$

.

Now let the variable that is queried twice be x_{dup} and the nodes querying the variable be denoted as x^1_{dup} and x^2_{dup}. If at most one of these nodes is in the k-th level then we can simply follow the method of the first case to reduce the function into $\prod_{i=1}^{k}(x_i \oplus \bar{d_i})$.

If both the node querying x_{dup} are in the k-th level, then at least one of their parent nodes do not belong to the set $\{x_1, x_2 \ldots x_k\}$. Let the variable being queried by that node be x_{par} and it is parent of at-least x^1_{dup}. We fix the value of x_{par} to be c such that $val(x_{par}, \bar{c}) = x^1_{dup}$. Now we again fix all the value of the variables x_t on the k-th level except x^1_{dup} and x_k in the same way as in case 1 to reduce the function to $\prod_{i=1}^{k}(x_i \oplus \bar{d_i})$.

The function $\prod_{i=1}^{k}(x_i \oplus \bar{d_i})$ is isomorphic to the AND_k function and thus the proof is completed.

□

2.3 Parity Decision Trees and Separation results

We now explore the parity decision tree model introduced in [10]. This model is constructed using the fact that in the exact quantum query model, the value of $x_{i_1} \oplus x_{i_2}$ can be evaluated using a single query.

A parity decision tree is similar to a deterministic decision tree. But while in a decision tree a

query can only return the value of a variable x_i, in a parity decision tree a query can return either the value of a variable x_i or the parity of two variables $x_{i_1} \oplus x_{i_2}$. A parity decision tree represents a quantum algorithm in which the oracle is queried values of type x_{i_1} and $x_{i_1} \oplus x_{i_2}$. In fact in this case the work qubits can be measured after each query and reset to a default state.

Let f be a Boolean function that can be expressed as a k-depth decision tree in which every internal node either queries a variable x_i or the parity of two variables, $x_{i_1} \oplus x_{i_2}$. We can then say that $Q_E(f) \leq k$. Figure 2.3 gives an example of a parity decision tree.

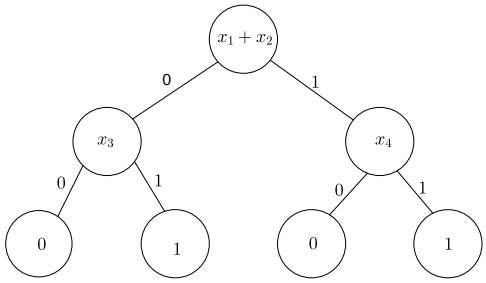

Figure 2.3: Example of a parity decision tree

The corresponding Boolean function is $(x_1 \oplus x_2)x_4 \oplus (x_1 \oplus x_2 \oplus 1)x_3$, with deterministic query complexity 3 and exact quantum query complexity 2.

A k-depth parity decision tree can only evaluate a function of algebraic degree less than or equal to k, whereas there may exist a Boolean function of degree higher than k that can be evaluated using k queries. Thus, although this model does not completely capture the power of the quantum query model, we use the generalized structure of this model to find separable query friendly functions for certain values of n.

We say a parity decision tree T completely describes a Boolean function f if T is a parity decision tree with the minimum depth (say $depth_f$) among all parity decision trees that represent f and $Q_E(f)$ is equal to $depth_f$.

Lemma 6. *Given any k there exists a Boolean function f with $2^{k+1}-2$ variables such that $Q_E(f) = k$.*

Proof. This proof follows directly from the definition of parity decision trees and the proof of existence of a Boolean function with $2^k - 1$ variables with $D(f) = k$. Again we construct a k depth complete parity decision tree such that every internal node is a query of the form $x_{i_1} \oplus x_{i_2}$ such

that no variable appears twice in the tree. This tree represents a Boolean function f of $2(2^k - 1)$ variables and inherently $Q_E(f) \leq k$. This function can also be reduced to the AND_k function which implies $Q_E(f) \geq f$. This implies $Q_E(f) = k$. We skip the proof of reduction to avoid repetition.

This is also the maximum number of influencing variables that a function f can have so that $Q_E(f) = k$ and f can be completely described using parity decision trees. This can be proven in the same way as in lemma 4 and we do not repeat it for brevity. □

We now prove some observations related to separability for a broader class of functions and then explore separability in query friendly functions.

Theorem 7. *If $n \neq 2^k - 1$ for any k, then there exists a Boolean function for which $Q_E(f) < DQ_n$.*

Proof. Let $2^{k-1} - 1 < n < 2^k - 1$ for some natural number k. In this case $DQ_n = k$. However, there exist Boolean functions f_Q with n influencing variables such that $Q_E(f_Q) = k - 1$. We define a generic class of such functions using parity decision trees. Let $n = 2^{k-1} - 1 + y$. Then we can always construct a complete parity decision tree of depth $k - 1$ with the following constraints:

- Every variable appears only once in the tree.
- y internal nodes have query of the form $x_{i_1} \oplus x_{i_2}$. The rest of the internal nodes query the value of a single variable.

Since $y \leq 2^{k-1} - 1$, which is the number of internal nodes in a complete parity decision tree of depth $k - 1$, such a function always exists. □

However, if $n = 2^k - 1$ for some k, then there not does not exist any Boolean function f that can be completely expressed using the parity decision trees such that $Q_E(f) < DQ_n$. If $n = 2^k - 1$ then $DQ_n = k$ as well and there does not exist any Boolean function f_Q with n variables that can be expressed using parity trees and has $Q_E(f_Q) \leq k - 1$. This is true as we have already obtained that the Boolean function with maximum number of influencing variables and depth $k - 1$, that can be expressed using parity decision tree is $2^k - 2$ (putting $k - 1$ in place of k in Lemma 6 above).

Moreover, there does not exist any Boolean function with 3 influencing variable such that exact query complexity is less than DQ_3, which is equal to 2. It is interesting to note that if for some

$n = 2^k - 1$ there exists a Boolean function with $Q_E(f) = k - 1$ then there exists separation for all $n = 2^j - 1 : j > k$. This can be easily proven with induction.

Lemma 7. *If there exists a function f_k with $2^k - 1$ influencing variables such that $Q_E(f) = k - 1$, then there exists a function f_j with $2^j - 1$ influencing variables such that $Q_E(f) \leq j - 1$ for all $j > k$.*

Proof. If there exists a function f_k with the specified property then f_{k+1} can be constructed as follows:

$$f_{k+1} = x_{2^{k+1}-1}(f_k(x_1, x_2, \ldots x_{2^k-1}) \oplus (x_{2^{k+1}-1} \oplus 1) f_k(x_{2^k}, x_{2^k+1}, \ldots x_{2^{k+1}-2}).$$

It is easy to see $Q_E(f_{k+1}) \leq k$. Using this construction recursively yields a desired function for any $j > k$. □

We complete the categorization by defining a generalized subclass of Query friendly Boolean functions. We define this subclass such that a function f, belonging to this, has $Q_E(f) = DQ_n - 1$.

2.3.1 Separable Query Friendly functions

We construct a generic function for this set of query friendly functions using parity decision trees for values of n such that there exists $k, 2^{k-1} - 1 < n \leq 2^{k-1} + 2^{k-2} - 1$. We first describe the construction using a parity decision tree and then prove the query complexity values of the function.

Let us construct a parity decision tree of depth $k - 1$ in the following manner. The first $k - 2$ levels are completely filled, with each internal node querying a single variable. All variable appears exactly once in this tree. Let these variables be termed $x_1, x_2, \ldots, x_{2^{k-2}-1}$. In the $(k - 1)$-th level, there are $\lceil \frac{n-(2^{k-2}-1)}{2} \rceil$ internal nodes, with each query being of the form $x_{i_1} \oplus x_{i_2}$. (In case $n - 2^{k-2} + 1$ is odd, there is one node querying a single variable). Then if $n = 2^{k-1}$ there are $2^{k-3} + 1$ internal nodes in $(k - 1)$-th level and if n=$2^{k-1} + 2^{k-2} - 1$ there are 2^{k-2} nodes in the $(k - 1)$-th level, resulting in a fully-complete binary tree of depth $k - 1$. We denote this generic function as $f_{(n,2)}$.

Theorem 8. *The Boolean function $f_{(n,2)}$ on n influencing variables has $D(f) = DQ_n$ and $Q_E(f) = DQ_n - 1$.*

Proof. If $2^{k-1} - 1 < n \leq 2^{k-1} + 2^{k-2} - 1$ then $DQ_n = k$. We first prove that $Q_E(f_{(n,2)}) = k - 1$. Since there exists a parity decision tree of depth $k - 1$,

$$Q_E(f_{(n,2)}) \leq k - 1. \tag{2.1}$$

If we fix one of the variables of each query of type $x_{i_1} \oplus x_{i_2}$ to zero then the reduced tree corresponds to a non-separable function shown in 2.2.2 of depth $k - 1$,that is the function can be reduced to AND_{k-1}. This implies

$$Q_E(f_{(n,2)}) \geq k - 1. \tag{2.2}$$

Combining (2.1) and (2.2) we get $Q_E(f_{(n,2)}) = k - 1$.

Now we show that $D(f_{(n,2)}) = k$ by converting the parity decision tree to a deterministic decision tree of depth k. All the internal nodes of the parity decision tree from level 1 to level $k - 2$ queries a single variable. The nodes in the $k - 1$-th level have queries of the form $x_{i_1} \oplus x_{i_2}$. Each such node can be replaced by a deterministic tree of of depth 2 in the following way. Suppose there is a internal node $x_{i_1} \oplus x_{i_2}$ in the $(k - 1)$-th level.

We replace this node with a tree, whose root is x_{i_1}. Both the children of the node queries x_{i_2} and the leaf node values are swapped in the two subtrees. Without loss of generality, suppose in the original tree $val(x_{i_1} \oplus x_{i_2}, 0) = 0$ and $val(x_{i_1} \oplus x_{i_2}, 1) = 1$ Then in the root node $val(val(x_1, 0), 0) = 0$ and $val(val(x_1, 1), 0) = 1$ and so on. Figure 2.4 gives a pictorial representation of the transformation. The resultant deterministic decision tree is of depth k as there is at least 2^{k-3} node in the $k - 1$-th level in the parity decision tree which goes through transformation. This implies $D(f_{(n,2)}) \leq k$. We also know that in this case $DQ_n = k$. Combining the two results we get $D(f_{(n,2)}) = k$. □

Remark 1. *It should be noted that although we use a particular function f for any n to show the separation for $Q_E(f)$ and $D(f)$, this immediately means that this separation is established for at least the class of functions on n influencing variables that are PNP equivalent to f.*

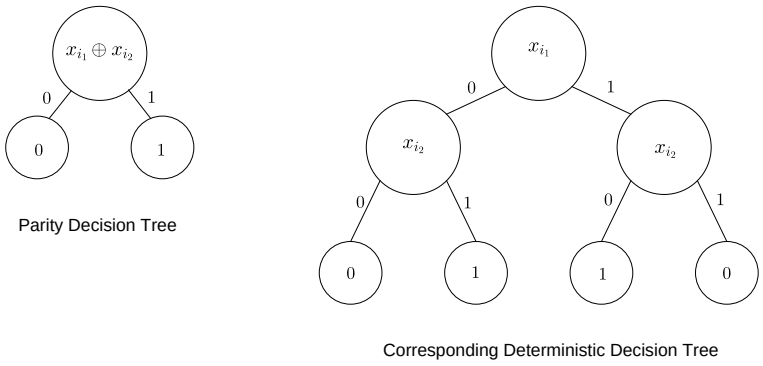

Figure 2.4: Conversion of a node in the parity decision tree to a deterministic decision tree

Let us now consider a function of the form $f_{(5,2)}$ described by its ANF as below:

$$\begin{aligned} f &= (x_1 \oplus 1)(x_2 \oplus x_3) \oplus x_1(x_4 \oplus x_5) \\ &= x_1x_2 \oplus x_1x_3 \oplus x_1x_4 \oplus x_1x_5 \oplus x_2 \oplus x_3. \end{aligned}$$

This provides an example for $n = 5, D(f) = 3$, and $Q_E(f) = 2$. In Figure 2.5 we present the decision tree for this function and the corresponding quantum circuit is provided in Figure 2.6.

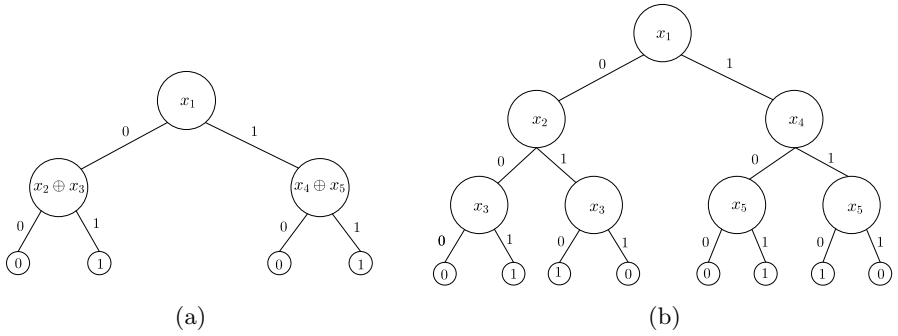

Figure 2.5: Parity Decision (a) and Deterministic (b) Tree corresponding to $f_{(5,2)}$

We now explain for the sake of completeness the difference in working of the exact quantum and deterministic algorithm for this function.

Suppose we want to evaluate this function at the point $(1, 0, 1, 0, 1)$. The deterministic algorithm will first query x_1, and getting its value as 1 it will then query x_4. Since x_4 is 0 it will query the

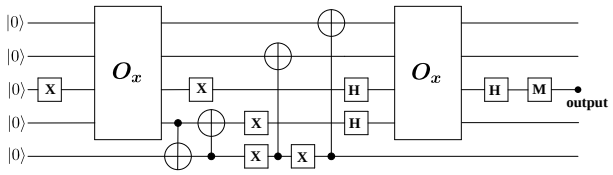

Figure 2.6: Quantum algorithm responding to $f_{(5,2)}$

x_5 node which is it's left children and then output 1 as x_5 is 1.

The quantum algorithm will evaluate as follows.

1. Here $\psi_{start} = |0\rangle |0\rangle |0\rangle |0\rangle |0\rangle$.
2. The first X gate transforms it into $|1\rangle_2 |0\rangle |0\rangle$

   Here $|i\rangle_2$ implies $|a\rangle |b\rangle |c\rangle$ where abc is the binary representation of integer i.
3. Then we get $O_x(|1\rangle_2 |0\rangle |0\rangle) = |1\rangle_2 |x_1\rangle |0\rangle = |1\rangle_2 |1\rangle |0\rangle$.
4. The CNOT gates, the not gate and the Hadamard gates ($\mathsf{H^3}$ and $\mathsf{H^4}$) transform the state into $(\frac{|4\rangle_2+|5\rangle_2}{\sqrt{2}}) |-\rangle |0\rangle$ where $|-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$.
5. Now

$$O_x(\frac{|4\rangle_2 + |5\rangle_2}{\sqrt{2}}) |-\rangle |1\rangle = \\ (\frac{(-1)^{x_4} |4\rangle_2 + (-1)^{x_5} |5\rangle_2}{\sqrt{2}}) |-\rangle |1\rangle .$$

   Let this state be $|\phi\rangle$.
6. $\mathsf{H^3} |\phi\rangle = \frac{1}{2}((-1)^{x_4} + (-1)^{x_5}) |4\rangle_2 + ((-1)^{x_4} - (-1)^{x_5}) |5\rangle_2) |-\rangle |1\rangle$
7. since $x_4 = 0$ and $x_5 = 1$ we get $|5\rangle_2 |-\rangle |1\rangle$ which is equal to $|1\rangle |0\rangle |1\rangle |-\rangle |1\rangle$. Measuring the third qubit in computational basis we get the desired output, 1.

Here please note that that only of the 3rd and 4th CNOT gate will work for any given run of the circuit. If $x_1 = 1$ we need to evaluate only $x_4 \oplus x_5$ and the 4th CNOT gate has $|1\rangle$ in its control. However if $x_1 = 0$ then we would have to output $x_2 \oplus x_3$ in which case the 3rd CNOT gate would have been activated and the 4th CNOT gate will not be doing anything. Since the circuit needs to evalaute for all possible inputs, both the CNOT gates are present. This completes the example of separation.

Finally, we conclude this section by proving that our construction of separable query friendly function indeed finds such examples for all cases where a parity decision tree can compute such a function. This completes the characterization using parity decision trees.

Theorem 9. *If $2^{k-1} + 2^{k-2} - 1 < n \leq 2^k - 1$, there does not exist any separable query friendly function that can be completely described using parity decision trees.*

Proof. Let f_n be a query friendly function on $2^{k-1} + 2^{k-2} + 1 < n \leq 2^k - 1$ influencing variables. In this case $DQ_n = k$, and hence $D(f_n) = k$. Therefore there exists a corresponding k-depth decision tree T_f. As we know there are at most $2^k - 1$ internal nodes in such a tree and at least $2^{k-1} + 2^{k-2}$ variables that needs to be queried at least once. Therefore there can be at most $2^{k-2} - 1$ internal nodes which query variables that appear more than once in the tree.

This implies that there exists a node in the k-th level querying a variable $x_{i_k^0}$ such that it appears only once in the decision tree. We consider the root(x_{i_1}) to $x_{i_k^0}$ path. It is to be noted that the root variable needs to be queried only once in any optimal tree. Let us also assume for simplicity that $val(x_{i_k{}^0}, 0) = 0$ and

$$val(x_{i_t}, d_t) = x_{i_{t+1}},\ 1 \leq t \leq k-2$$
$$val(x_{i_{k-1}}, d_{k-1}) = x_{i_k^0}$$

Let us now define the following sets of variables:

$$W_j \subseteq \{x_1, x_2, \ldots x_n\}$$
$$X_j = W_j \cup \{x_{i_k^0}\}$$
$$Y_j \subseteq (\{x_1, x_2, \ldots x_n\} \setminus \{x_{i_k^0}\})$$
$$\text{where } 1 \leq j \leq k$$

Let g_j and $h_j, 1 \leq j \leq k$ be functions with influencing variables belonging from the sets X_j, Y_j respectively. Then the ANF of f_n can be described as:

$$f_n = (x_{i_1} \oplus \overline{d_1}) g_1(X_1) \oplus (x_{i_1} \oplus d_1) h_1(Y_1)$$

This is because the variable $x_{i_k^0}$ can influence the function if and only if $x_{i_1} = d_1$. This is due to the fact that $x_{i_k^0}$ is queried only once in the decision tree. Similarly,

$$g_1(X_1) = (x_{i_2} \oplus \overline{d_2})g_2(X_2) \oplus (x_{i_2} \oplus d_2)h_2(Y_2),$$

and so on. Finally we have

$$g_{k-2}(X_{k-2}) = (x_{i_{k-1}} \oplus \overline{d_{k-1}})x_{i_k^0} \oplus (x_{i_{k-1}} \oplus d_{k-1})h_{k-2}(Y_{k-2}).$$

Therefore, the function f_n can be written as

$$f_n = (x_{i_1} \oplus \overline{d_1})(x_{i_1} \oplus \overline{d_2}) \dots (x_{i_{k-1}} \oplus \overline{d_{k-1}})x_{i_k^0} \oplus h_{k-1}(Y_k).$$

This implies that the resultant ANF contains a k-term monomial $x_{i_1}x_{i_2} \dots x_{i_k^0}$, i.e., $\deg(f) \geq k$.

It has been shown in [10, 3.1] that the minimum depth of any parity decision tree completely describing f is at equal to or greater than $\deg(f)$, which implies there does not exist any query friendly function that can be completely described with a parity decision tree of depth $k-1$. This concludes our proof. □

With this proof of limitation we conclude the study of Query friendly functions in this chapter. Finally, we present a graphical representation of our understanding of the query friendly functions in Figure 2.7 for $n \leq 950$ variables.

To summarize, our understanding is as follows.

- For values of n such that $n \in \{2^k - 1, 2^k - 2\}, k \in I_{\geq 2}$ where $I_{\geq 2}$ is the set of all integers greater than 2 we have obtained that no query friendly functions on n variables is separable.

- For values of n such that $2^{k-1} - 1 < n \leq 2^{k-1} + 2^{k-2} - 1$ we have obtained functions whose exact quantum query complexity is less than DQ_n and this complexity can be achieved using a parity decision tree.

- Finally, for other values of n, that is $2^{k-1} + 2^{k-2} - 1 < n < 2^k - 2$ we have proven that one cannot design a parity decision tree for a query friendly function on n variables such that the

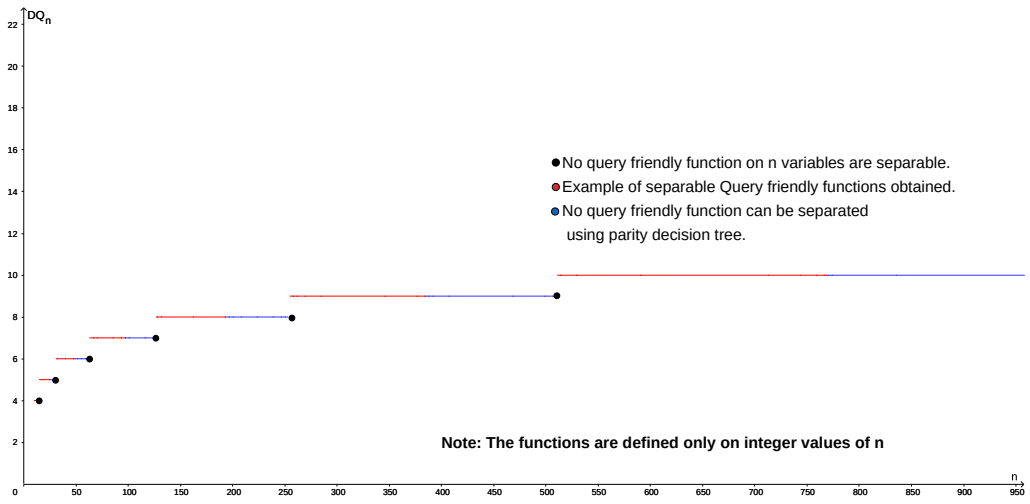

Figure 2.7: Current understanding of query friendly functions

query complexity of the tree is less than DQ_n. This completes our study of query friendly functions.

2.4 Conclusion

In this chapter we have first discussed the separation between the deterministic and exact quantum query model in terms of the number of influencing variables in Section 2.2 and 2.3. We have used the parity decision tree model to find separation between deterministic and exact quantum query in a special class of Boolean functions (Query Friendly functions) using the structured nature of the parity decision tree model. The characterization achieved by us in terms of query friendly functions is as follows.

1. For all varies of n there exists a non-separable query friendly function.

2. If $n = 2^k - 1$ or $n = 2^k - 2$, $k > 2$, then all query friendly functions are non-separable.

3. If $n \neq 2^k - 1$, then we construct a set of non-separable functions, namely $f_{(n,1)}$.

4. If $2^{k-1} - 1 < n \leq 2^{k-1} + 2^{k-2} - 1$, then we construct a set of separable functions, namely $f_{(n,2)}$.

5. If $2^{k-1} + 2^{k-2} - 1 < n \leq 2^k - 1$, we show that no separable function on n variables can be completely described using parity decision trees.

In this regard we have observed the following open problems which shall exhaustively determine the limitation of the parity decision tree model in these cases. The problems are as follows:

1. Does there exist a function f_1 with $n = 2^k - 1$ influencing variables such that $Q_E(f_1) < k$?

2. Does there exist a separable query friendly function f_2 with n influencing variables, where $2^{k-1} + 2^{k-2} - 1 < n < 2^k - 2,\ k > 2$?

If any of the above problems yield a negative result that would imply the parity decision tree model indeed completely characterizes the functions in such a scenario.

Bibliography

[1] A. Ambainis. 2018. Understanding Quantum Algorithms via Query Complexity Proceedings of the International Congress of Mathematicians, pp. 3265-3285.

[2] A. Ambainis. 2016. Superlinear Advantage for Exact Quantum Algorithms SIAM J. Comput. 45, pp. 617-631.

[3] A. Ambainis. 2013. Superlinear advantage for exact quantum algorithms. In Proceedings of the 45th Annual ACM Symposium on Theory of Computing (ACM Press, New York), pp. 891–900.

[4] A. Ambainis , J. Iraids and D. Nagaj. 2017. Exact Quantum Query Complexity of $\mathsf{EXACT}^{n}_{k,l}$. Theory and Practice of Computer Science. SOFSEM 2017. LNCS, vol 10139.

[5] H. Barnum, M. Saks and M. Szegedy. 2003. Quantum query complexity and semi-definite programming, In proceedings of 18th IEEE Annual Conference on Computational Complexity, pp. 179-193.

[6] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. 2001. Quantum lower bounds by polynomials. Journal of the ACM, 48(4), pp. 778-797.

[7] H. Buhrman and R. de Wolf. 2002. Complexity measures and decision tree complexity: a survey Theoretical Computer Science, 288(1), pp. 21-43.

[8] W. Chen, L. Li and Z. Ye. 2020. Characterization of exact one-query quantum algorithms Phys. Rev. A 101, 022325.

[9] M. Grant and S. Boyd. 2011. CVH: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com/cvx`

[10] A. Montanaro, R. Jozsa, and G. Mitchison. 2015. On Exact Quantum Query Complexity. Algorithmica 71, pp. 775–796.

Chapter 3

Novel Exact Quantum Query Algorithms from ANF

3.1 Introduction

In this chapter our main aim is to analyze certain structural properties of Boolean functions to obtain new separations between the deterministic classical and exact quantum query model via a combination of new algorithmic techniques through un-entangling of systems along with classical results in the domain of Boolean function analysis. In the query complexity model, different aspects of Boolean functions are analyzed to design efficient quantum query algorithms. Examples are symmetry and Walsh Spectrum (Fourier spectrum) of the function in consideration. In this chapter we concentrate on the $\mathbb{F}_2$ polynomial of a Boolean function. This is also called the algebraic normal form (ANF). Notably ANF is a very important property of Boolean functions from a cryptographic aspect [7], in particular the algebraic degree. However ANFs are seldom used in complexity theoretic studies. Against this backdrop, a central theme of this chapter is to use the ANF of the functions in the pdsp class to merge our positive and negative results. Our exact quantum query algorithm $Q_{algo}()$ is designed by analyzing the ANF of these functions. In this regard we have created a novel untangling protocol that gives us optimal complexity. On the other hand the functions in the pdsp class have high granularity. Granularity is a property of the Fourier spectrum of a function and higher value implies higher parity decision tree complexity. Now, the high granularity of the pdsp class is a direct consequence of its ANF structure. This result allows us to separate $QC_{algo}(f)$ and $D_{\oplus}(f)$, which, alongside the novel untangling protocol, is the main result of this chapter.

As we have discussed before, query complexity can be defined for both deterministic and probabilistic classical computational settings, as well as in the bounded error quantum and exact quantum model. Out of these models, the Exact Quantum Query complexity model is perhaps the least explored. Algorithms showing separations between the classical deterministic and the exact quantum query model has been formulated for very few classes of functions. In the exact quantum query model, a Boolean function f needs to be evaluated correctly for all possible inputs. The class of functions for which classical-quantum separation is known, and more importantly, for which we have exact quantum algorithms which outperform the classical algorithms are far and few. Mostly the exact quantum query algorithms that exist use the same method of calculating of parity of two input bits in one query, as mentioned in the work by Ambainis et. al. [2].

> "However, the techniques for designing exact quantum algorithms are rudimentary compared to the bounded error setting. Other than the well known 'XOR' trick — con-

structing a quantum algorithm from a classical decision tree that is allowed to 'query' the XOR of any two bits — there are few alternate approaches."

Even in this case, there is no generalized method for constructing parity decision trees that exhibit deterministic-exact quantum advantage for a given class of functions. The most striking result in this area is the example of super-linear separation between the deterministic classical and exact quantum models, shown in [1]. The work by Barnum et. al. [5] is also equally important, defining a semidefinite programming formulation to find out the exact quantum query complexity of a given function and also discovering an algorithm to achieve it. Finding such separations remains the most interesting problem in this area. In terms of the gap between $D(f)$ and $Q_E(f)$, the separations can be distinguished into two different kinds.

1. The first is identifying functions f so that $Q_E(f) < \frac{D(f)}{2}$. As explained in [12] this leads to super-linear separation between $Q_E(f^k)$ and $D(f^k)$ where f^k is obtained by recursively expanding the function f.

2. The second is where $Q_E(f) \geq \frac{D(f)}{2}$. We do not have any known method of converting such a result into super-linear separation. However, studying separation of this kind is still of considerable importance. This is because there are very few results in the exact quantum query model, and it is always of interest to find new approaches of evaluating functions in this model beyond the well known parity method, as highlighted in the quote above [2].

Before proceeding further, let us first review the main results in this area in a chronological fashion to show where exactly our work is placed among the state of the art literature.

2012: [1] Superlinear separation between exact quantum query complexity ($Q_E(f)$) and deterministic query complexity ($D(f)$) is obtained. This could be achieved by first obtaining a function f with $Q_E(f) < \frac{D(f)}{2}$ and then recursively expanding it.

2013: [3] Exact quantum query complexities of the symmetric function classes $\mathsf{Threshold}^n_k$ and Exact^n_k have been obtained. For both the cases $Q_E(f) > \frac{D(f)}{2}$ and thus these results provided linear separation only.

2015: [4] Near quadratic separation between $Q_E(f)$ and $D(f)$ was obtained using the concept of pointer functions.

2016: [2] Exact quantum query complexity of the symmetric function class $\mathsf{Exact}^{\mathsf{n}}_{\mathsf{k,l}}$ was obtained. For all the functions we have $Q_E(f) > \frac{D(f)}{2}$ and thus the separation was linear.

As observed from the chronology above, discovering exact quantum query complexity, with only linear separation between the classical and quantum models, that is, $Q_E(f) > \frac{D(f)}{2}$, remained a relevant topic even after discovering the results related to near quadratic separation.

Against this backdrop, we study the exact quantum query model for Boolean functions by a combined analysis of $\mathbb{F}_2$ polynomial and Fourier spectrum to obtain linear separation between $Q_E(f)$ and $D(f)$ and show that our algorithms are more efficient than any parity decision tree method. In fact, the algorithms we design outperform generalized parity decision tree methods, where one can obtain the parity of any $i \leq n$ variables using a single query. Another interesting characteristics of the class of functions we obtain is that, the size of these classes are considerably larger compared to the symmetric function classes for which linear separations were previously obtained. The comparison of the existing results with our findings are summarized in Table 3.1.

Function	Ref.	Complexity of Exact Quantum Query Algorithm	Total functions covered for n	Provably Optimal?
$\mathsf{Exact}^{\mathsf{n}}_{\mathsf{k}}$	[3]	$\max\{k, n-k\}$	$n+1$ (one for each value of k)	yes
$\mathsf{Threshold}^{\mathsf{n}}_{\mathsf{k}}$	[3]	$\max\{k, n-k+1\}$	$n+1$ (one for each value of k)	yes
$\mathsf{Exact}^{\mathsf{n}}_{\mathsf{k,l}}$	[2]	$\max\{n-k, l\}+1$	$\binom{n}{2}$ (one for each $\{k,l\}$ pair)	For most cases
The class – pdsp	our work	$\lfloor \frac{3n}{4} \rfloor$	$\Omega(\sqrt{2^{\sqrt{n}}})$	yes
A subclass of MM type Bent functions	our work (Chapter 4)	$\lceil \frac{5n}{8} \rceil$	$\Omega((2^{\lfloor \frac{n}{4} \rfloor}!)^2 2^{2^{\lfloor \frac{n}{4} \rfloor}})$	No

Table 3.1: Advantage achieved by Query Algorithms

Before proceeding further, we first define the following notations that we use in the document.

Definition 3.

1. $\boldsymbol{D_{\oplus}(f)}$ *and* $\boldsymbol{D^{(2)}_{\oplus}(f)}$*: We define the generalized parity decision tree complexity* $D_{\oplus}(f)$ *of a function* f *as the minimum number of queries any algorithm must make where the algorithm can obtain any parity* $\oplus_{i\in S}x_i$ *in a single query where* S *is any subset of* $[n] = \{1, \ldots, n\}$*. If we restrict* $|S| = 2$ *then the algorithm is a parity decision tree, which is the most well known exact quantum query algorithm and we denote by* $D^{(2)}_{\oplus}(f)$ *the minimum number of queries any parity decision tree needs to make to evaluate* f*. Consequently* $D_{\oplus}(f) \leq D^{(2)}_{\oplus}(f)$*.*

2. $\boldsymbol{Q_{algo}(f)}$ *and* $\boldsymbol{QC_{algo}(f)}$*: For any Boolean function* f*, we denote by* $Q_{algo}(f)$ *the exact quantum query algorithm designed in this document to evaluate the function. The number of queries required by* $Q_{algo}(f)$ *is denoted with* $QC_{algo}(f)$*, which we denote as the query complexity of the exact quantum query algorithm* $Q_{algo}(f)$*. We call an algorithm* $Q_{algo}(f)$ *optimal if we have* $QC_{algo}(f) = Q_E(f)$*. Here it should be clearly noted that* $Q_{algo}(f)$ *is an exact quantum query algorithm we design whereas* $QC_{algo}(f)$ *is a number, which is the query complexity of* $Q_{algo}(f)$*.*

We have the following relations between the aforementioned quantities.

Fact 1. *For any Boolean function* f *we have*

1. $Q_E(f) \leq D^{(2)}_{\oplus}(f) \leq D(f)$.
2. $D_{\oplus}(f) \leq D^{(2)}_{\oplus}(f) \leq D(f)$.

One should note here that unlike the situation of $Q_E(f) \leq D^{(2)}_{\oplus}(f)$, there is no strict relationship between $D_{\oplus}(f)$ and $Q_E(f)$. In fact for the simple parity function on n variables, we have $Q_E(f) = \lceil \frac{n}{2} \rceil$ whereas it only takes a single query in the generalized parity decision tree model by definition, making $D_{\oplus}(f) = 1$. Let us now lay out the organization and contribution of this chapter.

3.1.1 Organization and Contribution

Section 3.2 is a warm up to the contributory sections ahead. We discuss the different notions and definitions we use to obtain our result, along with a Semi Definite Programming based buildup to our results.

Section 3.3 is on the pdsp class of functions. We build various algorithmic techniques needed towards proving Theorem 14, which is the main contribution of this chapter. We show the existence

of a class of direct sum based non symmetric functions of size $\Omega\left(\sqrt{2\sqrt{n}}\right)$ functions for which we show that $QC_{algo}(f) = Q_{algo_E}(f) = \lceil \frac{n}{4} \rceil$ and the generalized parity decision complexity is always higher than $QC_{algo}(f)$, ranging from $\lceil \frac{n}{4} \rceil + 1$ to $n - 1$. En route to proving these results, we also obtain a new untangling protocol whose applications beyond these classes and this model should be interest.

This chapter is concluded in Section 3.4 with open problems and future research directions.

3.2 Warm up

In this text we will be interested in the exact quantum and deterministic classical model. Like we have discussed, different properties of Boolean functions can be exploited to design quantum query algorithms that outperform classical query algorithms. In the exact quantum model, we have two classes of functions for which superlinear separation have been obtained[2]. The work on [2] was based on starting with the symmetric function $x_1x_2 \oplus x_1x_3 \oplus x_2x_3$ and then recursive amplifying it to obtain superlinear advantage. The other work was based on pointer functions, which obtained separation between $Q_E(f)$ and $D(f)$ in non-Boolean functions and then carried over the transformation to Boolean function via appropriate modification. These are the two seminal works in this domain. Beyond that, optimal exact quantum query algorithm with linear separation has been obtained for some classes of symmetric functions such as $\mathsf{EXACT}^{\mathsf{n}}_{\mathsf{k}}$, $\mathsf{THRESHOLD}_{\mathsf{k}}$ [3], by exploiting properties of symmetry. However, given that there are 2^{2^n} Boolean functions on n variables, this constitutes only a minuscule proportion of functions for which optimal algorithms are known in the exact quantum query model. This makes analysis and design of new exact quantum algorithms necessary to have a better understanding of this paradigm. To this end we analyze the algebraic normal form (ANF) of Boolean functions. The ANF of a function $f(\mathbf{x}) = f(x_1, \ldots, x_n)$ is the unique $\mathbb{F}_2$ polynomial on these variables that represents a Boolean function. Any ANF has the two following operation:

- The $\cdot$ operation, which is the simple multiplication operator. For any two variables, or more we write $x_1 \cdot x_2$ simply as x_1x_2.
- The $\oplus$ operation, which is the addition modulo operation.

Naturally, let us first look into the following two ANFs for functions on n variables.

1. The function f_1 that only uses the $\oplus$ operation. he ANF is $f_1(\mathbf{x}) = x_1 \oplus x_2 \ldots \oplus x_n$. This is generally called the parity function on n variables. It has $Q_E(f) = \lceil \frac{n}{2} \rceil$ and $D(f) = n$. So we have a linear separation but since $Q_E(f) \geq \frac{D(f)}{2}$ there is no possibility of superlinear separation starting from this function.

2. The function that only uses the $\cdot$ operation. The corresponding ANF is $f_2(\mathbf{x}) = x_1 x_2 \cdots x_n$. This is generally called the $\mathsf{AND_n}$ function. For this function we have $Q_E(f) = D(f) = n$, and there is no separation.

Naturally we are interested in the two most immediate ANF with some degree of symmetry in their structure, described in the following problem statement.

Problem statement. *Let us assume that n is even with $n = 2k$. Then what is the $Q_E(f)$ and $D(f)$ value of the functions*

1. $f(\mathbf{x}) = x_1 x_{k+1} \oplus x_2 x_{k+2} \ldots x_k x_n$*?*

2. $f(\mathbf{x}) = x_1 x_2 \cdots x_k \oplus x_{k+1} \cdots x_n$*?*

Studying these two ANF forms has led us to the results described in this Part. Before moving ahead, we now describe the parity decision tree and the more powerful general parity decision method, which is an important benchmark for the results to follow. First we describe the deterministic query model, which can also be viewed as a decision tree.

Decision Tree

3.2.1 Parity Decision Tree Method

The parity method is the most well known method in the exact quantum query model. It relies simply on the fact that for any two variables x_i and x_j one can obtain the value of $x_i \oplus x_j$ in the exact quantum model using a single query. In contrast it would require 2 queries in the classical model. We simply create the state $\frac{|i\rangle + |j\rangle}{\sqrt{2}} |-\rangle$ and apply the oracle O_x to it to get $\frac{(-1)^{x_i}|i\rangle + (-1)^{x_j}|j\rangle}{\sqrt{2}} |-\rangle$ and apply a Hadamard like get to get back $|x_i \oplus x_j\rangle |-\rangle$. This method optimally evaluates the parity function in the exact quantum query model. In this regard, any exact quantum query algorithm that either queries a single variable or the parity of any two variables can be viewed as a decision tree where each internal node is either a variable x_i or the parity of two variables $x_i \oplus x_j$. These

algorithms are called parity decision trees. Given a function f, we have defined by $D^{(2)}_{\oplus}(f)$ the minimum number of queries a parity decision tree would have to make to evaluate a function in the worst case scenario.

Thus, given a Boolean function, if $Q_E(f) = D^{(2)}_{\oplus}(f)$ then one does not need to formulate any other strategy to have an optimal exact quantum query algorithm. Next, we simply extend this model to one in which one can take the parity of any number of variables using a single query. This we denote as the generalized parity decision tree. For example, in this model the query complexity of the parity function would just be 1, as opposed to $Q_E(f) = \lceil \frac{n}{2} \rceil$. We call the minimum query requirement in this model as the generalized parity decision tree complexity $D_{\oplus}(f)$. Note that unlike the original parity decision tree model, this does not generate from an equivalent quantum protocol, but is rather an extension of the model. This however trivially implies $D_{\oplus}(f) \leq D^{(2)}_{\oplus}(f)$ for any function.

Having described these two models, now we discuss a result on $D_{\oplus}(f)$ due to a property of the Fourier spectrum of a Boolean function termed "granularity" citeqalgo-PAR. This result gives us lower bound on the generalized parity decision tree complexity which separates our exact quantum query algorithm from any parity decision tree technique.

3.2.2 Granularity

First we define the Fourier spectrum of a function, The Fourier Spectrum of a function f on n variables $\hat{f} : 2^{[n]} \to \mathbb{R}$ defined as follows.

Definition 4 (Fourier Spectrum)**.**

1. *For a set $S \subseteq [n]$ the Fourier Character at S is defined as $\chi_S(\mathbf{x}) = \prod_{i \in S} (-1)^{x_i}$.*
2. *The Fourier coefficient of f on a set $S \subseteq [n]$ is denoted as $\hat{f}(S) = \frac{\sum_{\mathbf{x} \in \{0,1\}^n} \left((-1)^{f(\mathbf{x})} \chi_S(\mathbf{x}) \right)}{2^n}$.*

The Fourier spectrum is one of the most important tools for Boolean function analysis, with impact in domains ranging from cryptography and coding theory to complexity theory. Here we discuss the property of granularity, which was used by [16] to give lower bounds on the generalized decision parity tree complexity.

Definition 5 (Granularity)**.** *The granularity of a rational number t is defined as $gran(f) = k$ where k is the smallest power of two such that $t \times k$ is an integer.*

Now we note the result in [16] that elegantly shows $D_{\oplus}(f)$ is connected to granularity.

Theorem 10. *[16] The general parity complexity of a Boolean function f on n variables is lower bounded by $D_{\oplus}(f) \geq gran_m(f) + 1$ where $gran_m(f) = \max_{S \subseteq [n]} gran(\hat{f}(S))$.*

Now we discuss the class of functions that we study in this text along with some basic motivation behind the selection.

3.2.3 Perfect Direct Sum and Beyond

Next we look into functions of the form $f(\mathbf{x}) = \prod_{i=1}^{k} x_i \ \oplus \ \prod_{j=k+1}^{n} x_j$. As we shall formally prove in Section 3.3, due to Theorem 10 the generalized parity complexity of this function is $n-1$. When we checked $Q_E(f)$ value of this function for $n = 7$ we obtained $Q_E(f) = 6$, which implies that the parity decision tree method cannot be the optimal method for this class of functions. In this direction we worked on this function to create a class called perfect direct sum with product that we denote as pdsp class that contains $\Omega\sqrt{2^{\sqrt{n}}}$ functions. We obtain optimal exact quantum query algorithm outperforming parity decision as well as generalized parity decision tree bounds for this class in Section 3.3. Note that both the optimality of our novel algorithmic technique as well as the high $D_{\oplus}$ value are consequences of a study of its ANF structure. Finally we formally describe the environment in which the exact quantum query algorithms are designed along with some unitary operations that we use.

3.2.4 Setup for Quantum Query Algorithm

The set-up for a Quantum Query algorithm in relation to Boolean functions is as follows. Given a Boolean function on n influencing variables, a Quantum Query Algorithm for evaluating the function is defined on the Hilbert space $H = H^a \otimes H^q \otimes H^w$.

- Here H^a represents an n qubit register that contains the input to the function. The inputs stored in the input register can only be accessed using the oracle O_x, which operates on $H^a \otimes H^q$.

- The Hilbert space H^q is $n+1$ dimensional, and can be indexed with the basis states $|\mathbf{0}\rangle$ $|\mathbf{1}\rangle, |\mathbf{2}\rangle, \ldots |\boldsymbol{n}\rangle$. This space is used to make queries to the oracle and we call this the query register Q_n.
- The Hilbert space H^w is used as the working memory and has no restrictions. We define H^w to be formed of some w qubits, where the basis states of a qubit is $|0\rangle$ and $|1\rangle$ [1].

The oracle O_x works on the space $H^a \otimes H^q$ in the following way.

- $1 \leq i \leq n : O_x |\mathbf{x}\rangle |\boldsymbol{i}\rangle |w\rangle = (-1)^{x_i} |\mathbf{x}\rangle |\boldsymbol{i}\rangle |w\rangle$.
- $i = 0 : O_x |\mathbf{x}\rangle |\mathbf{0}\rangle |w\rangle = |x\rangle |\mathbf{0}\rangle |w\rangle$.

Since the input register remains unchanged throughout the algorithm, we describe our algorithm on $H^q \otimes H^w$, and describe the working of the oracle as $O_x |\boldsymbol{i}\rangle = (-1)^{x_i} |\boldsymbol{i}\rangle$, $1 \leq i \leq n$ and $O_x |\mathbf{0}\rangle = |\mathbf{0}\rangle$.

An algorithm that uses the oracle k times can be expressed as a series of unitaries $U_0, U_1, \ldots U_k$ applied on $H^q \otimes H^w$ with an oracle access between each U_i and U_{i+1}, $0 \leq i \leq k-1$. The algorithm starts with the state $|\psi\rangle = |\mathbf{0}\rangle |0\rangle \ldots |0\rangle$ and finally reaches the state $U_k O_x U_{k-1} \ldots U_1 O_x U_0 |\psi\rangle$, on which some measurement is performed and the output is decided depending on some predefined condition on the result.

An exact quantum query algorithm is one which evaluates a function correctly for any input. The Exact Quantum Query complexity (Q_E) of a function is the least possible number of queries an exact quantum query algorithm needs to make at most to evaluate the function in any point.

The workspace of Q_{algo}

The workspace of the algorithms that we design consists of Q_n and $l+1$ qubits for some l. In this document we denote the basis states of the query register with $|\mathbf{0}\rangle_0, |\mathbf{1}\rangle_0, \ldots, |\boldsymbol{n}\rangle_0$. The qubits are denoted by w_1 through w_{l+1}. We denote the computational basis states of the i-th work qubit by $|0\rangle_i$ and $|1\rangle_i$. Thus we describe this system with the basis states

$$|\boldsymbol{a_0}\rangle_0 \bigotimes_{i=1}^{t} |a_i\rangle_i, \ a_0 \in \{0, 1, \ldots, n\}, \ a_i \in \{0, 1\} \ \forall i \in \{1, \ldots, l+1\}.$$

[1] Therefore a Quantum Query Algorithm corresponding to a function f with n influencing variables is essentially a circuit defined on the Hilbert space H of $n + \lceil \log(n+1) \rceil + w$ qubits with the restriction that the n qubits corresponding to the input register can only be accessed through an oracle.

We finish this section by describing some general unitary operations that we use in our algorithms.

3.2.5 Some Unitary Matrices

We use the following classes of unitary operators (defined as matrices) to build the algorithms corresponding to different functions.

$\mathsf{P}^{\mathsf{n}}_{\mathsf{i}}$:

This operator is applied on the $n+1$ dimensional query register Q_n. This is a permutation matrix which transforms the state $|1\rangle_0$ to $|i\rangle_0$ and transforms $|i\rangle_0$ to $|1\rangle_0$ for all other basis states $|j\rangle_0, j \neq i$ it acts as $\mathsf{P}^{\mathsf{n}}_{\mathsf{i}} |j\rangle_0 = |j\rangle_0$. The corresponding $(n+1) \times (n+1)$ matrix can be built with the following directions.

- $\mathsf{P}^{\mathsf{n}}_{\mathsf{j}}(1, i) = 1$
- $\mathsf{P}^{\mathsf{n}}_{\mathsf{j}}(i, 1) = 1$
- $\mathsf{P}^{\mathsf{n}}_{\mathsf{j}}(j, j) = 1$ if $j \notin \{1, i\}$.
- $\mathsf{P}^{\mathsf{n}}_{\mathsf{j}}(j, k) = 0$ otherwise.

$\mathsf{Par}^{\mathsf{n}}_{\mathsf{i,j}}$:

This is a Hadamard like operator intended for the following transformation of the query register Q_n.

$$\frac{(-1)^a |i\rangle_0 + (-1)^b |j\rangle_0}{\sqrt{2}} \xrightarrow{\mathsf{Par}^{\mathsf{n}}_{\mathsf{i,j}}} (-1)^a |a \oplus b\rangle_0$$

This operation can be implemented as an $(n+1) \times (n+1)$ matrix with the following constraints.

- $\mathsf{Par}^{\mathsf{n}}_{\mathsf{i,j}}(0, i) = 0$
- $\mathsf{Par}^{\mathsf{n}}_{\mathsf{i,j}}(0, j) = \frac{1}{\sqrt{2}}$
- $\mathsf{Par}^{\mathsf{n}}_{\mathsf{i,j}}(1, i) = -\frac{1}{\sqrt{2}}$
- $\mathsf{Par}^{\mathsf{n}}_{\mathsf{i,j}}(1, j) = \frac{1}{\sqrt{2}}$

$\mathsf{S}^{\mathsf{n}}_{\mathsf{i,j}}$:

This operation simply transforms the state $|0\rangle_0$ to $\frac{|i\rangle_0 + |j\rangle_0}{\sqrt{2}}$. We can simply define this matrix as $\mathsf{S}^{\mathsf{n}}_{\mathsf{i,j}} = \left(\mathsf{Par}^{\mathsf{n}}_{\mathsf{i,j}}\right)^T$ where A^T is the transpose of A.

$\mathsf{S}^{n}_{i,j}$

This operation is also defined on an $n+1$ dimensional register. It only performs the transformation $|\mathbf{0}\rangle \xrightarrow{\mathsf{S}^{n}_{i,j}} \frac{1}{\sqrt{2}}(|\boldsymbol{i}\rangle + |\boldsymbol{j}\rangle)$.

The corresponding matrix can fairly simply be defined as $\mathsf{S}^{n}_{i,j} = \left(\mathsf{Par}^{n}_{i,j}\right)^{*}$. Below are examples of the matrices corresponding to operations of type P^{n}_{i}, $\mathsf{Par}^{n}_{i,j}$ and $\mathsf{S}^{n}_{i,j}$.

$$\mathsf{P}^{4}_{3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathsf{Par}^{4}_{1,3} = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathsf{S}^{4}_{0,1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

To avoid confusion, it should be noted that in a Quantum Computer built with qubits, a unitary operator working on z qubits is has a dimension of 2^{z}. In the following remark we state how a general n dimensional register can be implemented in such a setup.

Remark 2. *To implement an unitary operator U on an $n+1$ dimensional register, it would require $\lceil \log(n+1) \rceil$ qubits and thus the corresponding unitary matrix U' being applied on these qubits would actually be $2^{\lceil \log(n+1) \rceil}$ dimensional. The matrix U' can be formed by adding $2^{\lceil \log(n+1) \rceil} - (n+1)$ rows and columns to U, such that entries in the rows and columns corresponding to the basis states $|i\rangle, i \in \{n+1, \ldots, \lceil \log(n+1) \rceil - 1\}$ would simply be $U'(i,i) = 1$.*

For example, given the matrix P^{2}_{2}, the matrix $\mathsf{P}^{2'}_{2}$ that would be implemented in a $2^{\lceil \log(3+1) \rceil} = 4$ dimensional system built of 2 qubits is as follows:

$$\mathsf{P}^{2}_{2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \mathsf{P}^{2'}_{2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\mathsf{C_0}U$ and $\mathsf{C_1}U$

The algorithm we design uses unitary matrices that are controlled on the state of a work qubit w_1. At each step we apply a set of unitaries controlled on $w_1 = |0\rangle$ and another set controlled on $w_1 = |1\rangle$. Given any unitary U, we denote by $\mathsf{C_0}U$ the operation that is U controlled on $w_1 = |0\rangle$. We use the notation $\mathsf{C_1}U$ to denote the operation that is U controlled on $w_1 = |1\rangle$. It is easy to see that if U is a unitary operation, then so is $\mathsf{C_0}U$ and $\mathsf{C_1}U$.

$\mathsf{CNOT}^{\mathsf{a}}_{\mathsf{b}}$

This operation is the Controlled-NOT operation from register a to register b, with register a as control and register b as target. Here either one of the registers is the query register, or else both the registers are work qubits. Let us suppose register a is the query register, then the transformation will be denoted by

- $|\mathbf{1}\rangle_a |0\rangle_b \rightarrow |\mathbf{1}\rangle_a |1\rangle_b$
- $|\mathbf{1}\rangle_a |1\rangle_b \rightarrow |\mathbf{1}\rangle_a |0\rangle_b$

If both the registers are qubits, then it works as the conventional C-NOT operation. If a and b are both qubits, then it is a 4 dimensional unitary operation, otherwise it is a $2(n+1)$ dimensional operation.

$swap(a, b)$

This operation is simply defined as $swap(a, b) = \mathsf{CNOT}^{\mathsf{a}}_{\mathsf{b}}\ \mathsf{CNOT}^{\mathsf{b}}_{\mathsf{a}}\ \mathsf{CNOT}^{\mathsf{a}}_{\mathsf{b}}$, and swaps the value of two registers a and b, with dimensions d_1 and d_2, so that it is defined on the computational basis states $|i\rangle \otimes |j\rangle : i, j \leq \min(d_1, d_2)$.

3.3 The pdsp class

Before formalizing things we give an introduction to the functions we deal with in this section and the way the results follow. We will assume $n = 2k$ for brevity for now. We start with the function $f_1(\mathbf{x}) = \prod_{i=1}^{k} x_i \bigoplus \prod_{j=k+1}^{n} x_j$, which is our second function of interest in understanding ANF structures with respect to exact quantum query complexity. This function is just the direct sum

of two $AND_{\frac{n}{2}}$ defined on two disjoint sets of variables. Thus this can be called a "perfect direct sum" function.

One very interesting fact about this function is that the granularity of this function is high with it being $n = 1$. This implies no parity decision tree can exactly compute this function for all values by making less than $n-1$ queries. In this regard we will form an algorithm via a novel untangling protocol Q_{algo} with $QC_{algo}(f_1) = \lceil \frac{3n}{4} \rceil$. However we cannot show that $Q_E(f_1) = QC_{algo}(f_1)$ for this function and thus our methodology is not yet proven to be optimal. In this direction we design the function $f_2(\mathbf{x}) = \prod_{l=\lfloor \frac{3n}{4} \rfloor + 1}^{n} x_l \left(\prod_{i=1}^{k} x_i \bigoplus \prod_{j=k+1}^{\lfloor \frac{3n}{4} \rfloor} x_j \right)$ and observe that for this function we have $Q_E(f_2) = QC_{algo}(f_2) = \lceil \frac{3n}{4} \rceil$ and granularity and this $D_{\oplus}(f_2) = n-1$ which is the ideally desirable situation. We call this function a "perfect direct sum with product", abbreviated as pdsp. We further modify the product term $\prod_{j=k+1}^{\lfloor \frac{3n}{4} \rfloor} x_j$ and break it up into a perfect direct sum of multiple monomials. As we shall observe this modification does not effect the $Q_E(f)$ and $QC_{algo}(f)$ values. However, we need certain restrictions so that the granularity value remains high and our algorithm is essential. Now we formally define the pdsp class. Then we shall obtain the deterministic and generalized parity decision tree complexity before moving onto the design of our exact quantum query algorithm.

Definition 6 (The pdsp class)**.** *A function is called a perfect direct sum function on n variables if all the variables $x_i, 1 \leq i \leq n$ appear only once in the function's unique algebraic normal form ($\mathbb{F}_2$ polynomial).*

A function f is said to belong to the class pdsp(n, l, q) *if the variable space $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ consisting of n variables can be divided into the two subspaces $\hat{\mathbf{x}} = (x_{r_1}, x_{r_2}, \ldots, x_{r_l})$ and $\tilde{\mathbf{x}} = (x_{r_{l+1}}, x_{r_{l+2}}, \ldots, x_{r_n})$ containing l and $n-l$ variables respectively so that*

1. *$f(\mathbf{x}) = f_1(\hat{\mathbf{x}}) f_2(\tilde{\mathbf{x}})$.*

2. *f_1 is a perfect direct sum function defined on the l variables $\hat{\mathbf{x}}$, which consists of q monomials such that each monomial consists of at least q variables.*

3. *f_2 is the product function of the $n-l$ variables in $\tilde{\mathbf{x}}$. That is, $f_2(\tilde{\mathbf{x}}) = \prod_{i=l+1}^{n} x_{r_i}$. If $l = n$ then f_2 function is not defined.*

3.3.1 $D(f)$ and $D_{\oplus}(f)$ for pdsp:

We first obtain the deterministic query complexity of the functions defined in Definition 6 by analyzing the polynomial degree of the function. Let $\mathsf{pdeg}(f)$ be the degree of the unique real multilinear polynomial $p : \mathbb{F}_2^n \to \mathbb{R}$ such that $f(\mathbf{x}) = p(\mathbf{x})\ \forall \mathbf{x} \in \mathbb{F}_2^n$. From [6], we know $D(f) \geq \mathsf{pdeg}(f)$. Then we have the following result.

Theorem 11. *For any function $f \in \mathsf{pdsp}(n, l, q)$ we have $\mathsf{pdeg}(f) = n$.*

Now let us discuss the generalized parity decision tree complexity $D_{\oplus}(f)$ for these functions. 1In this regard we have the following result.

Lemma 8. *Let f be a function defined on n variables such that $f \in \mathsf{pdsp}(n, l, q)$. Then we have $D_{\oplus}(f) = gran_m(f) + 1 = gran(\hat{f}(\{\phi\}) + 1 = n - q + 1$.*

Proof. We prove this by first showing $gran(\hat{f}(\{\phi\}) = n - q$ which implies $D_{\oplus}(f) \geq n - q + 1$ and then describe a simple general parity decision tree with complexity $n - 1 + 1$. For simplicity let us assume $r_i = i$.

$gran_m(f) \geq n - q$ The ANF of f_1 can be represented as a partition of $\hat{\mathbf{x}} = \{x_{r_1}, x_{r_2}, \ldots x_{r_l}\}$ into q disjoint sets. We denote these sets as $m_i, 1 \leq i \leq q$, where m_i consists of q_i variables. Then f can be written as

$$f(\mathbf{x}) = \left(\bigoplus_{i=1}^{q} \left(\prod_{x_j \in m_i} x_j \right) \right) \prod_{p=l+1}^{n} x_p.$$

We know that $\hat{f}(\{\phi\}) = \sum_{\boldsymbol{a} \in \{0,1\}^n} (-1)^{f(\boldsymbol{a})} = 2^n - 2wt(f)$ where $wt(f)$ is the number of input points for which the function outputs 1.

The output of f_1 for some input is 1 if some odd number of these q monomials are evaluated to 1 and $x_{r_p} = 1,\ l + 1 \leq p \leq n$. Let us denote by x^j all inputs from $\{0,1\}^l$ for which j of the said monomials evaluate to 1. If j is odd then for each input in x_j f_1 evaluates to 1. For each such $a \in x^j$, there is only input $a' \in \{0,1\}^n$ for which f evaluates to 1, where $a' = a||1_{n-l}$. We can represent the number of ones in the truth table of f as $\sum_{i=0}^{\lfloor \frac{q-1}{2} \rfloor} \left| x^{2i+1} \right|$. Then we have $\left| x^1 \right| = \sum_{i=1}^{q} \left(\prod_{j \neq i} (2^{q_j} - 1) \right)$ as it consists of inputs for which exactly one monomial has all variables set to one, and because of the monomial disjoint nature of the function there is no repetition in the counting. We can express x^1 as

$\left|x^1\right| = \alpha_1 2^q + q$ such that α_1 is an integer, if q is odd.

$\left|x^1\right| = \alpha_1 2^q - q$ such that α_1 is an integer, if q is even.

This is because in expansion of $\left|x^1\right|$ in each product term we have a $(-1)^{q-1}$ (-1 if k is even $+1$ otherwise) and all other terms are of the form $\pm 2^{q_{i_1}+q_{i_2}\cdots+q_{i_j}}$. since $q_i \geq q\ \forall i$, all these terms are integer multiple of 2^q, and thus their sum is also an integer multiple of 2^q, or zero. Now since each product term has a $(-1)^{q-1}$ and there are q terms in the expansion can be written as some $\alpha_1 2^q + (-1)^{q-1}q$. Similarly, x^i can be expressed as $\alpha_i 2^q + (-1)^{q-1}\binom{q}{i}$ and therefore the support set of f is of the size $\sum_{i=0}^{\lfloor \frac{q-1}{2} \rfloor} \left(\alpha_{2i+1} 2^q + (-1)^{q-1}\binom{q}{2i+1}\right) = \alpha 2^q + (-1)^{q-1}2^{q-1}$. Therefore the Fourier coefficient of the function at $S = \{\phi\}$ is

$$\widehat{f}(\{\phi\}) = \frac{2^n - 2\left(\alpha 2^q + (-1)^{q-1}2^{q-1}\right)}{2^n} = \frac{2^n - \alpha 2^{q+1} + (-1)^q 2^q}{2^n}.$$

Thus granularity of the Fourier coefficient at this point is $gran(\widehat{f}(\{\phi\})) = n - q$ and therefore $gran_m(f) \geq n - q + 1$.

$D_\oplus \leq n - q + 1$ We now show a simple general parity tree of with $n - q + 1$ queries that evaluates f, showing $D_\oplus(f) \leq n - q + 1$. Given an input $\mathbf{a} = \{a_1, a_2, \ldots, a_n\}$ It first queries all but one variable from each monomial of f_1. This takes $l - q$ queries. For the monomial m_i the product of these variables evaluate to $\tilde{m}_i = \prod_{j=1}^{q_i - 1} a_{i_j}$. Then only if $\tilde{m}_i = 1$ the output of f_1 depends on $x_{i_{q_i}}$. Therefore the final query to evaluate f_1 is the linear function $\bigoplus_{i=1}^{q} \tilde{m}_i x_{i_{q_i}}$ as the value of $\tilde{m}_i$ are already calculated. Thus evaluating f_1 needs $l - q + 1$ queries. Now we can simply evaluate f_2 which is defined on $n - l$ variables by querying each of the variables individually which enables us as to output the function $f(\mathbf{x}) = f_1(\hat{\mathbf{x}}) f_2(\tilde{\mathbf{x}})$. Therefore this method requires a total of $l - q + 1 + n - l = n - q + 1$ query, which shows $D_\oplus(f) \leq n - q + 1$.

Since $D_\oplus(f) \geq gran_m(f)$ and we have $gran_m(f) \geq n - q + 1$ and $D_\oplus(f) \leq n - q + 1$ this implies $D_\oplus(f) = gran_m(f) = n - q + 1$.

□

Having determined $D(f)$, $D_\oplus(f)$ and $D_\oplus^{(2)}(f)$, we now describe the family of exact quantum query algorithms Q_{algo} that we design, and their query complexity $QC_{algo}(f)$.

3.3.2 The Exact Quantum Query Algorithms for pdsp

Remark 3. *From here on we assume $n \equiv 2 \bmod 4$ with $k = \frac{n}{2}$. This is to simply reduce the tediousness of the proof. For other cases the algorithms and the bounds develop in an almost identical manner, conforming to the same generalized query complexity value. One can refer to the extended version of this work [14] to view the simple modifications needed to incorporate the other cases.*

The general flow of the algorithms are as follows.

- The function is expressed as $f(\mathbf{x}) = g(\hat{\mathbf{x}}) \oplus h(\tilde{\mathbf{x}})$ where $\hat{\mathbf{x}}$ and $\tilde{\mathbf{x}}$ are two subspaces that form a disjoint partition of $\mathbf{x}$.
- We then start with the state $|\mathbf{0}\rangle_0 \bigotimes_{i=1}^{s+1} |0\rangle_i$ where s is dependent on the structure of the function.
- We apply a Hadamard gate on the first work qubit w_1 to obtain the state

$$|\psi_0\rangle = \frac{1}{\sqrt{2}} \left(|\mathbf{0}\rangle_0 |0\rangle_1 \otimes_{i=1}^{s} |0\rangle_{i+1} + |\mathbf{0}\rangle_0 |1\rangle_1 \otimes_{i=1}^{s} |0\rangle_{i+1} \right).$$

- Here we have two product states in equal superposition, identifiable with the value of w_1. Note that the value of this qubit remains unchanged till the penultimate step of our algorithm and allows us to parallely obtain values of the variables. We transform the superposition state with $w_1 = |0\rangle$ according to values of the variables in $\hat{\mathbf{x}}$ and the superposition state with $w_1 = |1\rangle$ according to the variables in $\tilde{\mathbf{x}}$. This brings us to the state

$$|\psi_f\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\hat{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{i=1}^{s} |x_i\rangle_{i+1} + (-1)^{h(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{i=1}^{s} |x_{k+i}\rangle_{i+1} \right) \tag{3.1}$$

  using some $\hat{s} \leq s$ queries. Here note that we can have different variables instead of $x_i \in \hat{\mathbf{x}}$ and $x_{k+i} \in \tilde{\mathbf{x}}$ reflected in the qubits $|i+1\rangle$ with the two product states depending on the structure of the function, but it does not affect the analysis and the progress of the algorithms, so we write it as such for simplicity.

At this stage if we had $x_{k+i} = x_i = m_i$ (say) $\forall i$ we could write the state as

$|\mathbf{0}\rangle_0 \frac{1}{\sqrt{2}} \left((-1)^{g(\hat{\mathbf{x}})} |0\rangle_1 + (-1)^{h(\tilde{\mathbf{x}})} |1\rangle_1 \right) \bigotimes_{i=1}^{s} |m_i\rangle_{i+1}$ and we could simply apply a Hadamard gate on the w_1 to obtain

$|\mathbf{0}\rangle_0 \frac{1}{\sqrt{2}}|g(\hat{\mathbf{x}}) \oplus h(\tilde{\mathbf{x}})\rangle \bigotimes_{i=1}^{s} |m_i\rangle_{i+1}$ (ignoring global phase) and measuring w_1 would suffice. However we do not have any way of ensuring $x_{r(i)} = x_{r(l+i)}$ which leaves the state $|\psi_f\rangle$ in an entangled form. Here we design a new untangling protocol that finally gives us the separations.

The Untangling Protocol:

Our algorithm is currently in the state

$$|\beta_0\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\hat{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{i=1}^{s} |x_i\rangle_{i+1} + (-1)^{h(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{i=1}^{s} |x_{k+i}\rangle_{i+1} \right).$$

If the system was in a product state at this stage, we could have simply obtained the parity of the phases $(-1)^{g(\hat{\mathbf{x}})}$ and $(-1)^{h(\tilde{\mathbf{x}})}$, which would have given us the desired outcome. However, the system is entangled as the value of x_i may differ depending on the input on which we have no control. At this stage we design a technique of untangling two qubits deterministically using a single query. This protocol, and the fact that we can untangle "two" qubits using a single query is what gives us the advantage over the deterministic and parity decision tree technique. This can be summarized as follows.

Theorem 12. *Let a quantum query algorithm be in the state*

$$|\gamma\rangle = \frac{1}{\sqrt{2}} \left(|\boldsymbol{x_a}\rangle_0 |0\rangle_1 |x_b\rangle_2 |W_1\rangle + |\boldsymbol{x_c}\rangle_0 |1\rangle_1 |x_d\rangle_2 |W_2\rangle \right)$$

Here x_a, x_b, x_c and x_d are inputs to a function corresponding to an oracle. Then this state can be transformed to

$$|\gamma'\rangle = (-1)^{x_b} \frac{1}{\sqrt{2}} \left(|\boldsymbol{x_b}\rangle_0 |0\rangle_1 |x_d\rangle_2 |W_1\rangle + |\boldsymbol{x_b}\rangle_0 |1\rangle_1 |x_d\rangle_2 |W_2\rangle \right)$$

using a single query to the oracle. Here $|W_1\rangle$ and $|W_2\rangle$ represent any two arbitrary m-qubit states.

Proof. We again define a protocol, untangle which enables the defined transformation by making a single query to the oracle.

We first define the unitaries U_1 and U_2 that act on Q_n. The structure of untangle is as follows. First $\mathsf{C_0}U_1$ and $\mathsf{C_1}U_2$ are applied, followed by the oracle O_x and then $\mathsf{C_0Par^n_{a,d}}$ and $\mathsf{C_1Par^n_{b,c}}$. That

is, we define

$$\mathsf{untangle} = \left(\mathsf{C_0Par^n_{a,d}}\ \mathsf{C_1Par^n_{b,c}}\ O_x\ \mathsf{C_0}U_1\ \mathsf{C_1}U_2\right),$$

and show that $|\gamma\rangle \xrightarrow{\mathsf{untangle}} |\gamma'\rangle$.

We denote $|\boldsymbol{x_a}\rangle_0 |0\rangle_1 |x_b\rangle_2 |W_1\rangle = |\gamma_1\rangle$ and $|\boldsymbol{x_c}\rangle_0 |1\rangle_1 |x_d\rangle_2 |W_2\rangle = |\gamma_2\rangle$.Then $|\gamma\rangle = \frac{1}{\sqrt{2}}(|\gamma_1\rangle + |\gamma_2\rangle)$ Let us now observe the evolution of the two states $|\boldsymbol{x_a}\rangle_0 |0\rangle_1 |x_b\rangle_2$ and $|\boldsymbol{x_c}\rangle_0 |1\rangle_1 |x_d\rangle_2$ individually, depending on the state of the w_1.

We start with the case when $w_1 = |0\rangle$. U_1 can be looked as the composition of two operations U_{10} and U_{11}. U_{10} and U_{20} acts on the register Q_n depending on if $w_2 = |0\rangle$ or $|1\rangle$, i.e. $x_b = 0$ or $x_b = 1$ respectively. That is U_{10} and U_{20} are operators acting on $H^q \otimes H_2$. Therefore at any point, only one of the unitaries actually perform their transformations, depending on the value of x_b. These transformations are defined as follows.

U_{10}

1. $|\mathbf{0}\rangle_0 \rightarrow \frac{1}{\sqrt{2}}(|\boldsymbol{a}\rangle_0 + |\boldsymbol{d}\rangle_0)$
2. $|\mathbf{1}\rangle_0 \rightarrow \frac{1}{\sqrt{2}}(-|\boldsymbol{a}\rangle_0 + |\boldsymbol{d}\rangle_0)$

U_{11}

1. $|\mathbf{0}\rangle_0 \rightarrow \frac{1}{\sqrt{2}}(-|\boldsymbol{a}\rangle_0 + |\boldsymbol{d}\rangle_0)$
2. $|\mathbf{1}\rangle_0 \rightarrow \frac{1}{\sqrt{2}}(|\boldsymbol{a}\rangle_0 + |\boldsymbol{d}\rangle_0)$

That is,

- $|\boldsymbol{x_a}\rangle_0 \xrightarrow{U_{10}} \frac{1}{\sqrt{2}}((-1)^{x_a}|\boldsymbol{a}\rangle_0 + |\boldsymbol{d}\rangle_0)$
- $|\boldsymbol{x_a}\rangle_0 \xrightarrow{U_{11}} \frac{1}{\sqrt{2}}((-1)^{x_a+1}|\boldsymbol{a}\rangle_0 + |\boldsymbol{d}\rangle_0)$

The oracle is then applied on $\mathsf{C_0}U_{10}\mathsf{C_0}U_{11}|\gamma_1\rangle$, followed by the Unitary Operation $\mathsf{C_0Par^n_{a,d}}$. We now observe the state $\left(\mathsf{C_0Par^n_{a,d}}O_x\mathsf{C_0}U_{10}\mathsf{C_0}U_{11}\right)|\gamma_1\rangle$, depending on the value of x_2 and compare the resultant state with

$$(-1)^{x_b}\left(|\boldsymbol{x_b} \oplus \boldsymbol{x_d}\rangle_0\right)|0\rangle_1 |x_b\rangle_2.$$

We tabulate the comparisons for both $x_b = 0$ and $x_b = 1$ in Table 3.2. The transformations $\mathsf{C_0}U_10, \mathsf{C_0}U_{11}, O_x$ and $\mathsf{C_0Par^n_{a,d}}$ only act on the query register, depending on the values of the qubits w_1 and w_2, which remain unaltered throughout. Therefore we only show the evolution of the query register.

$\mathbf{x_b = 0}$:

x_a	$\mathsf{C_0}U_{10}\left\|\gamma_1\right\rangle$	$O_x\mathsf{C_0}U_{10}\left\|\gamma_1\right\rangle$	$\mathsf{C_0Par^n_{a,d}}O_x\mathsf{C_0}U_{10}\left\|\gamma_1\right\rangle$	$\left\|\boldsymbol{\beta}\right\rangle$
0	$\frac{1}{\sqrt{2}}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}\left\|\boldsymbol{d}\right\rangle_0$	$\frac{1}{\sqrt{2}}(-1)^{x_a}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}(-1)^{x_d}\left\|\boldsymbol{d}\right\rangle_0$	$(-1)^{x_a}\left\|\boldsymbol{x_a}\oplus\boldsymbol{x_d}\right\rangle_0$ $=\left\|\boldsymbol{x_d}\right\rangle_0$	$\left\|\boldsymbol{x_d}\right\rangle_0$
1	$-\frac{1}{\sqrt{2}}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}\left\|\boldsymbol{d}\right\rangle_0$	$\frac{1}{\sqrt{2}}(-1)^{x_a+1}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}(-1)^{x_d}\left\|\boldsymbol{d}\right\rangle_0$	$(-1)^{x_a+1}\left\|\boldsymbol{x_a}\oplus\boldsymbol{x_d}\oplus\mathbf{1}\right\rangle_0$ $=\left\|\boldsymbol{x_d}\right\rangle_0$	$\left\|\boldsymbol{x_d}\right\rangle_0$

$\mathbf{x_b = 1}$:

x_a	$\mathsf{C_0}U_{11}\left\|\gamma_1\right\rangle$	$O_x\mathsf{C_0}U_{11}\left\|\gamma_1\right\rangle$	$\mathsf{C_0Par^n_{a,d}}O_x\mathsf{C_0}U_{11}\left\|\gamma_1\right\rangle$	$\left\|\boldsymbol{\beta}\right\rangle$
0	$-\frac{1}{\sqrt{2}}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}\left\|\boldsymbol{d}\right\rangle_0$	$\frac{1}{\sqrt{2}}(-1)^{x_a+1}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}(-1)^{x_d}\left\|\boldsymbol{d}\right\rangle_0$	$(-1)^{x_a+1}\left\|\boldsymbol{x_a}\oplus\boldsymbol{x_d}\oplus\mathbf{1}\right\rangle_0$ $=-\left\|\boldsymbol{x_d}\oplus\mathbf{1}\right\rangle_0$	$-\left\|\boldsymbol{x_d}\oplus\mathbf{1}\right\rangle_0$
1	$\frac{1}{\sqrt{2}}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}\left\|\boldsymbol{d}\right\rangle_0$	$\frac{1}{\sqrt{2}}(-1)^{x_a}\left\|\boldsymbol{a}\right\rangle_0$ $+\frac{1}{\sqrt{2}}(-1)^{x_d}\left\|\boldsymbol{d}\right\rangle_0$	$(-1)^{x_a}\left\|\boldsymbol{x_a}\oplus\boldsymbol{x_d}\right\rangle_0$ $=-\left\|\boldsymbol{x_d}\oplus\mathbf{1}\right\rangle_0$	$-\left\|\boldsymbol{x_d}\oplus\mathbf{1}\right\rangle_0$

Table 3.2: Evolution of $\left|\gamma_1\right\rangle$ and comparison with $\left|\boldsymbol{\beta}\right\rangle = (-1)^{x_b}\left|\boldsymbol{x_b}\oplus\boldsymbol{x_d}\right\rangle_0$

Therefore in all the cases the state post these transformations is

$$(-1)^{x_b}\left|\boldsymbol{x_b}\oplus\boldsymbol{x_d}\right\rangle_0\left|0\right\rangle_1\left|x_b\right\rangle_2.$$

Now we describe the evolution of the state $\left|\boldsymbol{x_c}\right\rangle_0\left|1\right\rangle_1\left|x_d\right\rangle_2$. As in the previous case, we apply an unitary $\mathsf{C_1}U_2$ and then the state queries to the oracle, which is followed by $\mathsf{C_1Par^n_{b,c}}$. We define U_2 as the composition of two unitary operators defined on $H^q \otimes H_2$, U_{20} and U_{21}. Similar to U_{10} and U_{11}, these are operators that transform the query register depending on $w_2 = \left|0\right\rangle$ and $\left|1\right\rangle$, respectively. The transformations due to U_{20} and U_{21} are as follows.

U_{20}

1. $\left|\mathbf{0}\right\rangle_0 \to \frac{1}{\sqrt{2}}(\left|\boldsymbol{b}\right\rangle_0 + \left|\boldsymbol{c}\right\rangle_0)$
2. $\left|\mathbf{1}\right\rangle_0 \to \frac{1}{\sqrt{2}}(\left|\boldsymbol{b}\right\rangle_0 - \left|\boldsymbol{c}\right\rangle_0)$

U_{21}

1. $\left|\mathbf{0}\right\rangle_0 \to \frac{1}{\sqrt{2}}(\left|\boldsymbol{b}\right\rangle_0 - \left|\boldsymbol{c}\right\rangle_0)$
2. $\left|\mathbf{1}\right\rangle_0 \to \frac{1}{\sqrt{2}}(\left|\boldsymbol{b}\right\rangle_0 + \left|\boldsymbol{c}\right\rangle_0)$

That is

- $\left|\boldsymbol{x_c}\right\rangle_0 \xrightarrow{U_{20}} \frac{1}{\sqrt{2}}(\left|\boldsymbol{b}\right\rangle_0 + (-1)^{x_c}\left|\boldsymbol{c}\right\rangle_0)$

- $|\boldsymbol{x_c}\rangle_0 \xrightarrow{U_{21}} \frac{1}{\sqrt{2}}(|\boldsymbol{b}\rangle_0 + (-1)^{x_c+1}|\boldsymbol{c}\rangle_0)$

The oracle is applied on $\mathsf{C_1}U_{21}\mathsf{C_1}U_{20}|\gamma_2\rangle$ and on the resultant state, $O_x\mathsf{C_1}U_{21}\mathsf{C_1}U_{20}|\gamma_2\rangle$ we apply $\mathsf{C_1Par^n_{b,c}}$. We observe the evolution for all possible $\{x_b, x_c, x_d\}$ tuples and compare the final state with $(-1)^{x_b}|\boldsymbol{x_b} \oplus \boldsymbol{x_d}\rangle_0 |1\rangle_1 |x_d\rangle_2$. We again list solely the evolution of the query register in Table 3.3, as the other registers remain unchanged.

x_d	$\mathsf{C_1}U_{21}\mathsf{C_1}U_{20}\|\gamma_2\rangle$	$O_x\mathsf{C_1}U_{21}\mathsf{C_1}U_{20}\|\gamma_1\rangle$	$\mathsf{C_1Par^n_{b,c}}O_x$ $\mathsf{C_1}U_{21}\mathsf{C_1}U_{20}\|\gamma_2\rangle$	$\|\boldsymbol{\beta}\rangle$
0	$\frac{1}{\sqrt{2}}\|\boldsymbol{b}\rangle_0$ $+(-1)^{x_c}\frac{1}{\sqrt{2}}\|\boldsymbol{c}\rangle_0$	$\frac{1}{\sqrt{2}}(-1)^{x_b}\|\boldsymbol{b}\rangle_0$ $+\frac{1}{\sqrt{2}}(-1)^{2x_c}\|\boldsymbol{c}\rangle_0$	$(-1)^{x_b}\|\boldsymbol{x_b}\rangle_0$	$(-1)^{x_b}\|\boldsymbol{x_b}\rangle_0$
1	$\frac{1}{\sqrt{2}}\|\boldsymbol{b}\rangle_0$ $+(-1)^{x_c+1}\frac{1}{\sqrt{2}}\|\boldsymbol{c}\rangle_0$	$\frac{1}{\sqrt{2}}(-1)^{x_b}\|\boldsymbol{b}\rangle_0$ $+\frac{1}{\sqrt{2}}(-1)^{2x_c+1}\|\boldsymbol{c}\rangle_0$	$(-1)^{x_b}\|\boldsymbol{x_b} \oplus \boldsymbol{1}\rangle_0$	$(-1)^{x_b}\|\boldsymbol{x_b} \oplus \boldsymbol{1}\rangle_0$

Table 3.3: Evolution of $|\gamma_2\rangle$ and comparison with $|\boldsymbol{\beta}\rangle = (-1)^{x_b}|\boldsymbol{x_b} \oplus \boldsymbol{x_d}\rangle_0$

Therefore in all the cases the state post these transformations is

$$(-1)^{x_b}|\boldsymbol{x_b} \oplus \boldsymbol{x_d}\rangle_0 |0\rangle_0 |x_d\rangle_1 .$$

We now look at the collective effect of the transformations $\mathsf{C_0}U_1$, $\mathsf{C_1}U_2$, O_x, $\mathsf{C_0Par^n_{a,d}}$ and $\mathsf{C_1Par^n_{b,c}}$. The state at start was

$$\frac{1}{\sqrt{2}}\left(|\boldsymbol{x_a}\rangle_0 |0\rangle_1 |x_b\rangle_2 |W_1\rangle + |\boldsymbol{x_c}\rangle_0 |1\rangle_1 |x_d\rangle_2 |W_2\rangle\right).$$

The state after these operations are applied is

$$\frac{1}{\sqrt{2}}\left((-1)^{x_b}|\boldsymbol{x_b} \oplus \boldsymbol{x_d}\rangle_0 |0\rangle_1 |x_b\rangle_2 |W_1\rangle + (-1)^{x_b}|\boldsymbol{x_b} \oplus \boldsymbol{x_d}\rangle_0 |1\rangle_1 |x_d\rangle_2 |W_2\rangle\right).$$

We now apply the operations $\mathsf{C_0CNOT^{Q_n}_{w_2}}$ followed by $\mathsf{CNOT^{w_2}_{Q_n}}$, evolving the system to

$(-1)^{x_b}\frac{1}{\sqrt{2}}\left(|\boldsymbol{x_b}\rangle_0 |0\rangle_1 |x_d\rangle_2 |W_1\rangle + |\boldsymbol{x_b}\rangle_0 |1\rangle_1 |x_d\rangle_2 |W_2\rangle\right)$. and this completes the step. This also shows that for this method the qubit w_2 can be swapped with any other work qubit, and the method is indifferent towards its choice. □

Observe that this subroutine does not depend on the function we are dealing with. However, the advantage is most prominent for the classes of functions that we discuss. Given the general framework of this technique, it is an interesting problem to check if this technique can have applications in other black box problems in the quantum paradigm as well as if this methodology can be further optimized in the bounded error quantum model.

Let us now denote the generalized routine in this regard that form part of the exact quantum query algorithm. This is simply obtained by applying the untangling protocol many times, each time untangling two new qubits. We omit this proof for brevity.

Lemma 9. *Let there be a quantum query algorithm defined on the variables* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ *with* $\hat{\mathbf{x}}$ *and* $\tilde{\mathbf{x}}$ *are two subspaces that form a disjoint partition of* $\mathbf{x}$*, where* $s = 2t$*. Define any two injective maps* $r : [n] \to [n]$ *and* $\hat{r} : [n] \to [n]$ *such that if* $i \leq k$ *then* $r(i), \hat{r}(i) \leq k$ *and* $r(i), \hat{r}(i) > k$ *otherwise. That is,* r *and* $\hat{r}$ *are both combinations of two permutations acting on* $\hat{\mathbf{x}}$ *and* $\tilde{\mathbf{x}}$ *individually. Then, the state*

$$|\beta_0\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\hat{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{i=1}^{s} |x_{r_i}\rangle_{i+1} + (-1)^{h(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{j=1}^{s} |x_{r_{k+j}}\rangle_{j+1} \right)$$

can be evolved to the state $|\beta_f\rangle$ *using the protocol* $\mathsf{untangle}^{\mathsf{s}}_{\mathsf{n}}$*, where,*

$$|\beta_f\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\hat{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 + (-1)^{h(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \right) \bigotimes_{i=1}^{t} \left(\left|x_{\hat{r}(i)}\right\rangle_{2i} \left|x_{\hat{r}(k+i)}\right\rangle_{2i+1} \right).$$

by making t *queries to the oracle* O_x*.*

Using this protocol on the state $|\psi_f\rangle$ described in Equation (3.1) gives us the state using a further t queries:

$$|\psi_{end'}\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\hat{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 + (-1)^{h(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \right) \bigotimes_{i=1}^{t} \left(\left|x_{\hat{r}(i)}\right\rangle_{2i} \left|x_{\hat{r}(k+i)}\right\rangle_{2i+1} \right).$$

Applying a Hadamard gate and then measuring w_1 in the computational basis gives us the output after a total of $k + \lceil \frac{s}{2} \rceil$ queries. The efficiency of the algorithm relies on how well can we partition $\mathbf{x}$ into $\hat{\mathbf{x}}$ and $\tilde{\mathbf{x}}$ and then choose s properly. We now move onto the specific functions and the corresponding algorithms.

Complete Description of Q_{algo} for the pdsp class:

First we establish a common omission that we shall follow, before going into the algorithm.

Remark 4. *From hereon we omit any global phase that occurs in the algorithm for notational simplicity, because of the fact that the global phase does not in anyway effect the evolution of the algorithms.*

Now we describe a simple protocol to obtain the values of two variables parallely into the two different product states, which forms the first phase of our algorithms. This is then followed by the untangling protocol described above.

Lemma 10. *Let $f(\mathbf{x})$ be a Boolean function on n variable which is being evaluated using an algorithm $Q_{algo}(f)$ with the registers Q_n and k qubits of working memory. Let $|W_1\rangle$ and $|W_2\rangle$ be two states defined on the qubits w_2 through w_k.*

Then the state $|\phi\rangle = \frac{1}{\sqrt{2}}\left(|\mathbf{0}\rangle_0 |0\rangle_1 |W_1\rangle + |\mathbf{0}\rangle_0 |1\rangle_1 |W_2\rangle\right)$. can be converted into the state $|\phi_f\rangle = \frac{1}{\sqrt{2}}\left(|\boldsymbol{x_a}\rangle_0 |0\rangle_1 |W_1\rangle + |\boldsymbol{x_b}\rangle_0 |1\rangle_1 |W_2\rangle\right)$. by making a single query to the oracle where $1 \leq a, b \leq n$ where we denote the protocol $\mathsf{get}(a, b)$.

Proof. We show that this transformation can be achieved by using the $\mathsf{get}(a, b)$ transformation defined as the sequential application of the following unitaries and the oracle in the given order:

$$\mathsf{C_0S^n_{0,a}}, \mathsf{C_1S^n_{0,b}}\ O_x,\ \mathsf{C_0Par^n_{0,a}}, \mathsf{C_1Par^n_{0,b}}.$$

The stepwise transformation is as follows.

$$\begin{aligned}
|\phi\rangle =& \frac{1}{\sqrt{2}}\left(|\mathbf{0}\rangle_0 |0\rangle_1 |W_1\rangle + |\mathbf{0}\rangle_0 |1\rangle_1 |W_2\rangle\right) \\
&\xrightarrow{\mathsf{C_0S^n_{0,i}}\ \mathsf{C_1S^n_{0,k+i}}} \frac{1}{\sqrt{2}}\left(\left(\frac{|\mathbf{0}\rangle_0 + |\boldsymbol{i}\rangle_0}{\sqrt{2}}\right)|0\rangle_1 |W_1\rangle + \left(\frac{|\mathbf{0}\rangle_0 + |\boldsymbol{k+i}\rangle_0}{\sqrt{2}}\right)|1\rangle_1 |W_2\rangle\right) \\
&\xrightarrow{O_x} \frac{1}{\sqrt{2}}\left(\left(\frac{|\mathbf{0}\rangle_0 + (-1)^{x_i}|\boldsymbol{i}\rangle_0}{\sqrt{2}}\right)|0\rangle_1 |W_1\rangle + \left(\frac{|\mathbf{0}\rangle_0 + (-1)^{x_{k+i}}|\boldsymbol{k+i}\rangle_0}{\sqrt{2}}\right)|1\rangle_1 |W_2\rangle\right) \\
&\xrightarrow{\mathsf{C_0Par^n_{0,i}}\ \mathsf{C_1Par^n_{0,k+i}}} \frac{1}{\sqrt{2}}\left(|\boldsymbol{x_a}\rangle_0 |0\rangle_1 |W_1\rangle + |\boldsymbol{x_b}\rangle_0 |1\rangle_1 |W_2\rangle\right) = |\phi_f\rangle\,.
\end{aligned}$$

□

We now start describing our complete exact quantum query algorithm, starting with $f_1(\mathbf{x}) == \prod_{i=1}^{k} x_i \bigoplus \prod_{j=k+1}^{n} x_j$. Then we describe the modifications needed to cover the targeted set of functions from the pdsp class.

This coupled with the generalized parity decision tree complexity of the function provides the first separation result.

Theorem 13. *For the function* $f_1 = \prod_{i=1}^{\frac{n}{2}} x_i \oplus \prod_{j=\frac{n}{2}+1}^{n} x_j$, *we have* $QC_{algo}(f_1) = \lfloor \frac{3n}{4} \rfloor$ *and* $D_{\oplus}(f_1) = n - 1$.

Proof.

$D_{\oplus}(f) = n - 1$: This is a direct implication of Proposition 8 where $f \in \mathsf{pdsp}(n, n, 2)$

$QC_{algo}(f) = \lfloor \frac{3n}{4} \rfloor$:

We run Algorithm 1 initializing it in the state $|\mathbf{0}\rangle_0 \otimes_{i=1}^{k} |0\rangle_i$. This algorithm makes a total of $\frac{n}{2} + \lfloor \frac{n}{4} \rfloor = \lfloor \frac{3n}{4} \rfloor$ queries, which completes the proof.

□

For f_1 we are able to separate $QC_{algo}(f)$ and $D_{\oplus}(f)$, but the algorithm is not provably optimal for this function. However we observe that this technique is indeed optimal for the following function.

Corollary 2. *For the function* f_2 *on* $n = 2k$ *variables where* $f_2(\mathbf{x}) = \prod_{i=1}^{\lfloor \frac{3n}{4} \rfloor} x_i \oplus \prod_{j=\frac{n}{2}+1}^{n} x_j$ *we have* $QC_{algo}(f_2) = Q_E(f_2) = \lfloor \frac{3n}{4} \rfloor$ *and* $D_{\oplus}(f_2) = n - 1$.

Proof.

$Q_E(f) \geq \lfloor \frac{3n}{4} \rfloor$ We can reduce f_1 to $\mathrm{AND}_{\lfloor \frac{3n}{4} \rfloor}$ by fixing the variables $x_i = 0, \lfloor \frac{3n}{4} \rfloor + 1 \leq i \leq n$, and therefore evaluating f must take at least $\lfloor \frac{3n}{4} \rfloor$ queries as we know $Q_E(\mathrm{AND}_{\lfloor \frac{3n}{4} \rfloor}) = \lfloor \frac{3n}{4} \rfloor$.

$QC_{algo}(f) = \lfloor \frac{3n}{4} \rfloor$ This function can in fact be written as

Algorithm 1 $Q_{algo}(f)$ to evaluate $f(\mathbf{x}) = \prod_{i=1}^{\frac{n}{2}} x_i \oplus \prod_{j=\frac{n}{2}+1}^{n} x_j$ along with query complexity count $(QC_{algo}(f))$:

1 Begin with the state $|\mathbf{0}\rangle_0 \bigotimes_{i=1}^{k} |0\rangle_i$, consisting of the Query register and $\frac{n}{2}$ work qubits $w_i, 1 \leq i \leq \frac{n}{2}$.

2 We apply a Hadamard to the first work qubit w_1 to get $|\psi_0\rangle = \frac{1}{\sqrt{2}} \left(|\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{i=2}^{k} |0\rangle_i + |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{i=2}^{k} |0\rangle_i \right)$.

3 We do the following steps for $1 \leq i \leq \frac{n}{2} - 1$.

   i Apply $\mathsf{get}\left(i, \frac{n}{2}\right)$

   ii Swap the value of Q_n with the $i+1$-th work qubit w_{i+1} by applying $\mathsf{CNOT}_{\mathsf{Q_n}}^{\mathsf{W_{i+1}}}$ followed by $\mathsf{CNOT}_{\mathsf{Q_n}}^{\mathsf{W_{i+1}}}$.

   Each application of $\mathsf{get}()$ consists of one query these steps the state from $|\psi_0\rangle$ to $|\psi_{\frac{n}{2}-1}\rangle$, where

$$|\psi_i\rangle = \frac{1}{\sqrt{2}} \left(|\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{j=2}^{i} |x_{j-1}\rangle_j \bigotimes_{j=i+1}^{k} |0\rangle_j + |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{j=2}^{i} |x_{k+j-1}\rangle_j \bigotimes_{j=i+1}^{k} |0\rangle_j \right).$$

4 Here let us define $g(\hat{\mathbf{x}}) = \prod_{i=1}^{\frac{n}{2}} x_i$ and $h(\tilde{\mathbf{x}}) = \prod_{j=1}^{\frac{n}{2}} x_{\frac{n}{2}+j}$. Apply an $k = 1$ controlled CNOT operation C^{k-1} with Q_n as the target and $w_2, \ldots w_k$ as the controls. This converts the state to

$$\frac{1}{\sqrt{2}} \left(\left| \prod_{\boldsymbol{i=1}}^{\boldsymbol{k-1}} \boldsymbol{x_i} \right\rangle_0 |0\rangle_1 \bigotimes_{j=2}^{i} |x_{j-1}\rangle_j \bigotimes_{j=i+1}^{k} |0\rangle_j + \left| \prod_{\boldsymbol{i=1}}^{\boldsymbol{k-1}} \boldsymbol{x_{k+i}} \right\rangle_0 |1\rangle_1 \bigotimes_{j=2}^{i} |x_{k+j-1}\rangle_j \bigotimes_{j=i+1}^{k} |0\rangle_j \right).$$

5 Apply $\mathsf{P}_{\frac{\mathsf{n}}{2}}^{\mathsf{n}}$ controlled on $w_1 = |0\rangle$ and $\mathsf{P}_{\mathsf{n}}^{\mathsf{n}}$ controlled on $w_1 = |1\rangle$ followed by a call to an oracle and then another application of C^{k-1} to restore Q_n to $|\mathbf{0}\rangle$ state. Then after $\frac{n}{2}$ queries the system is in the state

$$|\psi_{\frac{n}{2}}\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\hat{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{j=2}^{k} |x_{j-1}\rangle_j + (-1)^{h(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{j=2}^{k} |x_{k+j-1}\rangle_j \right).$$

6 We then apply the transformation $untangle_n^s$ described in Lemma 9 where $s = \frac{n}{2} - 1$. This step requires a further $\lfloor \frac{n}{4} \rfloor$ queries, and finally the system is in the state

$$|\beta_f\rangle = (-1)^{g'(\mathbf{x})} |\mathbf{0}\rangle_0 |g(\hat{\mathbf{x}}) \oplus h(\tilde{\mathbf{x}})\rangle_1 \bigotimes_{i=1}^{k_1} (|x_{2i}\rangle_{i+1} |x_{k+2i}\rangle_{i+2})$$

   after a total of $\lfloor \frac{3n}{4} \rfloor$ queries.

6 Get the output by then measuring w_1 in the computational basis.

$$f(\mathbf{x}) = \left(\prod_{i=1}^{\frac{n}{2}} x_i \bigoplus \prod_{i=\frac{n}{2}}^{n} x_i\right) \prod_{j=\frac{n}{2}+1}^{\lfloor\frac{3n}{4}\rfloor} x_j.$$

We proceed in the same direction as Theorem 13. After $\frac{n}{2}$ queries the system is in the state

$$|\psi_{\frac{n}{2}}\rangle = \frac{1}{\sqrt{2}}\left((-1)^{g(\hat{\mathbf{x}})}|\mathbf{0}\rangle_0|0\rangle_1 \bigotimes_{j=2}^{k}|x_{j-1}\rangle_j + (-1)^{h(\tilde{\mathbf{x}})}|\mathbf{0}\rangle_0|1\rangle_1 \bigotimes_{j=2}^{k}|x_{k+j-1}\rangle_j\right).$$

Now only apply the $\mathsf{untangle}^{\mathsf{s}}_{\mathsf{n}}$ in such a way that the values of the variables $x_{\frac{n}{2}+1}, \ldots x_{\lfloor\frac{3n}{4}\rfloor}$ are carried over so that w_{2i+2} is in the state $\left|x_{\frac{n}{2}+i}\right\rangle_{2i+2}$. Thus the system's state after $\lfloor\frac{3n}{4}\rfloor$ queries is

$$|\mathbf{0}\rangle_0 \frac{1}{\sqrt{2}}\left((-1)^{g(\hat{\mathbf{x}})}|0\rangle_1 + (-1)^{h(\tilde{\mathbf{x}})}|\mathbf{0}\rangle_0|1\rangle_1\right)\bigotimes_{j=1}^{s}|x_{j-1}\rangle_{2j}|x_{k+j-1}\rangle_{2j+1}.$$

We can apply a Hadamard to obtain the state $|\mathbf{0}\rangle_0|g(\hat{\mathbf{x}}) \oplus h(\tilde{\mathbf{x}})\rangle \bigotimes_{j=1}^{s}|x_{j-1}\rangle_{2j}|x_{k+j-1}\rangle_{2j+1}$.

We can now obtain the value of $f(\mathbf{x})$ as we obtain the value $g(\hat{\mathbf{x}}) \oplus h(\tilde{\mathbf{x}}) = \left(\prod_{i=1}^{\frac{n}{2}} x_i \bigoplus \prod_{i=\lfloor\frac{3n}{4}\rfloor+1}^{n} x_i\right)$ by measuring w_1 and the variables variable $x_{\frac{n}{2}}, \ldots x_{\lfloor\frac{3n}{4}\rfloor}$. measuring the qubits of the form w_{2i+2}.

□

Theorem 14. *Let $f \in \mathsf{pdsp}(\mathsf{n}, \lceil\frac{3\mathsf{n}}{4}\rceil, \mathsf{t}+1)$ be a function on $n = 2k$ variables such that*

$$f(\mathbf{x}) = \left(\prod_{i=1}^{\frac{n}{2}} x_i \bigoplus g(\mathbf{x}')\right)\left(\prod_{j=\frac{n}{2}+1}^{\lfloor\frac{3n}{4}\rfloor} x_j\right),\ \mathbf{x}' = \left(x_{\lfloor\frac{3n}{4}\rfloor+1}, x_{\lfloor\frac{3n}{4}\rfloor+2}, \ldots, x_n\right).$$

where g is perfect direct sum function defined on $\left(x_{\lfloor\frac{3n}{4}\rfloor+1}, x_{\lfloor\frac{3n}{4}\rfloor+2}, \ldots, x_n\right)$ so that it contains t monomials such that each monomial consists of at least $t+1$ variables. Then we have (i) $QC_{algo}(f) = Q_E(f) = \lfloor\frac{3n}{4}\rfloor$, (ii) $D_{\oplus}(f) = n - t$, (iii) $D(f) = n$.

Proof.

$Q_E(f) \geq \lfloor\frac{3n}{4}\rfloor$ For any such function, if we fix the variables $x_i, \lfloor\frac{3n}{4}\rfloor + 1 \leq i \leq n$ to 0 then the function is reduced to $\mathrm{AND}_{\lfloor\frac{3n}{4}\rfloor}$ which implies $Q_E(f) \geq \lfloor\frac{3n}{4}\rfloor$.

$D_{\oplus}(f) = n - t$ This is a direct implication of Proposition 8 where the number of monomials is $t + 1$.

$QC_{algo}(f) = \lfloor \frac{3n}{4} \rfloor$ We initialize the algorithm in the state $|\mathbf{0}\rangle_0 \bigotimes_{i=1}^{k+1} |0\rangle_i$. We first apply a Not gate and a Hadamard gate to w_{k+1} to get $|\mathbf{0}\rangle_0 \bigotimes_{i=1}^{k} |0\rangle_i |-\rangle$. We proceed in the same way as Algorithm 1 till step 3 and w_{k+1} is not modified. This evolves the state after $\frac{n}{2} - 1$ queries to

$$\frac{1}{\sqrt{2}} \left(|\mathbf{0}\rangle_0 |0\rangle_1 \left(\bigotimes_{i=2}^{k} |x_{i-1}\rangle_i \right) + |\mathbf{0}\rangle_0 |1\rangle_1 \left(\bigotimes_{i=2}^{k} |x_{k+i-1}\rangle_i \right) \right) |0\rangle_{k+1} |-\rangle_{k+2} .$$

The only difference in the functions in this class from f_2 is that instead of $\prod_{i=\lfloor \frac{3n}{4} \rfloor + 1}^{n} x_i$ now we have have perfect direct sum of variable disjoint monomials on the variables $x_{\lfloor \frac{3n}{4} \rfloor + 1}$ to x_n. However all, but x_n of these variable's value is now in the state. Let the monomial of which x_n is a part of be $m(\tilde{\mathbf{x}}) = \prod_{i \in \Gamma} x_i$ where $\Gamma \in [n]$ and $n \in \Gamma$.
So we do the following, we apply a $k-1$ controlled not gate on Q_n controlled on $w_1 = |0\rangle$. We apply another multi controlled gate on Q_n controlled on $w_1 = |1\rangle$ so that all the qubits containing the values of $x_i, i \in \Gamma$ is used as a control.

Then we apply the oracle and again the controlled operations and after $\frac{n}{2}$ queries have the state

$$|\psi_{\frac{n}{2}}\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{j=2}^{k} |x_{j-1}\rangle_j + (-1)^{m,(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{j=2}^{k} |x_{k+j-1}\rangle_j \right) |-\rangle_{k+1} .$$

Now to obtain the perfect direct sum of the other monomials with $m(\tilde{\mathbf{x}})$ we simply make multi controlled not operations controlled on $w_1 = |1\rangle$ on the $|-\rangle_{k+1}$ state and this just adds local phase to $m(\tilde{\mathbf{x}})$ because of the well known phase back protocol. These protocols do not take any queries and after obtaining the direct sum of all the monomials we are in the state

$$|\psi_{\frac{n}{2}}\rangle = \frac{1}{\sqrt{2}} \left((-1)^{g(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{j=2}^{k} |x_{j-1}\rangle_j + (-1)^{h(\tilde{\mathbf{x}})} |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{j=2}^{k} |x_{k+j-1}\rangle_j \right) |-\rangle_{k+1} .$$

From here on the algorithm proceeds identically as Corollary 2, and after applying $\mathsf{untangle}_{\mathsf{n}}^{\mathsf{k}}$ and a total of $\lfloor \frac{3n}{4} \rfloor$ queries the algorithm is in the state $|\mathbf{0}\rangle_0 |g(\tilde{\mathbf{x}}) \oplus h(\tilde{\mathbf{x}})\rangle \bigotimes_{j=1}^{s} |x_{j-1}\rangle_{2j} |x_{k+j-1}\rangle_{2j+1} |-\rangle_{k+1}$.

We can now obtain the requisite function by simply obtaining the product $\prod_{i=k=1}^{\lfloor \frac{3n}{4} \rfloor} x_i$. This

completes the proof.

□

The number of functions covered by the class, referred in Theorem 14, is as follows.

Corollary 3. *For any n there are $\Omega\left(2^{\frac{\sqrt{n}}{2}}\right)$ functions (without considering permutation of variables) for which we can obtain $Q_{algo}(f) = Q_E(f) < D_{\oplus}(f)$.*

Proof. We give a lower bound on number of functions which satisfy the constraints of the function described in Theorem 14. Without considering the permutation of variables, we can simply count the number of ways the function $g(\mathbf{x}')$ can be constructed. The function g is defined on $\lceil \frac{n}{4} \rceil$ variables and is it self a perfect direct sum function as defined in Definition 6. If g contains t monomials then then each of the monomial must have at least $t+1$ variables in them. This is because $\prod_{i=1}^{k} x_i \bigoplus g(\mathbf{x}')$ must satisfy the constraints of Definition 6. Therefore each construction of g is a different way of partitioning the $\lceil \frac{n}{4} \rceil$ variables into t sets. If we do not consider which variable is in which monomial, and rather just the distribution of the variables in the partitions, then this becomes same as finding the number of solutions to $\sum_{i=1}^{t} v_i = \lceil \frac{n}{4} \rceil$ where $v_i \geq t+1 \ \forall i$. We do this is as it is well known that if a function is derived from some other function just by a permutation of the variable names, they have the same query complexity and are called PNP equivalent [12]. We aim to count the functions that are not PNP equivalent with each other. The number of such partitions is $\binom{n+t-(t+1)^2-1}{t-1} = \binom{\lceil \frac{n}{4} \rceil - t^2 - t - 1}{t-1}$. Here t is minimum 1 and at maximum $\sqrt{\lceil \frac{n}{4} \rceil - 1}$. Therefore the total possible number of function is

$$\left(\sum_{x=1}^{\sqrt{\lceil \frac{n}{4} \rceil - 1}} \binom{\lfloor \frac{n}{4} \rfloor - x^2 - x - 1}{x-1} \right) > \left(\sum_{x=1}^{\sqrt{\lceil \frac{n}{4} \rceil - 1}} \binom{\sqrt{\lceil \frac{n}{4} \rceil - 1}}{x} \right) = \Omega\left(2^{\sqrt{\frac{n}{4}}}\right) = \Omega\left(2^{\frac{\sqrt{n}}{2}}\right).$$

□

Again, note that the advantage is from being able to deterministically untangle two qubits with a single query, owing to the result of Theorem 12 and the fact that these functions have high granularity.

The next important fact is in untangling we have a degree of freedom in terms of which variables we want to carry over to the end, and then their values can again be deterministically to obtain other monomials. In fact in the state $|\beta_0\rangle$, if there are s variables each whose values are stored in

the two superposition state, we can carry over $\lceil \frac{s}{2} \rceil$ values from each of the superposition states to the final state that is simply a tensor product of qubits in computational basis states, meaning the value of all the variables stored in the working memory can be deterministically obtained.

This is evident from the structure of the state that we obtain at the end for $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ (For f_2 we have already decided on which values from $x_i, \frac{n}{2} \leq i \leq n$ we want to carry over to the final, pre-measurement state):

$$|\beta_f\rangle = (-1)^{g'(\mathbf{x})} |\mathbf{0}\rangle_0 \left| \prod_{i=1}^{k} x_i \oplus \prod_{i=k+1}^{n} x_i \right\rangle_1 \bigotimes_{i=1}^{k_1} (|x_{r(2i)}\rangle_{i+1} |x_{k+i}\rangle_{i+2}).$$

The algorithm for the other functions progresses in a similar manner. This coupled with the fact that the pdsp class has high granularity, which allows us to efficiently lower bound the generalized parity tree complexity gives us the desired advantage.

3.4 Conclusion and Future Directions

In this document we have designed a new family of exact quantum algorithms (Q_{algo}) for certain classes of non-symmetric functions f with query complexity $QC_{algo}(f)$.

First we have described the class $\mathsf{pdsp}(n, \lceil \frac{3n}{4} \rceil, q)$ using perfect direct sum constructions with products, and shown that for a set of $\Omega(2^{\frac{\sqrt{n}}{2}})$ functions in this class we get $Q_E(f) = QC_{algo}(f) = \lfloor \frac{3n}{4} \rfloor$ with $D_{\oplus}(f) > \lfloor \frac{3n}{4} \rfloor$. For these set of functions we have $\lfloor \frac{3n}{4} \rfloor + 1 \leq D_{\oplus}(f) \leq n - 1$, depending on the value of q in $\mathsf{pdsp}(n, \lceil \frac{3n}{4} \rceil, q)$. We have obtained this result by designing exact quantum query algorithms based on $\mathbb{F}_2$ polynomial structure and then proven separation from generalized parity complexity technique by exploiting the high granularity of these functions.

In this regard we design a subroutine as described in Theorem 10 which un-entangles two qubits in an entangled system with a single query, which allows us to obtain the said separations and is central to our algorithms. It would be interesting to study if this subroutine can be modified to be more efficient in the bounded error quantum query model.

In fact, we not only obtain advantage over the parity decision tree model in which the parity of two bits is calculated in a single query, but also the stronger generalized parity decision tree model in which parity of any number of bits can be calculated in a single query. It remains of interest to understand the extent to which these techniques can be applied and how can they be modified to

get optimal query complexity for other classes of Boolean functions, towards better understanding of this domain.

Bibliography

[1] A. Ambainis. Superlinear advantage for exact quantum algorithms. In Proceedings of the forty-fifth annual ACM symposium on Theory of Computing (STOC'13), 891-900 (2013). Arxiv: https://arxiv.org/abs/1211.0721

[2] A. Ambainis, J. Iraids and D. Nagaj. Exact Quantum Query Complexity of $\mathsf{EXACT}^{n}_{k,l}$. SOFSEM 2017: Theory and Practice of Computer Science, 243-255 (2016).

[3] A, Ambainis, J. Iraids and J. Smotrovs. Exact quantum query complexity of EXACT and THRESHOLD. Proceedings of the 8th conference on the theory of quantum computation, communication, and cryptography (TQC'13), pp 263–269. Arxiv: https://arxiv.org/abs/1302.1235

[4] A. Ambainis, K. Balodis, A. Belovs, T. Lee, M. Santha and J. Smotrovs. Separations in Query Complexity Based on Pointer Functions. Journal of the ACM, DOI: https://doi.org/10.1145/3106234 (2017). Arxiv Version: https://arxiv.org/abs/1506.04719 (2015).

[5] H. Barnum, M. Saks and M. Szegedy. Quantum query complexity and semi-definite programming. In proceedings of 18th IEEE Annual Conference on Computational Complexity, pp. 179-193, (2003).

[6] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca and Ronald de Wolf. Quantum lower bounds by polynomials. J. ACM 48, 4 (July 2001), 778–797. DOI: https://doi.org/10.1145/502090.502097.

[7] T. Cusick and P. Stanica. Cryptographic Boolean Functions and Applications. Academic Press, Elsevier (2009).

[8] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. In Proceedings of Royal Society London, vol. 439, issue 1907, pp. 553–558, DOI: https://doi.org/10.1098/rspa.1992.0167, 1992.

[9] J. F. Dillon. Elementary Hadamard Difference sets. Ph.D. Dissertation, Univ. of Maryland (1974).

[10] M. Grant and S. Boyd. CVH: Matlab software for disciplined convex programming, version 1.21, (2011). http://cvxr.com/cvx

[11] L. K. Grover. A fast quantum mechanical algorithm for database search. In Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing (STOC '96), Association for Computing Machinery, New York, NY, USA, 212–219 (1996).

[12] A. Montanaro, R. Jozsa and G. Mitchison. On Exact Quantum Query Complexity. Algorithmica 71, 775–796 (2015).

[13] C. S. Mukherjee and S. Maitra. Classical-Quantum Separations in Certain Classes of Boolean Functions– Analysis using the Parity Decision Trees. Arxiv: https://arxiv.org/abs/2004.12942 (2020).

[14] C. S. Mukherjee and S. Maitra. Exact Quantum Query Algorithms Outperforming Parity – Beyond The Symmetric functions (Extended Version) Arxiv: https://arxiv.org/abs/2008.06317

[15] M. A. Nielsen and I. L. Chuang. Quantum Computation and Quantum Information. 10th Anniversary Edition, Cambridge University Press, January 2011.

[16] V. V. Podolskii and A. Chistopolskaya, Parity Decision Tree Complexity is Greater Than Granularity, https://arxiv.org/abs/1810.08668 (2018).

[17] D. R. Simon. On the Power of Quantum Computation. SIAM Journal on Computing, vol. 26, no. 5, pp. 1474–1483, October 1997, DOI: https://doi.org/10.1137/S0097539796298637.

[18] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comput. 26, 5, 1484–1509 (1997).

Chapter 4

Results on Maiorana-McFarland Type Bent Functions.

4.1 Introduction

Now we focus on the first kind of functions described in the problem statement for $n = 2k$ in Chapter 3. The ANF has k monomials, each with degree 2 and each variable appears only once in the ANF. A first observation is that this function in fact is an MM type bent function. This class of functions is defined as follows.

Definition 7 (MM type functions)**.** *For any two positive integers n_1 and n_2 with $n_1 + n_2 = n$, An MM Boolean function on $\mathbb{F}_2^n$ is defined as*

$$f(\hat{\mathbf{x}}, \tilde{\mathbf{x}}) = \phi(\hat{\mathbf{x}}) \cdot \tilde{\mathbf{x}} \oplus g(\hat{\mathbf{x}})$$

where the subspaces are $\hat{\mathbf{x}} \in \mathbb{F}_2^{n_1}$, $\tilde{\mathbf{x}} \in \mathbb{F}_2^{n_2}$, g is any Boolean function defined on $F_2^{n_1}$ and ϕ is a map of the form $\phi : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$.

Here we assume $\frac{n}{2} = k$ and $\hat{\mathbf{x}} = (x_1, \ldots, x_k)$ and $\tilde{\mathbf{x}} = (x_{k+1}, \ldots x_n)$.

Here $a \cdot b$ is the dot product of two n_2 dimensional vectors, defined as $a \cdot b = a_1 b_1 \oplus a_2 b_2 \ldots a_{n_2} b_{n_2}$. If we set $n_1 = n_2$ and restrict ϕ to be a bijective map, then all resultant Boolean functions are bent and are called as MM type bent functions. These are the functions with highest possible nonlinearity for any even n [1]. We denote this class of MM Bent functions by $\mathcal{B}_n$. There are $2^{2^{\frac{n}{2}}}(2^{\frac{n}{2}}!)$ functions in this class and the algebraic degree of the functions in this class vary between 2 and $\frac{n}{2}$. The MM type Bent functions and its different modifications have extensive applications in cryptographic primitives and in coding theory [5]. Let us now layout the organization of this chapter.

In section 4.2 we study the deterministic query complexity of this class. We shall see that $D(f) = n$ for all $f \in \mathcal{B}_n$. Next we shall design a $\lfloor \frac{3n}{4} \rfloor$-query parity decision tree for any function in $\mathcal{B}_n$.

Then in Section 4.3 we design a $\lceil \frac{5n}{8} \rceil$-query exact quantum algorithm for the function f_n^{id} using the results and techniques of chapter 3. Next we modify the techniques so it can give us a $\lceil \frac{5n}{8} \rceil$-query exact quantum algorithm for a subclass of $\mathcal{B}_n$ of size approximately $2^{2^{\lfloor \frac{n}{4} \rfloor}}$ functions. We also extend this result for odd values of n by adding an extra variable linearly to a candidate MM type bent function on $n - 1$ variables.

4.2 On Decision and Parity Decision Tree Complexity

We first calculate the deterministic query complexity of any function in the $\mathcal{B}_n$. Given a point $\alpha \in \{0,1\}^{\frac{n}{2}}$ we define the point $\alpha^{(i)}, 1 \leq i \leq \frac{n}{2}$ as follows.

$$1 \leq j \leq \frac{n}{2},\ j \neq i\ :\ \alpha^{(i)} = \alpha_j$$
$$j = i\ :\ \alpha_j^{(i)} = \overline{\alpha_j}$$

We also define the points $A^1, A^0 \in 0, 1^{\frac{n}{2}}$ so that $A_i^1 = 1\ \forall i$ and $A_i^0 = 1\ \forall i$.

Theorem 15. *The deterministic query complexity of any function in $\mathcal{B}_n$ is n.*

Proof. Let us assume that there exists a deterministic decision tree D that queries $n-1$ variables to evaluate a function $f \in \mathcal{B}_n$ in the worst case. Let $D(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ denote the output obtained using the Deterministic tree with $D(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ as the input. This means the longest root to leaf vertex contains $n-1$ internal nodes (queries).

We consider a point $y \in \{0,1\}^{\frac{n}{2}}$ such that $\phi(y) = A^1$. Then $f(y,z) = x_{k+1} \oplus \ldots x_n \oplus h(y)$ for all $z \in \{0,1\}^{\frac{n}{2}}$. Therefore at any point (y,z) any deterministic decision tree (algorithm) has to query all $\frac{n}{2}$ bits of z to evaluate the function correctly.

Now if a decision tree doesn't query a variable $x_i \in \hat{\mathbf{x}}$ at a point (y,z) Then the decision tree traversal for the points (y,z) and $(y^{(i)},z)$ will be identical, so that $D(y,z) = D(y^{(i)},z)\ \forall z \in \{0,1\}^{\frac{n}{2}}$.

But weight of $(\phi y^{(i)})$ is at most $n-1$. This implies for any point $(y^{(i)},z)$ there is at least an index $1 \leq k \leq n$ such that $f(y^{(i)},z) = f(y^{(i)},z^{(k)})$.

However we know from the definition that $f(y,z) \neq f(y,z^{(i)})$. This contradicts the claim that a deterministic decision tree can evaluate a function $f \in B_n$ with $n-1$ queries in the worst case, and thus we have $D(f) = n$. □

Now we observe how parity decision tree can be used to form a quantum algorithm which can always evaluate a function in $\mathcal{B}_n$ with less than n queries.

We first provide a very simply derivable quantum advantage using parity decision trees.

Lemma 11. *Given any function $f \in \mathcal{B}_n$ we have $Q_E(f) \leq \lceil \frac{3n}{4} \rceil$.*

Proof. We prove this by describing an algorithm that can evaluate any Boolean function in of the type $\mathcal{B}_n$ using $\lceil \frac{3n}{4} \rceil$ queries.

The queries made by this quantum algorithm are of the form x_i or $x_{i_1} \oplus x_{i_2}$ and can therefore be expressed as a parity decision tree.

Given any input (y, z) where $y = (x_1, \ldots, x_k) = (y_1, \ldots, y_k)$ and $z = (x_{k+1}, \ldots, x_n) = (z_1, \ldots, z_k)$ the algorithm first queries the $\frac{n}{2}$ variables $x_1, x_2, \ldots, x_n$. Then depending on the definition of the function it does one of the following two tasks.

1. If $h(y) = 0$ then it evaluates $\bigoplus_{\phi(y)_i=1} z_i$
2. If $h(x) = 1$ then it evaluates $\Big(\bigoplus_{\phi(y)_i=1} z_i \Big) \oplus 1$

In either case this requires $\lceil \frac{wt(\phi(y))}{2} \rceil$ queries and therefore at max requires $\lceil \frac{n}{4} \rceil$ queries. This proves the upper bound. □

Now we obtain a lower bound on the exact quantum query complexity of the functions in $\mathcal{B}_n$.

Lemma 12. *The degree of the real polynomial corresponding to any function in $\mathcal{B}_n$ is n.*

Proof. The polynomial corresponding to the function $f \in \mathcal{B}_n$ such that $f(\hat{\mathbf{x}}, \tilde{\mathbf{x}}) = \phi(\hat{\mathbf{x}}) \cdot \tilde{\mathbf{x}} \oplus h(\hat{\mathbf{x}})$ can be formulated as follows. We observe that only one of the linear function defined on the variables $\tilde{\mathbf{x}}$ is evaluated for any input $(\hat{\mathbf{x}}, \tilde{\mathbf{x}})$ depending on the value of $\phi(\hat{\mathbf{x}})$. Therefore we first form the following product terms on the variables $\{x_1, x_2, \ldots, x_k\}$. We define the $\mathcal{P}_a, a \in \{0,1\}^{\frac{n}{2}}$ as

$$\mathcal{P}_a = \prod_{a_i=0} (1 - x_i) \prod_{a_i=1} x_i.$$

$\mathcal{P}_a$ evaluates to 1 iff $\hat{\mathbf{x}} = a$, 0 otherwise. Now we append the corresponding linear functions defined by $\phi(a)$ to each of these product terms. We also account for the function $h(x)$ which evaluates to $h(a)$ for any input (a, y).

Therefore the linear function to be evaluated is $\Big(\bigoplus_{\phi(a)_i=1} x_{k+i} \Big) \oplus h(a)$, which is represented as

$$\mathcal{L}_a = \frac{\Big(\prod_{\phi(a)_i=1} (2x_{k+i} - 1) \Big) (-1)^{h(a)} + 1}{2}$$

Therefore we have the polynomial $p(\mathbf{x})$ corresponding to the function f as

$$p(\mathbf{x}) = \sum_{a \in \{0,1\}^n} \mathcal{P}_a \mathcal{L}_a.$$

Therefore by definition we have $\deg^{\mathbb{R}}(\mathcal{L}_a) = wt(a)$ and $\deg^{\mathbb{R}}(\mathcal{P}_a) = \frac{n}{2}, \forall a$ and since there is only one value of a with $wt(a) = \frac{n}{2}$ this implies $\deg^{R}(p) = n$.

This polynomial is defined as $p : \mathcal{R}^n \to R$ but its range becomes $\{0, 1\}$ when the domain is restricted to $\{0, 1\}^n$. □

This proof is also another way of showing that the Deterministic Query complexity of any function in $\mathcal{B}_n$ is n.

Combining Lemma 11 and Lemma 12 we obtain the following result.

Theorem 16. *For any MM Bent function $f \in \mathcal{B}_n$, we have $\frac{n}{2} \leq Q_E(f) \leq \frac{3n}{4}$.*

The statement of Theorem 16 gives rise to the following corollary.

Corollary 4. *For all values of n there are two or more MM Bent functions that have different algebraic degree and same exact quantum query complexity.*

Proof. The algebraic degree of the functions in $\mathcal{B}_n$ vary between 2 and $\frac{n}{2}$ where as the exact quantum query complexity varies between $\frac{n}{2}$ and $\lceil \frac{3n}{4} \rceil$.

Therefore applying pigeonhole principle it is easy to see that there are at least two Boolean functions with different algebraic degree and same exact quantum query complexity. □

It is important to note that two functions in $\mathcal{B}_n$ may have the same algebraic degree yet different exact quantum query complexity. Characterizing these equivalence classes for $\mathcal{B}_n$ appears to be a very interesting problem. It is also interesting to observe that the real polynomial corresponding to any function in $\mathcal{B}_n$ can be obtained from the description of the corresponding parity decision tree.

We further observe the exact quantum query complexity of functions in $\mathcal{B}_4$ which gives us more insight into this problem. The different Boolean functions in the class $\mathcal{B}_4$ upto isomorphism are the following.

$$f_1(x) = x_1x_2 \oplus x_3x_4$$

$$f_2(x) = x_1x_2 \oplus x_3x_4 \oplus x_2x_3$$

The exact quantum query complexity of all 4 variable MM Bent functions can be observed from [3, Table A.1]. In this regard we observe that $Q_E(f_1) = Q_E(f_2) = 3$ which touches the upper bound of $\lceil \frac{3n}{4} \rceil$ in this case. However we observed that $Q_E(f^i d_6) = 4$ using the SDP formulation whereas the parity decision tree complexity due to our algorithm is 5. This motivated us to look beyond and we obtained a new algorithm using our $\mathbb{F}_2$-polynomial based methods.

4.3 A Novel Exact Quantum Query Algorithm for a Subclass of $\mathcal{B}_n$

We proceed in an almost identical fashion as we did for the pdsp class, creating equal superposition of two product state, parallely obtaining values of variables and then untangling the system to obtain the output. However although we obtain a $\lceil \frac{5n}{8} \rceil$-query exact quantum algorithm for a set of $\Omega\left(2^{2^{\frac{n}{4}}}\right)$ functions, we are not yet able to prove either the optimality of this algorithm or whether this is lower than parity decision tree complexity $D^{(2)}_{\oplus}(f)$, which remain two challenging open problems.

Here we shall assume that $n \equiv 0 \bmod 4$ for simplicity of calculation. The results remain same for $n \equiv 2 \bmod 4$ (bent functions are only defined for even n) but we leave the modifications out due to tediousness.

We first design a protocol similar to $\mathsf{get}(a, b)$ in Lemma 9 of Chapter 3 and then describe the $\lceil \frac{5n}{8} \rceil$-query algorithm.

Lemma 13. *Let $f(\mathbf{x})$ be a Boolean function on n variable which is being evaluated using an algorithm $Q_{algo}(f)$ with the registers Q_n and k qubits of working memory. Let $|W_1\rangle$ and $|W_2\rangle$ be two states defined on the qubits w_2 through w_k. Then the state $|\phi\rangle = \frac{1}{\sqrt{2}} \left(|\mathbf{0}\rangle_0 |0\rangle_1 |W_1\rangle + |\mathbf{0}\rangle_0 |1\rangle_1 |W_2\rangle \right)$. can be converted into $|\phi_f\rangle = \frac{1}{\sqrt{2}} \left((-1)^{x_a x_b} |\boldsymbol{x_b}\rangle_0 |0\rangle_1 |W_1\rangle + (-1)^{x_c x_d} |\boldsymbol{x_d}\rangle_0 |1\rangle_1 |W_2\rangle \right)$ by making a two queries to the oracle where $1 \leq a, b, c, d \leq n$ where we denote the protocol $\mathsf{get}(a, b, c, d)$.*

Proof. We first apply $\mathsf{get}(b, d)$ that consists of a single query to obtain $\frac{1}{\sqrt{2}} \left(|\boldsymbol{x_b}\rangle_0 |0\rangle_1 |W_1\rangle + |\boldsymbol{x_d}\rangle_0 |1\rangle_1 |W_2\rangle \right)$. Next we apply $\mathsf{C_0P_a^n}$ and $\mathsf{C_1P_c^n}$ then make a query to the oracle, and use the permutation matrices with the same controls. It is easy to see that this evolves the state to

$\frac{1}{\sqrt{2}} \left((-1)^{x_a x_b} |\boldsymbol{x_b}\rangle_0 |0\rangle_1 |W_1\rangle + (-1)^{x_c x_d} |\boldsymbol{x_d}\rangle_0 |1\rangle_1 |W_2\rangle \right)$ which is the desired state using a total of two queries.

□

The algorithm for f_n^{id} is same as that of the perfect direct sum function with only minimal changes. Specifically, $\mathsf{get}(a,b)$ is replaced by $\mathsf{get}(a,b,c,d)$ and after $\frac{n}{2}$ queries you have to untangle $\frac{n}{4}$ qubits instead of $\frac{n}{2}-1$. This takes a further $\lceil\frac{n}{8}\rceil$ queries and the query complexity of this algorithm is $\lceil\frac{5n}{8}$ which we show formally for $n \equiv 0 \bmod 4$ as we have discussed before.

Theorem 17. *The function f_n^{id} can be evaluated by an exact quantum algorithm that makes $\lceil\frac{5n}{8}\rceil$ queries to the oracle and uses $\lfloor\frac{n}{4}\rfloor+1$ qubits as working memory.*

Proof.

For $1 \le i \le \lfloor\frac{n}{4}\rfloor$ apply $\mathsf{get}(2i-1, 2i, 2i+k-1, 2i+k)$, followed by $\mathsf{CNOT}_{\mathsf{w_{i+1}}}^{\mathsf{Q_n}}$ and $\mathsf{CNOT}_{\mathsf{Q_n}}^{\mathsf{w_{i+1}}}$.

2 After $\frac{n}{2}$ queries the algorithm is in the state

$$\frac{1}{\sqrt{2}}\left((-1)^{f_1(\mathbf{x})}|\mathbf{0}\rangle_0|0\rangle_1\bigotimes_{j=2}^{\frac{n}{4}+1}|x_{2j-1}\rangle_j + (-1)^{f_2(\mathbf{x})}|\mathbf{0}\rangle_0|1\rangle_1\bigotimes_{j=2}^{\frac{n}{4}+1}|x_{2j+k-1}\rangle_j\right)|-\rangle_{k+1}.$$

where $f_1(\mathbf{x}) = x_1x_2 \oplus \ldots \oplus x_{k-1}x_k$ $f_2(\mathbf{x}) = x_{k+1}x_{k+2} \oplus \ldots \oplus x_{n-1}x_n$ so that $f_n^{id}(\mathbf{x}) = f_1(\mathbf{x}) \oplus f_2(\mathbf{x})$.

3 At this stage apply $\mathsf{untangle}_{\mathsf{n}}^{\frac{\mathsf{n}}{4}}$ which further makes $\lceil\frac{n}{8}\rceil$ queries and the system is in $\frac{1}{\sqrt{2}}\left((-1)^{g(\tilde{\mathbf{x}})}|\mathbf{0}\rangle_0|0\rangle_1 + (-1)^{h(\tilde{\mathbf{x}})}|\mathbf{0}\rangle_0|1\rangle_1\right)\bigotimes_{i=1}^{\frac{n}{4}}|x_{r(i)}\rangle_{2i}|x_{r(k+i)}\rangle_{2i+1}$ and measuring w_1 in the computational basis gives us the output.

□

4.3.1 Beyond the Identity Permutation

We have shown that our algorithm can evaluate the MM Bent functions of type $f_n^{id}(\mathbf{x}) \oplus g(\hat{\mathbf{x}}')$ where x' is a subset of $\hat{\mathbf{x}}$ consisting of at most $\lceil\frac{n}{4}\rceil$ variables. However, the techniques we have used are do not restrict the permutation to be identity permutation. The algorithm works on dividing the variables of $\hat{\mathbf{x}}$ into two (close to) equal disjoint sets and then calculating the value of the corresponding points of $\tilde{\mathbf{x}}$, depending on the permutation. In case of the identity permutation, since the importance of the variable $x_{\frac{n}{2}+i} \in \tilde{\mathbf{x}}$ depended solely on the value of $x_i \in \hat{\mathbf{x}}$ we could

realize this procedure in a sequential manner. Therefore, as long we have a permutation such that it can be expressed as the concatenation of two permutations on $\frac{n}{4}$ variables each, or more precisely concatenation of permutations on $\lfloor \frac{n}{4} \rfloor$ and $\lceil \frac{n}{4} \rceil$ variables, we should be able to calculate the influencing variables in $\tilde{\mathbf{x}}$ corresponding to the values of the variables in $\hat{\mathbf{x}}$ at parallel, and thus be able to evaluate the function with the same query complexity of $\lceil \frac{5n}{8} \rceil$. We now concretize this relaxation in restraint and the corresponding modifications needed in the algorithm.

Theorem 18. *Let f be an MM Bent function f on n variables such that $f = \phi(\hat{\mathbf{x}}) \cdot \tilde{\mathbf{x}} \oplus g(\hat{\mathbf{x}}')$, with the following constraints:*

1 ϕ_1 and ϕ_2 are two permutations such that $\phi(\hat{\mathbf{x}}) \cdot \tilde{\mathbf{x}} = \phi_1(\hat{\mathbf{y}}) \cdot \tilde{\mathbf{y}} \oplus \phi_2(\hat{\mathbf{z}}) \cdot \tilde{\mathbf{z}}$

2 The sets of variables $\hat{\mathbf{y}}, \hat{\mathbf{z}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}}$ are all disjoint, $|\hat{\mathbf{y}}| = \tilde{\mathbf{y}} = \lfloor \frac{n}{4} \rfloor$, $\hat{\mathbf{z}} = \tilde{\mathbf{z}} = \lceil \frac{n}{4} \rceil$

3 $\hat{\mathbf{y}} \cup \hat{\mathbf{z}} = \hat{\mathbf{x}}$ and $\tilde{\mathbf{y}} \cup \tilde{\mathbf{z}} = \tilde{\mathbf{x}}$

4 $\hat{\mathbf{x}}' \subset \hat{\mathbf{x}}, |\hat{\mathbf{x}}' \cap \hat{\mathbf{y}}| \leq \lceil \frac{n}{8} \rceil, |\hat{\mathbf{x}}' \cap \hat{\mathbf{z}}| \leq \lceil \frac{n}{8} \rceil$

Then the function can be evaluated by an exact quantum query algorithm that makes $\lceil \frac{5n}{8} \rceil$ queries to the oracle and uses $\frac{n}{2} + 1$ qubits as working memory.

Proof. Without loss of generalization, let us assume $\hat{\mathbf{y}} = \left\{ x_1, \ldots x_{\lfloor \frac{n}{4} \rfloor} \right\}$, $\hat{\mathbf{z}} = \left\{ x_{\lfloor \frac{n}{4} \rfloor + 1}, \ldots, x_{\frac{n}{2}} \right\}$ and $\tilde{\mathbf{y}} = \left\{ x_{\frac{n}{2}+1}, \ldots x_{\frac{n}{2}+\lfloor \frac{n}{4} \rfloor} \right\}$, $\tilde{\mathbf{z}} = \left\{ x_{\frac{n}{2}+\lfloor \frac{n}{4} \rfloor + 1}, \ldots, x_n \right\}$.

Here we know that linear function in $\tilde{\mathbf{x}}$ to be evaluated for an input is dependent on $\hat{\mathbf{x}}$ so that the influencing variables in $\tilde{\mathbf{y}}$ are solely dependent on $\hat{\mathbf{y}}$ and the influencing variables in $\tilde{\mathbf{z}}$ are solely dependent on $\hat{\mathbf{z}}$. Thus we need to first obtain the values of $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ in the two product states of the superposition state, and the obtain the local phase of the corresponding linear function and obtain the outcome using untangling protocol. This happens as follows.

1 We initialize the algorithm in the state $|\mathbf{0}\rangle_0 \bigotimes_{i=1}^{\lceil \frac{n}{4} \rceil + 1} |0\rangle_i$. and apply Hadamard on w_1 to obtain $|\psi_0\rangle = \frac{1}{\sqrt{2}} \left(|\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{i=2}^{\lceil \frac{n}{4} \rceil + 1} |0\rangle_i + |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{i=2}^{\lceil \frac{n}{4} \rceil + 1} |0\rangle_i \right)$.

2 We apply $\mathsf{get}\left(i, \frac{n}{2} + i\right)$ followed by $\mathsf{CNOT}^{\mathsf{Q_n}}_{\mathsf{w_{i+1}}}$ and $\mathsf{CNOT}^{\mathsf{w_{i+1}}}_{\mathsf{Q_n}}$ for $1 \leq i \leq \lfloor \frac{n}{4} \rfloor$ and then controlled on $w_1 = |1\rangle_1$ also obtain the value of $x_{\frac{n}{2}}$ in case $\lceil \frac{n}{4} \rceil = \lfloor \frac{n}{4} \rfloor + 1$ (we shall anyhow assume $n \equiv 0 \bmod 4$ for simplicity).

3 After $\frac{n}{4}$ queries the system is in the state

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}\left(|\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{i=1}^{\frac{n}{4}} |x_i\rangle_{i+1} + |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{i=1}^{\frac{n}{4}} |x_{\frac{n}{4}+i}\rangle_{i+1}\right).$$

4 Controlled on $w_1 = |0\rangle$ obtain the phase of the variables in $\tilde{\mathbf{y}}$ according to the values of the variables in $\hat{\mathbf{y}}$ using controlled not operations and oracle queries Similarly controlled on $w_1 = |1\rangle$ obtain the phase of the variables in $\tilde{\mathbf{z}}$ according to the values of the variables in $\hat{\mathbf{z}}$. These steps take a total of $\frac{n}{4}$ queries and thus $\frac{n}{2}$ queries the system is in the state

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}\left((-1)^{\phi_1(\hat{\mathbf{y}})\cdot\tilde{\mathbf{y}}} |\mathbf{0}\rangle_0 |0\rangle_1 \bigotimes_{i=1}^{\frac{n}{4}} |x_i\rangle_{i+1} + (-1)^{\phi_2(\hat{\mathbf{z}})\cdot\tilde{\mathbf{z}}} |\mathbf{0}\rangle_0 |1\rangle_1 \bigotimes_{i=1}^{\frac{n}{4}} |x_{\frac{n}{4}+i}\rangle_{i+1}\right)$$

.

4 Apply the protocol $\mathsf{untangle}_{\mathsf{n}}^{\frac{\mathsf{n}}{4}}$ which takes a further $\lceil\frac{n}{8}\rceil$ queries to bring the system to

$$|\beta_f\rangle = |\mathbf{0}\rangle_0 |\phi_1(\hat{\mathbf{y}})\cdot\tilde{\mathbf{y}} \oplus \phi_2(\hat{\mathbf{z}})\cdot\tilde{\mathbf{z}}\rangle_1 \bigotimes_{i=1}^{\lceil\frac{n}{8}\rceil} (|x_{r(i)}\rangle_{i+1} |x_{\hat{r}(i)}\rangle_{i+2})$$

where $\{r(i)\}_{i=1}^{\lceil\frac{n}{8}\rceil} \subset \hat{\mathbf{y}}$ and $\{\hat{r}(i)\}_{i=1}^{\lceil\frac{n}{8}\rceil} \subset \hat{\mathbf{z}}$ are as per our choice, which can then be used to evaluate $g(\hat{\mathbf{x}}')$, which completes the protocol.

□

The case of odd n

So far, we have concentrated on the class of MM Bent functions, which are defined for all even n, and have obtained a large class of functions with deterministic query complexity of n which our exact quantum algorithm evaluates using $\lceil\frac{5n}{8}\rceil$ queries.

However this technique can be extended for all odd values of odd n as well. This can be done as follows.

1. Take any function on $f = \phi(\hat{x}).\tilde{x} \oplus g(x')$ on $n = 2k$ variables such that ϕ and g follow the constraints of Theorem 18.

2. Form the function $f' = f(x) \oplus x_{n+1}$

Since f has a polynomial degree of n, as shown in [4], this implies f' has a polynomial degree of $n+1$. This function can be evaluated in the exact quantum model by first evaluating f using $\lceil \frac{5n}{8} \rceil$ queries and using one more query to obtain the value of x_{n+1}. Thus this takes $\lceil \frac{5n}{8} \rceil + 1 \leq \lceil \frac{5(n+1)}{8} \rceil + 1$ queries. The number of functions that can be evaluated in this case is same as that for n.

4.3.2 The number of functions evaluated:

We finally calculate the number of functions covered via the definition of Theorem 17 for even n ($|\Gamma_n|$), and the number of functions for any odd n is the same as the number of functions for $n-1$. We essentially give a lower bound on the number of functions, as our calculation is based on a single partition of $\hat{x}$ and $\tilde{x}$ into these four sets, and any choice of x'.

There are $2^{\lfloor \frac{n}{4} \rfloor}$ inputs to the first permutation and $2^{\lceil \frac{n}{4} \rceil}$ inputs to the second permutation, and x' contains $\lceil \frac{n}{4} \rceil$ inputs. Therefore the total number of functions are $\left(2^{\lfloor \frac{n}{4} \rfloor}!\right)\left(2^{\lceil \frac{n}{4} \rceil}!\right)\left(2^{2^{\lceil \frac{n}{4} \rceil}}\right)$.

We now recall the definition of PNP-equivalence from [3].

Definition 8. *Two functions f and g are called PNP-equivalent if f can be obtained from g by permuting the name of the variables in g, replacing some variables x_i with $x_i \oplus 1$ in g and by finally complementing the new formed function with 1.*

If two functions are PNP equivalent then they have the same deterministic and exact quantum query algorithm and often an algorithm to evaluate one of them can be very easily modified to evaluate the other using the same number of queries.

Corresponding to a function on n variables, there can be at most $n!2^{n+1}$ functions that are PNP-equivalent to it. This is because there can be $n!$ permutation of variables and each variable x_i can be replaced with $x_i \oplus 1$, and finally each function $f(x)$ can be replaced with $f(x) \oplus 1$. Also, the PNP-equivalence relation is reflective, symmetric and transitive in nature.

Therefore if there is a set of cardinality S consisting of functions on n variables, then it consists of at least $\frac{\mathsf{S}}{n!2^{n+1}}$ functions that are not PNP-equivalent.

Therefore in this case the class Γ_n (exactly evaluated by our algorithm using $\lceil \frac{5n}{8} \rceil$ or $\lceil \frac{5n}{8} \rceil + 1$ queries) must consist of at least

$$\frac{\left(2^{\lfloor \frac{n}{4} \rfloor}!\right)\left(2^{\lceil \frac{n}{4} \rceil}!\right)\left(2^{2^{\lceil \frac{n}{4} \rceil}}\right)}{n!2^{n+1}} = \Omega\left(2^{\left(\lfloor \frac{n}{4} \rfloor 2^{(\lfloor \frac{n}{4} \rfloor)}\right)}\right)$$

functions, which is doubly exponential in $\lfloor \frac{n}{4} \rfloor$.

In conclusion, the fact that this algorithm cannot evaluate all MM Bent functions and thus all functions derived using the Bent concatenation method for odd values of n is a limitation compared to the parity decision method, which we note down in the following remark.

Remark 5. *The parity decision tree method in [4] evaluates all MM Bent functions on n variables using $\lceil \frac{3n}{4} \rceil$ queries where as the algorithm described in this requires $\lceil \frac{5n}{8} \rceil$ queries, but is able to evaluate only the MM Bent functions that meet the constraints described in Theorem 18.*

While the family of algorithms designed by us evaluates a class of functions super exponential in $\lfloor \frac{n}{4} \rfloor$, with a query complexity lower than any known parity decision tree technique, it lacks in two areas. The first is that we are unable to show that $QC_{algo}(f) = Q_E(f)$ for these functions. The second is that we are unable to show $QC_{algo}(f) < D^2_{\oplus}(f)$ for any of these functions. That is, we do not know if there exists a parity decision tree technique that can have the same query complexity as the family of algorithms we have presented. We have noted down in Chapter 3, Theorem 10 that $D_{\oplus}(f)$ is lower bounded by granularity. It is known that MM type Bent functions have a flat Fourier Spectra, with $\hat{f}(S) = \frac{1}{2^{\frac{n}{2}}} \ \forall \ S \subseteq [n]$. Therefore granularity of any MM type Bent function is $\frac{n}{2}$ which gives us a lower bound that we can show to be tight.

4.4 Future Directions

In this chapter we have made progresses in terms of applying parity decision tree and our novel exact quantum query algorithms to find optimal query algorithms for the MM type bent functions, a non symmetric class of functions with a size of super exponential in $\mathcal{O}(n)$.

First in Section 4.2 we obtained a $\lceil \frac{3n}{4} \rceil$-query parity decision tree technique for all MM type bent functions by analyzing its $\mathbb{F}_2$ polynomial (algebraic normal form).

Then in Section 4.3 we applied the $\mathbb{F}_2$ polynomial and untangling based techniques of Chapter 3. In doing so we have designed algorithms for a subclass of MM type Bent functions (a variable XOR-ed with MM Bent function when n is odd) consisting of at least $\Omega(2^{2^{\lfloor \frac{n}{4} \rfloor}})$ functions that are not PNP equivalent for any value of n. This family of algorithms have query complexity of $\lceil \frac{5n}{8} \rceil$ where as the lowest query complexity of any known parity decision tree technique is $\lceil \frac{3n}{4} \rceil$. While $Q_{algo}(f)$ is optimal for $f = x_1x_4 \oplus x_2x_5 \oplus x_3x_6$, we could neither show $QC_{algo}(f) = Q_E(f)$ or that

$QC_{algo}(f) < D^{(2)}_{\oplus}(f)$ for these classes of functions, which we note down here as open problems.

1. Does there exist any parity based method that can evaluate functions from this subclass using less than $\lceil \frac{3n}{4} \rceil$ queries?

2. What is the exact quantum query complexity of the functions in this class?

Bibliography

[1] P. Camion, C. Carlet, P. Charpin and N. Sendrier. 1991. On correlation-immune functions. Advances in Cryptology: Crypto 1991, Proceedings, LNCS, Vol. 576, Springer, Berlin,1991, pp. 86–100.

[2] J. F. Dillon. 1974. Elementary Hadamard Difference sets. Ph.D. Dissertation, Univ. of Maryland.

[3] A. Montanaro, R. Jozsa, and G. Mitchison. 2015. On Exact Quantum Query Complexity. Algorithmica 71, pp. 775–796.

[4] C. S. Mukherjee and S. Maitra. Classical-Quantum Separations in Certain Classes of Boolean Functions– Analysis using the Parity Decision Trees. Arxiv: `https://arxiv.org/abs/2004.12942` (2020).

[5] P. Sarkar and S. Maitra. 2000. Construction of Nonlinear Boolean Functions with Important Cryptographic Properties. 485-506. Advances in Cryptology - EUROCRYPT 2000. LNCS, vol 1807.

Ending Note

In this document we have studied both implementational and theoretical aspects of quantum computation. Part I concentrates on efficient implementation of an algorithm on a given architecture. Specifically, we study the state of the art deterministic Dicke state preparation circuits, and further optimize them using the concept of "partial unitary transformation". Our modifications not only reduce the gate count of the circuits, but also reduces architectural dependence in terms of CNOT connectivity, which is beneficial when implementing on superconducting based architectures, such as the IBM-Q cloud platform. To demonstrate the advantage of these changes, we implement our algorithm for $|D_2^4\rangle$ on the "ibmqx2" machine and show that our improvements indeed reduce error by a significant margin. We also discuss a simplified error model that helps us capture error due to different implementation on expectation.

Part II of this document is dedicated to the query complexity model. In this model, the central problem is, given the description of a function f on n variables evaluating the function for all possible inputs, with the restriction that the value of the variables can only be accessed via querying an oracle. We introduce the model in Chapter 1. Then we explore the separations between classical deterministic and exact quantum query complexity in for different classes of Boolean functions in Chapter 2, 3 and 4.

Chapter 2 is dedicated to query friendly functions, which are functions on any n variables with minimum deterministic complexity of n variable functions. We obtain several separability and non-separability results and show in some cases the parity decision tree is the optimal method for exact quantum query complexity, giving new instances of said case and also point out to open questions that should help us better understand the power of parity method.

In chapter 3 we go beyond parity decision and general parity decision tree techniques for certain

classes of non-symmetric Boolean functions. We explore how the ANF structure of Boolean functions can be used to design optimal exact quantum query algorithms that outperform any parity decision tree technique. In doing so we design a novel untangling technique that is not limited in its implementation to these particular functions.

Chapter 4 contains study of deterministic and exact quantum query complexity for the MM type bent functions, which is a large class with doubly exponential in n functions for any even n. For this class we obtain deterministic query complexity and give bounds on parity decision tree complexity and exact quantum query complexity, although our bounds are not tight. We also show that our untangling based protocols outperforms all known parity decision methods for a subclass of MM type bent functions of size $\Omega\left(2^{2^{\frac{n}{4}}}\right)$ functions. We conclude with the open questions we have in this domain.