UNIVERSITY OF LATVIA

ALINA VASIĻJEVA

COMPLEXITY OF QUANTUM ALGORITHMS AND COMMUNICATION PROTOCOLS

Doctoral Thesis

Area: Computer Science
Sub-Area: Mathematical Foundations of Computer Science

Scientific Advisor:
Dr. habil. math., Prof.
Rūsiņš Mārtiņš Freivalds

Riga 2011

This work has been supported by the European Social Fund
within the project «Support for Doctoral Studies at University of Latvia».

Abstract

Quantum computing is a method of computation based on the laws of quantum mechanics. This subfield of computer science aims to employ quantum mechanical effects for the efficient performance of computational tasks. The main research object of this thesis is the quantum query model.

The aim of the present research study is to discover efficient quantum query algorithms for certain problems and to develop general techniques for designing algorithms. The study has produced several outcomes regarding different kinds of quantum algorithms: exact, bounded-error, and nondeterministic.

In the first part of the thesis, exact and bounded-error quantum query algorithms for computing Boolean functions are presented. In the second part, a query model is applied for computing multivalued functions. The third part is devoted to nondeterministic query algorithms.

Anotācija

Kvantu skaitļošana ir datorzinātnes apakšnozare, kas balstās uz kvantu mehānikas likumiem, kuru iespējas un priekšrocības tiek izmantotas, lai efektīvāk risinātu skaitļošanas uzdevumus. Galvenais darbā pētāmais objekts ir kvantu vaicājošo algoritmu modelis.

Pētījuma galvenais mērķis ir efektīvo kvantu vaicājošo algoritmu atrašana konkrētām problēmām, kā arī vispārīgo algoritmu konstruēšanas metožu izstrādāšana. Ir iegūti rezultāti attiecībā uz dažādiem kvantu vaicājošo algoritmu tipiem, kā eksaktais algoritms, algoritms ar kļūdas varbūtību un nedeterminētais algoritms.

Promocijas darba pirmajā daļā ir apskatīti kvantu eksaktie algoritmi un kvantu algoritmi ar kļūdas varbūtību Būla funkciju rēķināšanai. Otrajā daļā vaicājošo algoritmu modelis ir pielietots daudzvērtīgu funkciju rēķināšanai. Trešā daļa ir veltīta nedeterminētiem vaicājošiem algoritmiem.

Preface

The present thesis summarizes the research performed by the author and reflected in the following publications.

Most significant publications:

1. A. Vasilieva, T. Mischenko-Slatenkova. Quantum Query Algorithms for Conjunctions. *Proc. of the 9th International Conference UC 2010,* Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol. 6079/2010, ISBN: 978-3-642-13522-4, pp. 140-151, 2010
2. A. Vasilieva, R. Freivalds. Nondeterministic Query Algorithms. *Journal of Universal Computer Science (J. UCS)* 17(6): 859-873, 2011
3. A. Vasilieva. Uniformly Distributed Quantum Query Algorithms for Multifunctions. *Proc. of the Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)* , pp. 86-93, 2011
4. A. Vasilieva. Quantum Query Algorithms for Relations. *Proc. of the MFCS & CSL 2010 Satellite Workshop Randomized and quantum computation*, ISBN 978-80-87342-08-4, pp. 78-89, 2010
5. A. Vasilieva. Exact Quantum Query Algorithm for Error Detection Code Verification. *Proc. of the Fifth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS),* ISBN 978-80-87342-04-6, pp. 200-207, 2009
6. A. Vasilieva. Quantum Query Algorithms for AND and OR Boolean Functions. *Logic and Theory of Algorithms, Fourth Conference on Computability in Europe*, *CiE 2008, Proceedings*, pp. 453-462, 2008
7. A. Dubrovska. Properties and Application of Nondeterministic Quantum Query Algorithms. *SOFSEM 2007: Theory and Practice of Computer Science*; Proc. Volume II; MatFyz Press; ISBN 80-903298-9-6; pp. 37-49, 2007

Other publications:

8. A. Dubrovska. Quantum Query Algorithms for Certain Functions and General Algorithm Construction Techniques. *Quantum Information and Computation V*, Proc. of SPIE Vol. 6573 (SPIE, Bellingham, WA) Article 65730F, 2007
9. A. Dubrovska, T. Mischenko-Slatenkova. Computing Boolean Functions: Exact Quantum Query Algorithms and Low Degree Polynomials. *Proc. of SOFSEM 2006, Student Research Forum*; MatFyz Press; ISBN 80-903298-4-5; pp. 91-100, 2006
10. A. Dubrovska. Complexity of Dual Nondeterministic Quantum Query Algorithms. *Proc. of ERATO Conference on Quantum Information Science,* pp. 175-176, 2005
11. A.Vasilieva, T. Mischenko-Slatenkova. Computing Relations in the Quantum Query Model. *Scientific Papers, University of Latvia, Volume 770, Computer Science and Information Technologies*, ISBN 978-9984-45-377-4, pp. 68-89, 2011
12. A.Vasilieva. Quantum Algorithms for Computing the Boolean Function AND and Verifying Repetition Code. *Scientific Papers, University of Latvia, Volume 756, Computer Science and Information Technologies*, ISBN 978-9984-45-200-5, 2010, pp. 227-247, 2010

13. A. Vasiljeva. Quantum Query Algorithm Constructions for Computing AND, OR and MAJORITY Boolean Functions. *Scientific Papers, University of Latvia, Volume 733, Computer Science and Information Technologies*, ISBN 987-9984-825-47-0, pp. 215-238, 2008

All papers except 2, 4, 11, 12, and 13 have been presented by the author at the conferences by reports or posters.

Additionally, the author has been participated with posters in the following conferences:

1. A.Vasilieva, T. Mischenko-Slatenkova. Quantum Query Model: Application to Computing Mathematical Relations. *The Sixth Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC)*, Madrid, Spain, May 24-26, 2011
2. A. Vasilieva, R. Agadzanyan. Quantum Query Algorithms for AND, OR and MAJORITY Boolean Functions. *The Twelfth Workshop on Quantum Information Processing (QIP)*, Santa Fe, New Mexico, USA, January 12-16, 2009
3. A. Dubrovska. Properties and Application of Nondeterministic Quantum Query Algorithms. *The Ninth Workshop on Quantum Information Processing (QIP)*, Paris, France, January 16-20, 2006
4. A. Dubrovska, T. Mischenko-Slatenkova. Computing Boolean Functions: Exact Quantum Query Algorithms and Low Degree Polynomials. *The Ninth Workshop on Quantum Information Processing (QIP)*, Paris, France, January 16-20, 2006

Acknowledgements

I wish to express sincere thanks to respected professor Rūsiņš Freivalds, my supervisor, first of all, for arousing my interest about quantum computing; through deep knowledge of the subject and outstanding skills to vividly clarify even the most complicated things inherent to him, he opened the doors into the world of science in front of me. I am thankful for permanent support, advice, interesting ideas and thoughtful guidance he provided over the whole period while I worked on the present thesis.

Special thanks to professor Andris Ambainis and associated professor Juris Smotrovs for reviewing parts of the thesis and helpful comments.

Finally, my appreciation to my family, especially my husband Alexander, for many years of support, inspiration and patience.

This work has been supported by the European Social Fund within the project «Support for Doctoral Studies at University of Latvia».

Contents

1 Introduction 9

2 Theoretical Background 12

2.1 Classical Decision Trees 12

2.2 Quantum Computing 14

2.3 Quantum Query Model 16

2.4 Quantum Query Algorithm for *XOR* Function 18

3 Quantum Query Algorithms for Boolean Functions 20

3.1 Exact Quantum Query Algorithms 20

3.1.1 Basic Exact Quantum Query Algorithms 22

3.1.2 Exact Quantum Query Algorithm Classification 24

3.1.3 Algorithm Transformation Methods 27

3.1.4 Exact Quantum Query Algorithm for Verifying Repetition Code 33

3.1.5 Algorithm Concatenation Methods 37

3.1.6 Conclusion and Open Problems 42

3.2 Bounded-Error Quantum Query Algorithms 43

3.2.1 Quantum Query Algorithms for Conjunctions 43

3.2.2 Conclusion and Open Problems 55

4 Quantum Query Algorithms for Multivalued Functions 56

4.1 Multivalued Functions 57

4.2 Computing Multifunctions in a Query Model 58

4.2.1 A Note on Computing Multifunctions in a Definite Manner 59

4.3 First Example of Computing a Multifunction 59

4.3.1 Definite Query Complexity of the Multifunction M_1 60

4.3.2 Uniformly Distributed Query Complexity of the Multifunction M_1 61

4.3.3 Generalizations of the Multifunction M_1 63

4.4 Second Example of Computing a Multifunction 67

4.4.1 Uniformly Distributed Query Complexity of the Multifunction M_2 67

4.4.2 Generalizations of the Multifunction M_2 69

4.5 Third Example of Computing a Multifunction 72
4.5.1 Uniformly Distributed Query Complexity of the Multifunction M_3........ 73
4.5.2 Generalizations of the Multifunction M_3 75
4.6 Conclusion and Open Problems 77
5 Nondeterministic Query Algorithms 78
5.1 Traditional Nondeterministic Quantum Query Model 78
5.1.1 Dual Nondeterministic Quantum Query Algorithms 78
5.1.2 Properties of NQQA 79
5.1.3 Application of NQQA Properties 84
5.1.4 Conclusion and Open Problems 88
5.2 Alternative Nondeterministic Query Model 89
5.2.1 Definition of the Alternative Nondeterministic Query Model 89
5.2.2 Computing the Fano Plane Function 90
5.2.3 Conclusion and Open Problems 97
6 Conclusion 98
Bibliography 100
Appendix 1: Wolfram *Mathematica* program 103
Appendix 2: Computation of the quantum query algorithm for $EQUALITY_3$ 106
Appendix 3: Computation of the quantum query algorithm for $PAIR\_EQUALITY_4$..... 107
Appendix 4: Computation of the quantum query algorithm QM1 109

1 Introduction

Quantum computing is a subfield of computer science based on the laws of quantum mechanics. It applies the quantum mechanical effects for more efficient solution of computational problems than in a classical way. This area of science unites disciplines such as physics, mathematics and computer science. In the scope of computer science and mathematics the theoretical aspects, the potential and the limitations of quantum computers are studied. At the same time, physicists are working on developing practical implementations of quantum computing devices. This branch of science is very topical because it is a proven fact that quantum computing can solve certain problems faster than classical computing. Quantum computing became very popular in the mid-1990s after Peter Shor presented the polynomial time integer factoring algorithm [1]. Another famous example is Grover's search algorithm [2]. Although quantum computers are not yet available to everyone, many scientists all over the world are working to make them universally available in the future. The physical implementation is very complex; however, several quantum computer prototypes have been developed and are used to solve computational problems, e.g. [3], [4], [5]. Moreover, theoretical results in the field of quantum information processing are already successfully implemented in such areas as quantum cryptography [6], [7] and quantum teleportation [8], [9]. Several companies, for instance, *IDQuantique* (Geneva) [10], are already marketing commercial quantum key distribution systems. The *D-Wave* company even claims to offer commercial quantum computing systems [11]. Such rapid progress gives hope that sooner or later quantum computers will enter our everyday life in the same way as conventional computers did.

Algorithm complexity theory is a sub-field of computer science investigating the complexity of computational problems. One of the main tasks in complexity theory is designing efficient algorithms. In this thesis, theoretical aspects of quantum complexity theory and the present results of designing quantum algorithms are considered. The main goal of the research is to develop new, fast and efficient quantum algorithms for solving specific computational problems, as well as to improve general construction techniques for algorithms.

The main object of the research is the query model [12], a popular model of computation. In this model, the definition of the function $f(X)$ is known, but input $X = (x_1, x_2, ..., x_N)$ is hidden in a black box. Input values can be accessed only by querying the black box about x_i values. In the process of computation, the query algorithm asks questions about variable values, receives answers from the black box, performs the computation, and finally produces the function value output. The goal is to develop a query algorithm that would compute the value of a certain function correctly for an arbitrary input. The complexity of a query algorithm is measured by the number of questions it asks based on worst-case input. This computational model is widely applicable in software engineering. For instance, a database can be considered a black box. To speed up application performance, the number of database queries must be reduced. Another application is evaluation of conditional Boolean statements in computer programs.

The quantum query model differs from the quantum circuit model [13], and the algorithm construction techniques for this model are less developed. Although there is a large amount of lower and upper bound estimations of quantum query algorithm complexity [14], [15], [16], [17], [18], [19] examples of non-trivial and original quantum query algorithms are relatively few. Moreover, there is no special technique described to build a quantum algorithm for an arbitrary function with complexity defined in advance. Some results describing the development of such techniques are published in [20]. The goal of this research is to find new approaches for quantum query algorithm design.

Various types of query algorithms already exist and are actively studied. In the classical query model, there is a distinction among deterministic, randomized and nondeterministic query algorithms. The quantum query model has corresponding analogues, namely, exact, bounded-error and nondeterministic query algorithms. The goal of the present research is to study and analyze relations between different complexity measures, compare classical and quantum complexity of certain problems, and produce examples with a large separation between classical and quantum complexity.

The thesis is structured as follows.

In Chapter 2, the author provides the theoretical background and describes classical query algorithms, basics of quantum computing and the quantum query model.

In Chapter 3, the author presents the results of designing quantum query algorithms for computing Boolean functions. Two different types of quantum query algorithms are examined: the exact, which outputs the correct result with certainty, and the bounded-error, which outputs the correct result with some probability $p > 2/3$.

Summary of results:

- Two basic exact quantum query algorithms are presented. The first algorithm computes the three-variable Boolean function with two queries, while classically three queries are required. The second algorithm computes the four-variable Boolean function with two queries, while classically four queries are required.
- Exact quantum query algorithm transformation methods are introduced. These methods are helpful for enlarging the set of efficient exact algorithms.
- Exact quantum query algorithm concatenation methods are formulated. The methods can be used for generating examples of N versus $2N$ gaps between quantum and classical query complexity of a function.
- An exact quantum query algorithm for verification of repetition codes is developed. The algorithm complexity is N while classically $2N$ queries are required.
- A method for constructing a bounded-error quantum query algorithm for conjunction $f = f_1 \wedge f_2$ using exact quantum query algorithms for sub-functions f_1 and f_2 is developed. The correct probability of a correct answer is $p = 4/5$ and the complexity is equal to the largest complexity of sub-algorithms: $\max(Q_E(f_1), Q_E(f_2))$.

In Chapter 4, the author examines a more general type of a computational problem than Boolean functions and applies a query model to the computation of multivalued functions. In author's opinion, this section contains the most interesting and important results. Three different approaches for computing multivalued functions in a query model are proposed and examples are demonstrated in which the quantum query complexity is lower than in the classical case.

The most significant complexity gaps that have been obtained are the following:

- $C_{UD}(M_1^{GEN2}) = 2N$ versus $Q_{UD}(M_1^{GEN2}) \leq N$
- $C_{UD}(M_2^{GEN1}) \geq 3N$ versus $Q_{UD}(M_2^{GEN1}) \leq N$
- $C_{RD}(M_2^{GEN2}) \geq \frac{N}{2} + 1$ versus $Q_{RD}(M_2^{GEN2}) = 1$
- $C_{UD}(M_3^{GEN2}) = 6N$ versus $Q_{UD}(M_3^{GEN2}) \leq 2N$

In Chapter 5, the author discusses a nondeterministic query model. In Section 5.1, the results of designing algorithms in a traditional nondeterministic query model are presented. In Section 5.2, the author proposes a new alternative model for nondeterministic computation. The elaborated model is demonstrated through an example of computing a specific Boolean function, for which the gap between deterministic and nondeterministic query complexity is demonstrated to be 7^N versus $O(3^N)$.

2 Theoretical Background

This section contains definitions and provides theoretical background on the subject of the thesis. First, the author describes the classical decision trees and shows the way of computation of a simple Boolean function according to this model. Next, an overview is provided on the basics of quantum computing. Finally, the author describes the quantum query model in detail.

2.1 Classical Decision Trees

The classical version of the query model is known as *decision trees.* The black box contains the input $X = (x_1, x_2, ..., x_N)$ and can be accessed by querying x_i values. The algorithm must allow determination of the correct value of a function for an arbitrary input. The complexity of the algorithm is measured by the number of queries on the worst-case input. For more details, see the survey on decision tree complexity by Buhrman and de Wolf [12].

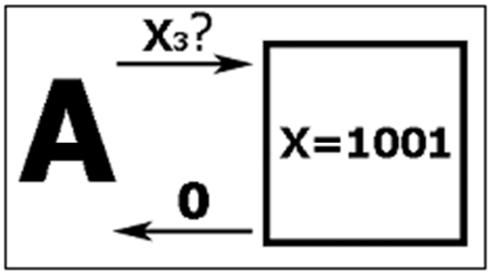


Fig. 2.1 Black-box mechanism of the query model

There exist different types of query algorithms, each exhibiting a specific behavior and defining different conditions for an algorithm to produce the correct result. In the present study, the following algorithm types are examined:

- deterministic;
- probabilistic;
- nondeterministic.

Deterministic decision tree is a tree with internal nodes labeled with variables x_i, arrows labeled with possible variable values and leafs labeled with function values. The deterministic decision tree always follows the same computational path for each input and produces correct result with probability $p = 1$.

Definition 2.1 [12]**.** *The* ***deterministic complexity*** *of a function f, denoted by* ***D(f)****, is the maximum number of questions that must be asked on any input by a deterministic algorithm for f.*

Definition 2.2 [12]**.** *The* ***sensitivity*** $s_x(f)$ ***of f on input*** *$(x_1,x_2,...,x_N)$ is the number of variables x_i with the following property: $f(x_1,...,x_i,...,x_N) \neq f(x_1,...,1-x_i,...,x_N)$. The* ***sensitivity of f*** *is* $s(f) = \max_x s_x(f)$.

It has been proven that $D(f) \geq s(f)$ [12].

Fig. 2.2 demonstrates a classical deterministic decision tree, computing Boolean function $MAJORITY_3(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$. In this figure, circles represent queries while rectangles represent the output values. It is evident that if values of first two queried variables are different the third query is required: $D(MAJORITY_3(x_1, x_2, x_3)) = 3$.

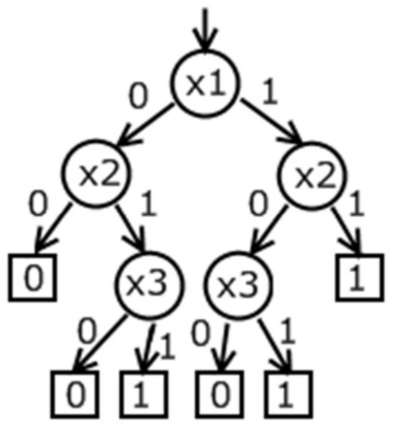


Fig. 2.2 Classical deterministic decision tree for computing $MAJORITY_3(x_1, x_2, x_3)$

As in many other models of computation, the power of randomization can be added to decision trees [12]. A *probabilistic decision tree* may contain internal nodes with a probabilistic branching, i.e., multiple arrows exiting from the above node, each one labeled with a probability for algorithm to follow. The total sum of all probabilities assigned to arrows in a probabilistic branching is supposed not to exceed 1. The result of a probabilistic decision tree is not determined with certainty any more by the input *X*. Instead, there is a probability distribution over the set of leaves. The total probability to obtain a result $b \in \{0,1\}$ after application of an algorithm on certain input *X* equals the sum of probabilities for each leaf labeled with *b* to be reached. The total probability of an algorithm to produce the correct result is the probability on the worst-case input.

Fig. 2.3 demonstrates a classical probabilistic decision tree, which computes Boolean function $AND_2(x_1, x_2) = x_1 \wedge x_2$.

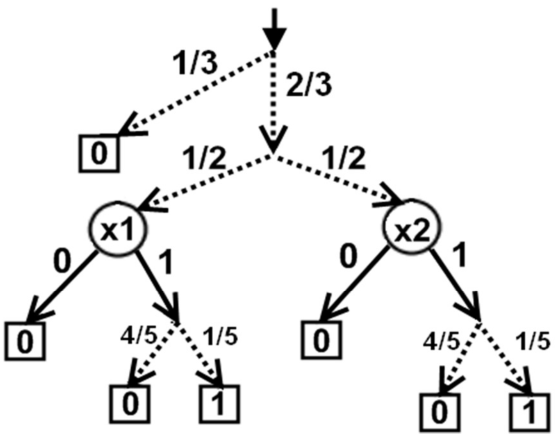


Fig. 2.3 Classical probabilistic decision tree for computing $AND_2(x_1, x_2)$

The *nondeterministic decision tree* differs from the deterministic one by an additional possibility that there can be more than one arrow labeled with the same value exiting each tree vertex. The interpretation of the nondeterministic computation is rather different from previously considered algorithm types.

Definition 2.3 [21] *The* ***nondeterministic decision tree*** *computes Boolean function f(X), if for an arbitrary input X it is true that:*

- *if f(X)=1, then a path exists from the root to the leaf with result 1;*
- *if f(X)=0, then a path exists from the root to the leaf with result 0;*
- *there is no path from the tree root to the leaf with incorrect function value.*

Fig. 2.4 demonstrates a classical nondeterministic decision tree, which computes Boolean function $MAJORITY_3(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$ with two queries.

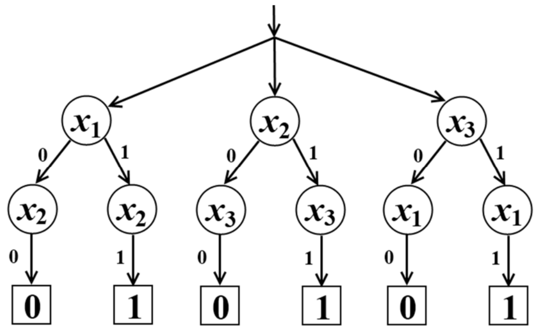


Fig. 2.4 Classical nondeterministic decision tree for computing $MAJORITY_3(x_1, x_2, x_3)$

2.2 Quantum Computing

The theory of quantum mechanics was developed during the first half of the 20th century by Niels Bohr, Werner Heisenberg, Max Planck and other distinguished scientists. The foundations of the quantum computing were established in 1982, when American physicist Richard Feynman [22] stated that attempts to simulate quantum mechanical systems on classical computers may lead to fundamental difficulties and complications, and suggested that these difficulties could be solved by building computers based on the principles of quantum mechanics. Later, in 1985, David Deutsch proposed a concrete computational model – the quantum Turing machine [23]. The theory of quantum computing became firmly established as a new and mature field of science only in the mid-1990s. For a long time it was unclear if it is possible to use the hypothetical power of quantum computer for solving practical problems more efficiently. Finally, in 1994, American mathematician Peter Shor presented his famous quantum algorithm for integer factorization [1]. The algorithm runs in polynomial time and its complexity is $O((\log n)^3)$. This algorithm demonstrates an exponential gap between classical and quantum algorithm complexity. The next achievement in 1995 was Lov Grover's algorithm for searching an unsorted database [2]. Algorithm searches an unsorted database with N entries in $O\left(\sqrt{N}\right)$ time, while classical complexity is $O(N)$. Since then, a number of quantum algorithms have been developed. It is worth to mention also algorithms based on quantum walks [24].

Shor's and Grover's algorithms are very fast; however, both algorithms are not exact and allow a certain error probability. The best known exact quantum query algorithm, always producing a correct result with probability $p = 1$, is an algorithm for *XOR* function [23], [25]. This algorithm uses $N/2$ queries to compute *XOR* of N bits, while classically N queries are required. It is a long-standing open question whether it is possible or not to build an exact quantum query algorithm for the total function, which would be more than twice faster than the best possible classical analogue.

In the remainder of this section, the author briefly outlines the basic notions of quantum computing. For more details, see textbooks by Gruska [26], Nielsen and Chuang [13] as well as Kaye et al. [27].

The fundamental concept of classical computing is a bit – the basic unit of information. The memory of a classical computer is made up of bits, where each bit represents either 0 or 1. The quantum analogue of the classical bit is a *qubit* – a unit of quantum information. Qubit also has two physical computational basis states. In Dirac or *bra-ket* notation [13], quantum basis states are denoted by $|0\rangle$ and $|1\rangle$. The main difference of qubit is that it can be in a *superposition* of the basis states. Formally, a pure qubit state is a linear combination of the basis states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where α and β are complex numbers, which are called *probability amplitudes* of the basis states. So, the arbitrary quantum state can be represented in a two dimentional complex space $\mathbb{C}^2$ as a unit vector $\left(|\alpha|^2 + |\beta|^2 = 1\right)$. When the qubit is measured in a standard basis, the probability of observing $|0\rangle$ is $|\alpha|^2$, and the probability of observing $|1\rangle$ is $|\beta|^2$.

More generally, an n-dimensional quantum pure state is a unit vector in an n-dimensional Hilbert space. Let $|0\rangle, |1\rangle, ..., |n-1\rangle$ be an orthonormal basis for $\mathbb{C}^n$. Then, any state can be expressed as $|\psi\rangle = \sum_{i=0}^{n-1} \alpha_i |i\rangle$ for some $\alpha_i \in \mathbb{C}$. The norm of $|\psi\rangle$ is 1: $\sum_{i=0}^{n-1} |\alpha_i|^2 = 1$. States $|0\rangle, ..., |n-1\rangle$ are called *basis states.* Any state of the form $\sum_{i=0}^{n-1} \alpha_i |i\rangle$ is called a *superposition* of $|0\rangle, ..., |n-1\rangle$. The coefficient α_i is called an *amplitude* of $|i\rangle$.

The state of a quantum system can be changed by applying the *unitary transformation.* The unitary transformation U is a linear transformation on $\mathbb{C}^n$ that maps vector of unit norm to another or the same vector of unit norm. Formally, unitary transformation is represented by a unitary matrix. A matrix U is *a unitary matrix* if and only if $U^* = U^{-1}$, where U^* is the conjugate transpose of U, and U^{-1} is the matrix inverse. The *transpose* of a matrix A is denoted by $A_{ij}^T = A_{ji}$. The *tensor product* of two matrices is denoted by $A \otimes B$.

An important trait of the quantum system is that it is not possible to inspect the superposition, i.e. to figure out the values α_i. This trait is disappointing in a sense, because on the one hand, the quantum system contains a large amount of information, but on the other hand, we are not able to observe and use this information in full. Limited information can be extracted from the quantum system by performing a measurement. There are various types of the quantum measurement; the quantum query model uses the simplest one – the full measurement in the computation basis. Performing this measurement on a state $|\psi\rangle = \alpha_0|0\rangle + ... + \alpha_{n-1}|n-1\rangle$ produces the outcome $|i\rangle$ with probability $|\alpha_i|^2$. The measurement changes the state of the system to $|i\rangle$ and destroys the original state $|\psi\rangle$. It means that repeated measurement makes completely no sense, because state $|i\rangle$ will be observed again with probability $p = 1$.

2.3 Quantum Query Model

The quantum query model is also known as the quantum black box model. This model is the quantum counterpart of decision trees and is intended for computing Boolean functions. For a detailed description, see surveys [12] and [16], textbooks by Kaye et al. [27], and de Wolf [14].

A quantum computation with T queries is a sequence of unitary transformations:

$$U_0\ ,\ Q_0\ ,\ U_1\ ,\ Q_1\ ,\ ...\ ,\ U_{T-1}\ ,\ Q_{T-1}\ ,\ U_T\ .$$

U_i's can be arbitrary unitary transformations not depending on input bits. Q_i's are unitary query transformations. Computation starts in the initial state $|\vec{0}\rangle$. Then transformations U_0, Q_0,..., Q_{T-1}, U_T are applied and the final state measured.

Bra and *ket* notations [13] are used to describe state vectors and algorithm structure:

Quantum algorithm computation process in *bra* notation: $\langle final| = \langle\vec{0}|U_0 Q_0 ... Q_{T-1} U_T$.

Quantum algorithm computation process in *ket* notation: $|final\rangle = U_T^* Q_{T-1}^* ... Q_0^* U_0^* |\vec{0}\rangle$.

The most important part of the model is the query. Every query has to correspond to a unitary transformation. There are many alternative ways to define a quantum query.

Most traditional formalization of a query is the following [12]: query to an input $X = (x_1, ..., x_N) \in \{0,1\}^N$ is a unitary transformation O that maps $|i, b, z\rangle$ to $|i, b \oplus x_i, z\rangle$. $|i, b, z\rangle$ is some m-qubit basis state, where i takes $\lceil \log N \rceil$ bits, b is one bit, z denotes the $(m - \lceil \log N \rceil - 1)$-bit "workspace" of the quantum computer, which is not affected by the query, and $\oplus$ denotes *XOR* operation.

In algorithms created by the present research study, a different definition of a query transformation is used: if the input is a state $|\psi\rangle = \sum_i \alpha_i |i\rangle$, then the output is:

$$|\gamma_i\rangle = \sum_{i=0}^{n-1} (-1)^{\varphi_i} \alpha_i |i\rangle, \text{ where } \varphi_i \in \{x_1, ..., x_N, 0, 1\}.$$

In other words, for each query for each basis state $|i\rangle$ a variable assignment φ_i may be arbitrarily chosen. It is also allowed to skip the variable assignment for any particular basis state, i.e. to set $\varphi_i = 0$ for $|i\rangle$; or inverse amplitude value sign by setting $\varphi_i = 1$ for a particular $|i\rangle$. Depending on the value of the assigned variable, the sign of the amplitude of the quantum basis state either changes to the opposite or remains unchanged.

The query of the second type can be simulated by the query of the first type [14], [25].

Formally, any transformation has to be defined by a unitary matrix. The following is a matrix representation of a quantum black box query.

$$Q = \begin{pmatrix} (-1)^{\varphi_1} & 0 & \dots & 0 \\ 0 & (-1)^{\varphi_2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & (-1)^{\varphi_m} \end{pmatrix}$$

After all query transformations Q_i are applied (alternating with fixed intermediate unitary transformations U_i), the last remaining action is to extract the result value from the final quantum state. It is achieved by measuring this state and interpreting the basis quantum state observed after that. In fact, each quantum basis state corresponds to the algorithm's output. A value of a function is assigned to each basis state. The probability of obtaining the result $b \in \{0,1\}$ after applying an algorithm on input *X* equals the sum of squared moduli of all amplitudes, which correspond to outputs with value *b*.

Quantum query algorithms can be conveniently represented in diagrams, and this approach is used throughout the thesis. Fig. 2.5 demonstrates a graphical representation of an algorithm in a general form.

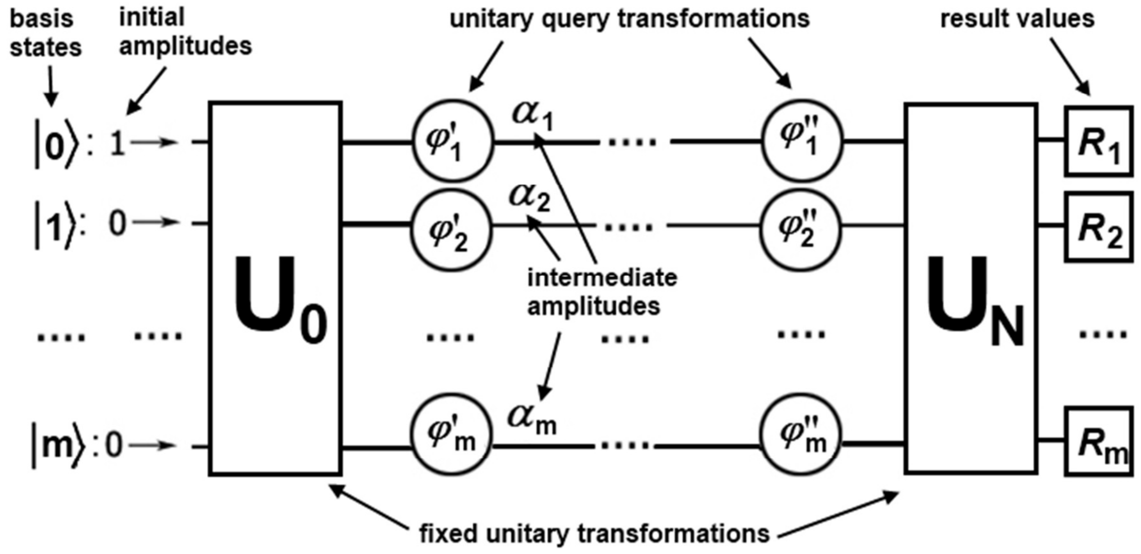


Fig. 2.5 Graphical representation of a quantum query algorithm

The most important characteristics of the graphical representation applied, is that *each horizontal line corresponds to the amplitude of the quantum basis state*, *not to a qubit*. So, if the quantum system consists of K qubits, there are 2^K horizontal lines on the diagram. The large rectangles represent the fixed unitary transformations. The vertical layers of the circles specify assignment of variables to basis states during a query. Finally, small squares at the end of each horizontal line define assigned function value for each basis state.

Like in a classical query model, also here various types of quantum query algorithms exist. In the scope of this thesis, the main interest lies in exact, bounded-error and nondeterministic algorithm types.

Definition 2.4 [12]. *A quantum query algorithm* ***computes f exactly*** *if the output equals f(X) with a probability* $p = 1$*, for all* $X \in \{0,1\}^N$. *Complexity is equal to the number of queries and is denoted by* $Q_E(f)$.

Definition 2.5 [12]. *A quantum query algorithm* ***computes f with bounded-error*** *if the output equals f(X) with probability* $p > 2/3$*, for all* $X \in \{0,1\}^n$. *Complexity is equal to the number of queries and is denoted by* $Q_p(f)$.

The notion of nondeterministic query algorithm is defined and discussed in detail in Section 5.1.

2.4 Quantum Query Algorithm for *XOR* Function

In this section, the author presents the simplest and at the same time the best-known exact quantum query algorithm to demonstrate the quantum query model in action[1]. This is an algorithm for computing the following Boolean function:

$$XOR_N(x_1, x_2, ..., x_N) = x_1 \oplus x_2 \oplus ... \oplus x_N = x_1 + x_2 + ... + x_N \pmod 2.$$

The original quantum algorithm has been developed by Deutsch in 1985 [23], and later improved [25]. To compute the value of this function, the quantum query algorithm uses two times less queries than the best possible classical algorithm.

Let us first consider the simplest case of two variables. The algorithm is presented in Fig. 2.6. H_2 is 2×2 Hadamard transform: $H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

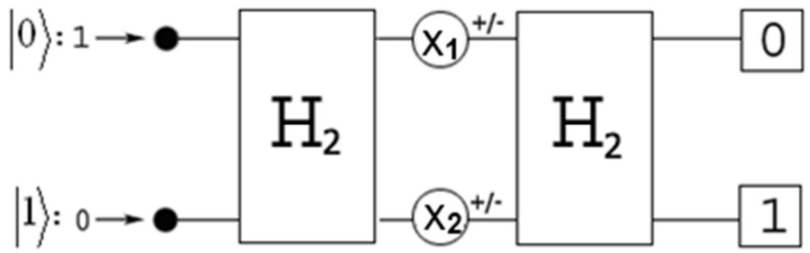


Fig. 2.6 Exact quantum query algorithm for computing $XOR_2(x_1,x_2)$ with one query

[1] Exact quantum algorithm with complexity $Q_E(f) < D(f)/2$ is not yet discovered for a total Boolean function. For partial Boolean functions (promise problems) this limitation can be exceeded. An excellent example is the Deutsch-Jozsa algorithm [28], [25].

The computation starts with the initial state $\left|\vec{0}\right\rangle = (1,0)^T$. Then, this vector is sequentially multiplied by the following matrices:

$$H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad Q = \begin{pmatrix} (-1)^{x_1} & 0 \\ 0 & (-1)^{x_2} \end{pmatrix} \qquad H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Finally, the measurement is performed on the final state.

Algorithm computation process for each input vector is presented in Table 2.1.

Table 2.1 Quantum query algorithm computation process for XOR_2

X	State after $H_2\left\|\vec{0}\right\rangle$	State after $QH_2\left\|\vec{0}\right\rangle$	State after $H_2QH_2\left\|\vec{0}\right\rangle$	$XOR_2(X)$
00	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)^T$	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)^T$	$(1,0)^T$	**0**
01	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)^T$	$\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)^T$	$(0,1)^T$	**1**
10	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)^T$	$\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)^T$	$(0,-1)^T$	**1**
11	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)^T$	$\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)^T$	$(-1,0)^T$	**0**

In the case of N variables, more sequential queries are carried out. A pair of variables is queried in each query, so the total number of queries is $N/2$. Algorithm is presented in Fig. 2.7. After all queries, the sign of the first amplitude is "+" only if among the first half of variables there is an even number of "1". The same is true for the second amplitude and the second half of variables. Amplitude signs are equal if the total number of "1" is even. Only in this case after the second Hadamard transform and the measurement quantum system collapses to "0".

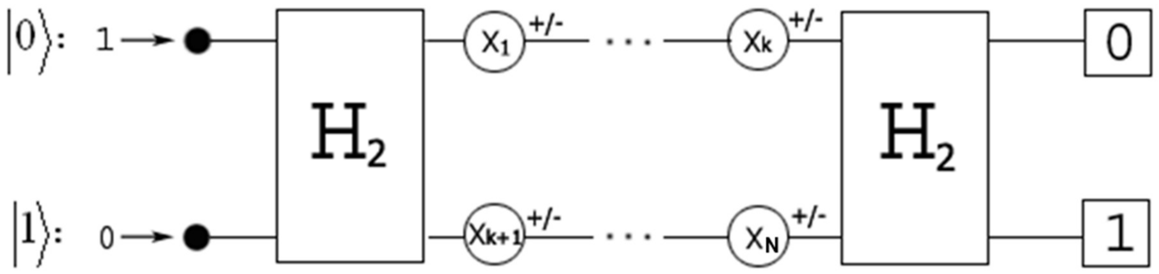


Fig. 2.7 Exact quantum algorithm for computing $XOR_N(X)$ with $N/2$ queries ($N = 2k$)

3 Quantum Query Algorithms for Boolean Functions

In this chapter, the author discusses the design and complexity of the quantum query algorithms for computing total functions.

The quantum query algorithm construction is a complicated problem. Although there is a large number of lower and upper bound estimations of quantum algorithm complexity available [14], [16], [15], [17] examples of non-trivial and original quantum query algorithms are relatively rare. Moreover, there are no special techniques for building a quantum algorithm for an arbitrary function with pre-defined complexity.

It is a rather complicated task to construct an efficient query algorithm for an arbitrary function. Difficulties arise even on attempts to develop a quantum algorithm saving just one query comparing to the classical one, for example, for the following finite functions:

$$F_4(x_1,x_2,x_3,x_4) = \neg\left(x_1 \oplus x_2\right) \wedge \neg\left(x_3 \oplus x_4\right),$$

$$F_6(X) = (\neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)) \wedge (\neg(x_4 \oplus x_5) \wedge \neg(x_5 \oplus x_6)),$$

or

$$F_{10}(X) = (f_1 \wedge f_2 \wedge f_3) \vee (f_1 \wedge f_2 \wedge f_4) \vee (f_1 \wedge f_3 \wedge f_4) \vee (f_2 \wedge f_3 \wedge f_4) \text{ where}$$

$$f_1 = (x_1 \oplus x_2) \vee (x_3 \oplus x_4);\ f_2 = x_5 \oplus x_6;\ f_3 = \neg(x_7 \oplus x_8) \wedge \neg(x_8 \oplus x_9);\ f_4 = \neg x_{10}.$$

Boolean functions are widely applied in the real life processes therefore the capacity of building a quantum algorithm for an arbitrary function appears to be extremely important. While working on common techniques how to achieve the above, the author was trying to collect examples of efficient quantum algorithms for building a base for powerful computation using the advantages of the quantum computer.

In this section, the results of designing different types of quantum query algorithms are presented. In Section 3.1, the author examines the exact quantum query algorithms. Section 3.2 is devoted to bounded-error quantum query algorithms.

3.1 Exact Quantum Query Algorithms

This section is based on the papers

- A. Dubrovska, T. Mischenko-Slatenkova. Computing Boolean Functions: Exact Quantum Query Algorithms and Low Degree Polynomials. *Proc. of SOFSEM 2006, Student Research Forum*; MatFyz Press; ISBN 80-903298-4-5; pp. 91-100, 2006
- A. Dubrovska. Quantum Query Algorithms for Certain Functions and General Algorithm Construction Techniques. *Quantum Information and Computation V*, Proc. of SPIE Vol. 6573 (SPIE, Bellingham, WA) Article 65730F, 2007
- Vasilieva. Exact Quantum Query Algorithm for Error Detection Code Verification. *Proc. of the Fifth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS),* ISBN 978-80-87342-04-6, pp. 200-207, 2009

In this section, the designing methods and complexity of the exact quantum query algorithm are discussed. Exact algorithms always produce a correct answer with probability $p = 1$. The error probability in algorithms of this kind is not allowed, the said limitation significantly complicating the design of the above algorithms. There are a significant number of efficient quantum algorithms with an error probability already developed. Applying those, the quantum algorithm speedup comparing to the classical case is known to be quadratic [2] or even exponential [1]. However, in certain types of computer software, we cannot allow even a small probability of error, for example, in spacecraft, aircraft, or medical software. For this reason, the development of exact algorithms is very important. By contrast with non-exact algorithms, the largest known complexity separation between the quantum exact and classical deterministic algorithm is only N versus $2N$ for XOR function.

Another category of exact quantum algorithms are algorithms for promise problems [28], [29]. In such problems, the domain of the function is restricted, i.e. input is promised to belong to a subset of all possible inputs. We are not interested in behavior of the algorithm outside this restriction. For promise problems, quantum versus classical separation is known to be exponential (for instance, Deutsch-Jozsa algorithm [28], [25]). Examples of exact quantum query algorithms for promise problems can be found in [30].

A long standing open question is whether it is possible to achieve a larger gap between quantum exact and classical deterministic query complexity of a total function with no error allowed. The conjecture about relation between complexity measures is the following:

$$Q_E(f) \geq \frac{D(f)}{2}.$$

Many authors have worked to either prove or refute this conjecture. This problem has been considered, for instance, in doctoral theses [31], [32]. Examples of a borderline gap of N versus $2N$ have been presented the above theses, but still nobody has been able to improve this result. Examples of exact quantum query algorithms can be found also in [33].

Also in the present thesis, a larger complexity separation for a total Boolean function has not been obtained [2]. The contribution of the author consists of:

- new examples of N versus $2N$ complexity separation;
- techniques for enlarging a set of efficiently computable Boolean functions;
- methods for generating algorithms producing instances of N versus $2N$ complexity separations.

In the first subsection, the author presents two basic exact quantum query algorithms, used as a base for algorithm transformation and designing methods throughout the thesis. In the second subsection, the author introduces a classification of exact quantum query algorithms. Subsequently, quantum algorithm transformation methods are presented, useful for extending a set of Boolean functions efficiently computable in the quantum query model. Furthermore, an exact quantum query algorithm for specific problem using N queries as opposed to the classical $2N$ queries is presented. Finally, the author describes a universal technique for obtaining functions and algorithms with a gap of N versus $2N$ between the quantum and the classical query complexity.

[2] In Chapter 4, computing of multivalued functions is examined. The model has different settings, but interesting complexity gaps (namely, N versus $3N$) are achieved when computing multivalued functions in it.

3.1.1 Basic Exact Quantum Query Algorithms

The author starts with exact quantum query algorithms for two simple problems of a finite input size. The first algorithm computes the three-argument while the second one computes four-argument Boolean function. Both algorithms are interesting: firstly, because they are better suited than the best possible classical algorithms. Secondly, algorithms satisfy specific properties, making them useful for computing more complex Boolean functions. In further sections, these algorithms are used as a base for building advanced algorithms.

To avoid misunderstanding in notation, at the very beginning of thesis, basic quantum algorithms are presented in a very detailed form.

To simplify calculations in the process of algorithm development and debugging, i.e. to automate the verification process, the author developed a simple program using Wolfram *Mathematica* software [34]. The program code for the first example is available in Appendix 1. A tool of this kind can also be used for generation of quantum algorithms.

3.1.1.1 First Basic Exact Algorithm: $Q_E(f) = 2$ versus $D(f) = 3$

The first example is a quantum query algorithm for a three-variable Boolean function saving one query in comparison with the best possible classical deterministic algorithm.

Problem: *Check if all input variable values are equal.*

A potential real life application is, for instance, an automated voting system, where the statement is automatically approved only if all participants have equally voted for acceptance/rejection. The author provides solution for a three-party voting routine. The author reduces the problem to computing the following Boolean function defined by the logical formula:

$$EQUALITY_3(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3).$$

Deterministic complexity: $D(EQUALITY_3) = 3$, by sensitivity on any accepting input, for instance, on $X = 000$. Change of any bit inverses the function value, so it is necessary to query all variables.

Quantum Algorithm 1. An exact quantum query algorithm for $EQUALITY_3$ is presented in Fig. 3.1. Horizontal lines correspond to the amplitudes of the quantum basis states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. It is assumed that calculations will be performed in *bra* notation:

$$\langle final| = \langle \vec{0}| \cdot U_0 \cdot Q_0 \cdot U_1 \cdot Q_1 \cdot U_2 .$$

The computation starts with an initial state $\langle \vec{0}| = (1,0,0,0)$. Three large rectangles correspond to the 4×4 unitary matrices U_0, U_1 and U_2. Two vertical layers of circles specify the queried variable order for queries Q_0 and Q_1. Finally, four small squares at the end of each horizontal line define the assigned function value for each output.

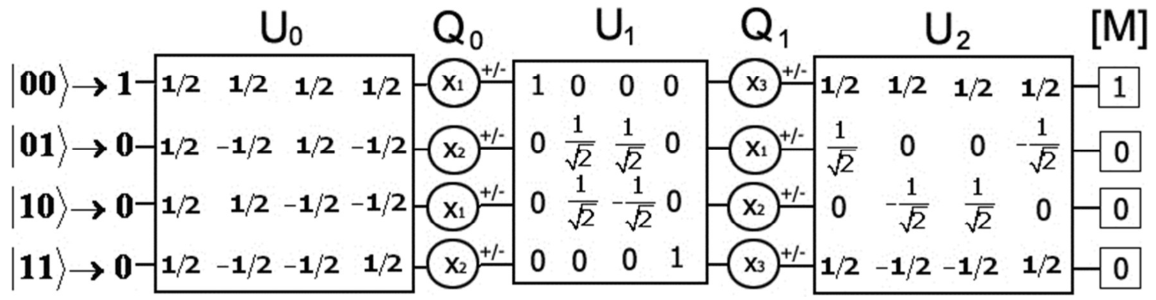


Fig. 3.1 Exact quantum query algorithm with two queries for *EQUALITY*$_3$

For illustration, the author demonstrates the full computation process for rejecting input $X = 011$:

$$\langle\psi| = (1,0,0,0)\begin{pmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & -1/2 & -1/2 \\ 1/2 & -1/2 & -1/2 & 1/2 \end{pmatrix} Q_0 U_1 Q_1 U_2 =$$

$$= \left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)\begin{pmatrix} (-1)^{X1} & 0 & 0 & 0 \\ 0 & (-1)^{X2} & 0 & 0 \\ 0 & 0 & (-1)^{X1} & 0 \\ 0 & 0 & 0 & (-1)^{X2} \end{pmatrix} U_1 Q_1 U_2 =$$

$$= \left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} Q_1 U_2 =$$

$$= \left(\frac{1}{2},0,-\frac{1}{\sqrt{2}},-\frac{1}{2}\right)\begin{pmatrix} (-1)^{X3} & 0 & 0 & 0 \\ 0 & (-1)^{X1} & 0 & 0 \\ 0 & 0 & (-1)^{X2} & 0 \\ 0 & 0 & 0 & (-1)^{X3} \end{pmatrix} U_2 =$$

$$= \left(-\frac{1}{2},0,\frac{1}{\sqrt{2}},\frac{1}{2}\right)\begin{pmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ \frac{1}{\sqrt{2}} & 0 & 0 & -\frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 1/2 & -1/2 & -1/2 & 1/2 \end{pmatrix} =$$

$$= \mathbf{(0,\text{-}1,0,0)} \Rightarrow p(\text{"}0\text{"}) = 1 \Rightarrow \text{"}\mathit{REJECT}\text{"} \Rightarrow \mathit{EQUALITY}_3(011) = 0\,.$$

The computational process for each function input is presented in detail in Appendix 2. The result always equals *EQUALITY*$_3$ value with probability $p = 1$.

3.1.1.2 Second Basic Exact Algorithm: $Q_E(f) = 2$ versus $D(f) = 4$

Subsequently, the author presents quantum query algorithm for the computational problem of comparing elements of a binary string.

Problem: *For a binary string of length 2k check if elements are equal by pairs:*

$$x_1 = x_2 \wedge x_3 = x_4 \wedge x_5 = x_6 \wedge ... \wedge x_{2k-1} = x_{2k}$$

The author presents an algorithm for a string of length four. The problem is reduced to computing the Boolean function of four variables. The Boolean function can be represented by the formula:

$$PAIR\_EQUALITY_4(x_1, x_2, x_3, x_4) = \neg(x_1 \oplus x_2) \wedge \neg(x_3 \oplus x_4).$$

Deterministic complexity: *D*(*PAIR_EQUALITY$_4$*)=4, by sensitivity on any accepting input.

Quantum Algorithm 2. An exact quantum query algorithm for *PAIR_EQUALITY$_4$* is presented in Fig. 3.2.

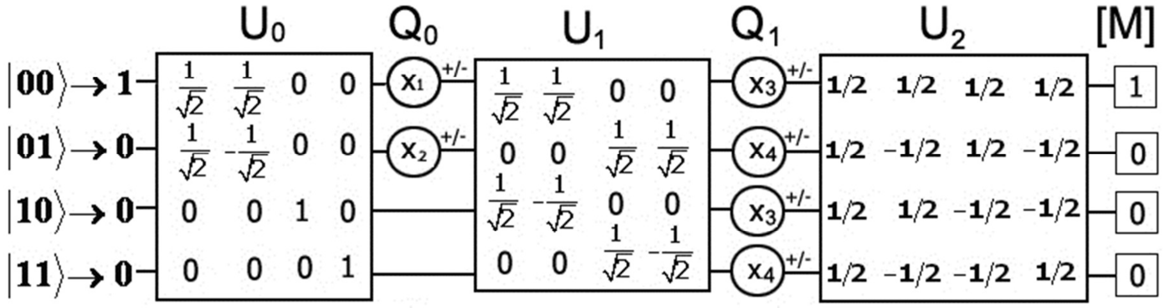


Fig. 3.2 Exact quantum query algorithm with two queries for *PAIR_EQUALITY*$_4$

The computational process for each function input is presented in detail in Appendix 3.

3.1.2 Exact Quantum Query Algorithm Classification

In this subsection, a quantum query algorithm classification is introduced. The classification is required because some of the algorithm transformation and construction methods presented further are applicable only to algorithm instances from specific classes. The classification is based on the properties of an amplitude distribution in the final algorithm state.

Definition 3.1. *An exact quantum query algorithm belongs to* ***Class 1*** *if and only if for any input, the quantum system state before measurement is the following: for exactly one amplitude value* α_i *it's true that* $|\alpha_i|^2 = 1$. *For other amplitudes it's true that* $|\alpha_j|^2 = 0$, *for* $\forall j \neq i$.

XOR$_N$, EQUALITY$_3$ and *PAIR_EQUALITY*$_4$ algorithms all belong to *Class 1.*

Definition 3.2. *An exact quantum query algorithm belongs to* **Class 2+** *if and only if*

- *there is exactly one accepting basic state;*
- *upon any input for its amplitude $\alpha \in \mathbb{C}$ only two values are possible before the final measurement: either $\alpha = 0$ or $\alpha = 1$.*

EQUALITY$_3$ algorithm belongs to *Class 2+*, while *XOR$_N$* and *PAIR_EQUALITY*$_4$ algorithms do not belong to this class.

Definition 3.3. *An exact quantum query algorithm belongs to* **Class 2-** *if and only if*

- *there is exactly one accepting basic state;*
- *upon any input for its amplitude $\alpha \in \mathbb{C}$ only two values are possible before the final measurement: either $\alpha = 0$ or $\alpha = -1$.*

Lemma 3.1 *Any algorithm belonging to Class 2- can be transformed into an algorithm belonging to Class 2+ by applying an additional unitary transformation.*

Proof. Let us assume there is a quantum query algorithm belonging to *Class 2-* and *k* is the number of accepting basis state. To transform the algorithm to *Class 2+* instance the following quantum gate has to be added as the last algorithm step:

$$U = (u_{ij}) = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j \neq k \\ -1, & \text{if } i = j = k \end{cases}$$

□

Definition 3.4. *An exact quantum query algorithm belongs to* **Class 3** *if and only if*

- *it belongs to Class 1;*
- *there is exactly one accepting basic state;*
- *upon any input the amplitude value of accepting state before measurement is $\alpha \in \{-1, 0, 1\}$.*

XOR$_N$, EQUALITY$_3$ and *PAIR_EQUALITY*$_4$ algorithms all belong to *Class 3*.

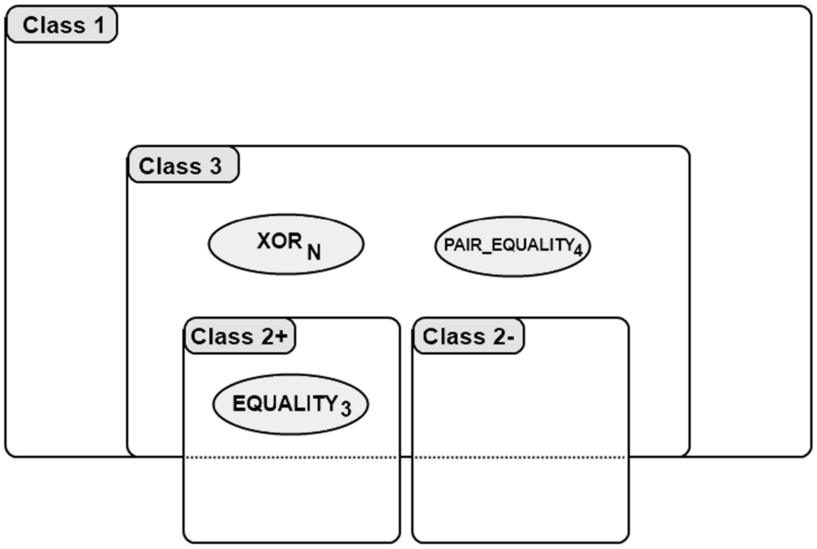


Fig. 3.3 Visualization of exact quantum query algorithm classes

3.1.2.1 Constructing Algorithms from Class 2+

In this subsection, an approach for designing algorithms of *Class 2+* is presented. Algorithms of this type will be required for a method of computing conjunctions $f_1 \wedge f_2$ in Section 3.2.1.4.

Given an arbitrary classical deterministic decision tree, it is possible to convert it into an exact quantum query algorithm which uses the same number of queries.

A classical query to the black box can be simulated with a quantum query algorithm construction presented in Fig. 3.4.

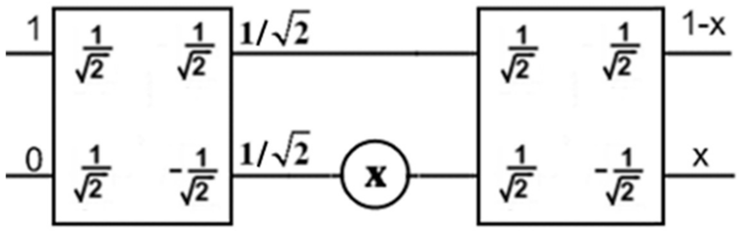


Fig. 3.4 Quantum query algorithm construction for simulating a classical query

After the second Hadamard gate the quantum state $(1,0)^T$ is obtained if $x = 0$, the state $(0,1)^T$ is obtained if $x = 1$. Then, querying other variables may be continued by logically splitting the algorithm flow into two separate parallel threads, etc. Algorithm constructions from Fig. 3.4 are simply concatenated one by one.

Subsequently, a complete example of converting a classical decision tree for computing $AND(x_1, x_2)$ into an exact quantum query algorithm is demonstrated. Fig. 3.5 shows a classical decision tree. Fig. 3.6 shows the corresponding exact quantum query algorithm.

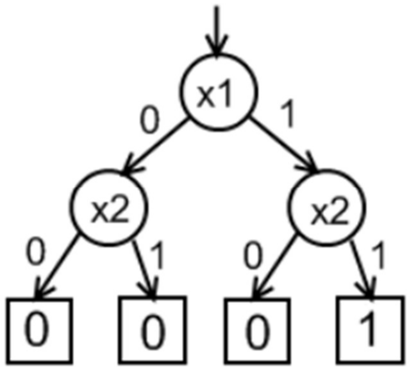


Fig. 3.5 A classical deterministic decision tree for $AND(x_1, x_2)$

It should be noted that, although such conversion is possible, it is not optimal. For instance, it is well known that *XOR* can be computed in a quantum model using 50% less queries than required in a classical model.

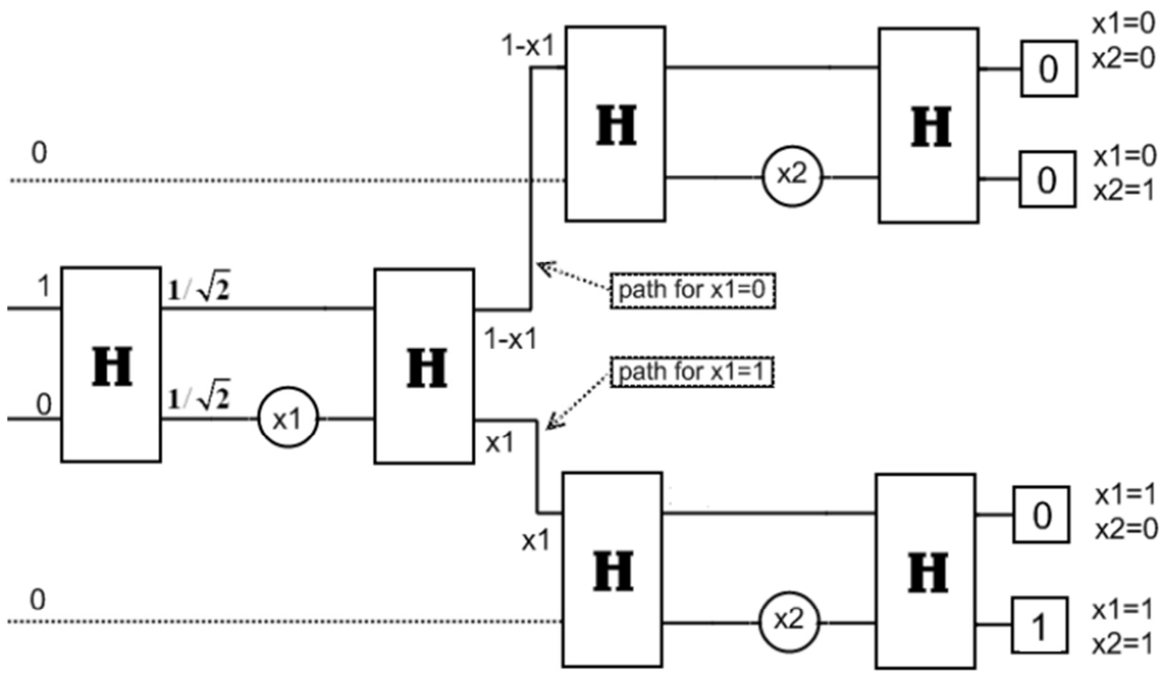


Fig. 3.6 An exact quantum query algorithm for computing $AND(x_1, x_2)$

Theorem 3.1 *A set of exact quantum query algorithms satisfying the condition of Class 2+ is infinite.*

Proof. Any deterministic decision tree having exactly one leaf with the output value "1" obviously might be converted into an algorithm of the class *Class 2+*. □

3.1.3 Algorithm Transformation Methods

In this subsection, quantum query algorithm transformation methods are introduced useful for enlarging a set of exactly computable Boolean functions. Each method receives an exact quantum query algorithm on input, processes it according to the rules producing as the result a slightly different exact algorithm computing another function.

3.1.3.1 Output Value Assignment Inversion

The first method is the simplest one. To transform an original algorithm, the assigned function value for each output has to be changed to the opposite. The method is described in Table 3.1.

Table 3.1 Description of the first transformation method

First transformation method - Output value assignment inversion
Input. An arbitrary exact quantum query algorithm that computes *f*(*X*). **Transformation actions.** • For each algorithm output change assigned function value to opposite. If original assignment was $QM = (\vert 0\rangle \to R_1,...,\vert m\rangle \to R_m)$, where $R_i \in \{0,1\}$, then it is transformed to $QM' = (\vert 0\rangle \to \overline{R}_1,...,\vert m\rangle \to \overline{R}_m)$, where $\overline{R}_i = 1 - R_i$. **Output.** An exact quantum query algorithm that computes $\overline{f}(X)$.

3.1.3.2 Output Value Assignment Permutation

The next method is applicable only to exact quantum query algorithms with specific properties. The method is described in Table 3.2.

Table 3.2 Description of the second transformation method

Second transformation method - Output value assignment permutation
Input. • An exact quantum query algorithm that belongs to *Class 1* and computes *f*(*X*). • Permutation σ of the set $OutputValues = \{R_1, R_2, ..., R_m\}$. **Transformation actions.** • Permute function values assigned to outputs in order specified by σ. If original assignment was $QM = (\vert 0\rangle \to R_1,...,\vert m\rangle \to R_m)$, where $R_i \in \{0,1\}$, then it is transformed to $QM' = (\alpha_1 \to \sigma(R_1),...,\alpha_m \to \sigma(R_m))$. **Output.** An exact quantum query algorithm for different function *g*(*X*).

Application of the above method does not break the exactness of the quantum query algorithm because of the terms of *Class 1* non-zero amplitude value is invariably obtained in exactly one output before the measurement. Since the function value is clearly specified for each output, the specific value will always be observed with probability $p = 1$ for any input.

The structure of new function *g*(*X*) strictly depends on the internal properties of the original algorithm. To explicitly define new function it is necessary to examine the original algorithm behavior on each input and construct the truth table for the new output value assignment.

The application of this method to the basic algorithm for computing function $EQUALITY_3$ is illustrated in Fig. 3.7.

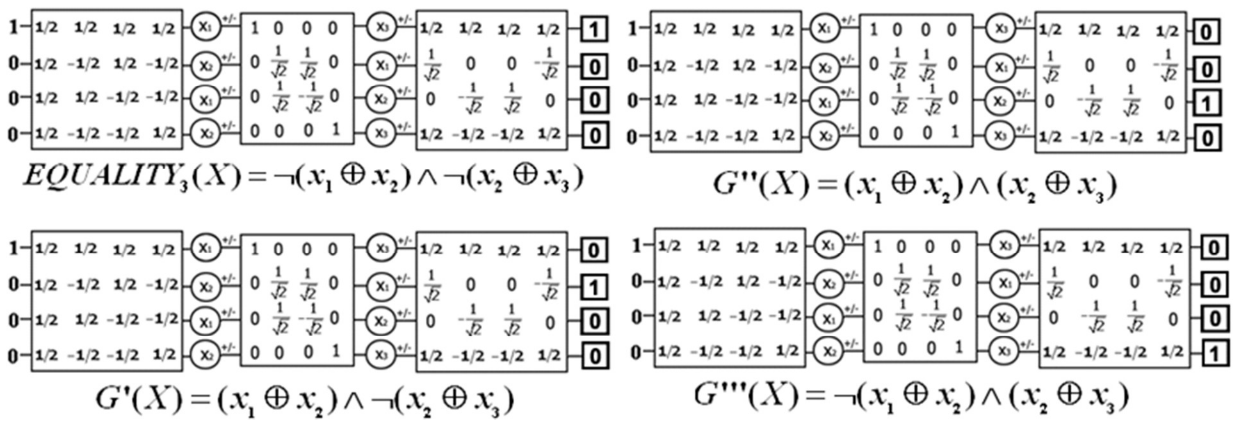

$EQUALITY_3(X)=\neg(x_1\oplus x_2)\wedge\neg(x_2\oplus x_3)$
$G''(X)=(x_1\oplus x_2)\wedge(x_2\oplus x_3)$
$G'(X)=(x_1\oplus x_2)\wedge\neg(x_2\oplus x_3)$
$G'''(X)=\neg(x_1\oplus x_2)\wedge(x_2\oplus x_3)$

Fig. 3.7 Application of the second transformation method to *EQUALITY*3 algorithm

3.1.3.3 Permutation of Query Variables

Let us assume σ is a permutation of the set $\{1,2,...,N\}$, where elements correspond to variable numbers. The function *g*(*X*) is obtained by permutation of *f*(*X*) variables if:

$$g(X)=f\left(x_{\sigma(1)},x_{\sigma(2)},...,x_{\sigma(N)}\right).$$

For example, function $PAIR\_EQUALITY_4(X)=\neg\left(x_1\oplus x_2\right)\wedge\neg\left(x_3\oplus x_4\right)$ is taken as a base and permutation $\sigma=\begin{pmatrix}1\ 2\ 3\ 4\\2\ 4\ 1\ 3\end{pmatrix}$ is applied to its definition. As a result, a slightly different function is obtained:

$$G_4(X)=PAIR\_EQUALITY_4(x_2,x_4,x_1,x_3)=\neg(x_2\oplus x_4)\wedge\neg(x_1\oplus x_3).$$

In the third transformation method, the idea of variable permutation is applied to the quantum query algorithm definition. Applying the third transformation method described in Table 3.3, variable values influence the new algorithm process according to the order specified by a permutation σ, and thus the algorithm computes *g*(*X*) instead of *f*(*X*).

Table 3.3 Description of the third transformation method

Third transformation method – Permutation of query variables
Input. • An arbitrary exact quantum query algorithm that computes *f*(*X*). • The permutation σ of variable numbers $VarNum = \{0,1,...,N\}$. **Transformation actions.** • Apply permutation of variable numbers σ to all query transformations. If original *i*-th query is defined as $Q_i = \begin{pmatrix} (-1)^{x_{k_1}} & ... & 0 \\ ... & ... & ... \\ 0 & ... & (-1)^{x_{k_m}} \end{pmatrix}$, then it is transformed to $Q_i = \begin{pmatrix} (-1)^{x_{\sigma(k_1)}} & ... & 0 \\ ... & ... & ... \\ 0 & ... & (-1)^{x_{\sigma(k_m)}} \end{pmatrix}$, where $k_i \in \{1,..,N\}$. **Output.** An exact quantum query algorithm computing a function: $g(X) = f\left(x_{\sigma(1)}, x_{\sigma(2)}, ..., x_{\sigma(N)}\right)$.

3.1.3.4 Results of Applying Transformation Methods to Basic Algorithms

Subsequently, the author demonstrates the transformation methods' application results for the basic exact algorithms from Section 3.1.1.

Using the algorithm for *EQUALITY*$_3$ function as a base it is possible to obtain a set of three-argument Boolean functions (denoted by *QFunc3*), where for each function there is an exact quantum query algorithm, which computes it with two queries. Algorithms for computing different functions are produced using the first and the second transformation methods. The third method (permutation of variables) does not produce algorithms for new functions, because all variables have identical roles in the definition of *EQUALITY*$_3$. In total, eight different functions are obtained: $|QFunc3| = 8$. Functions are presented in Table 3.4.

Table 3.4. Results of applying transformation methods to *EQUALITY*$_3$ algorithm (the set *QFunc3*)

X	*EQUALITY*	Output value assignment permutation			Output value assignment inversion			
	(1,0,0,0)	(0,1,0,0)	(0,0,1,0)	(0,0,0,1)	(0,1,1,1)	(1,0,1,1)	(1,1,0,1)	(1,1,1,0)
000	1	0	0	0	0	1	1	1
001	0	0	0	1	1	1	1	0
010	0	0	1	0	1	1	0	1
011	0	1	0	0	1	0	1	1
100	0	1	0	0	1	0	1	1
101	0	0	1	0	1	1	0	1
110	0	0	0	1	1	1	1	0
111	1	0	0	0	0	1	1	1
$D(f)$	3	3	3	3	3	3	3	2
$\boldsymbol{Q_E(f)}$	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**

Using the algorithm for *PAIR_EQUALITY*$_4$ function it is possible to obtain a set of four-argument Boolean functions (denoted by *QFunc4*), where for each function there is an exact quantum query algorithm which computes it with two queries. This time, application of all three methods produces different functions. In total, 24 different functions are obtained: $|QFunc4| = 24$. A half of *QFunc4* set (generated by application of the second and third methods) is presented in Table 3.5.

Proposed quantum query algorithm transformation methods are useful for enlarging a set of efficient quantum algorithms. Transformation methods can be applied to every new exact quantum query algorithm, thus constructing a larger set of efficiently computable Boolean functions. Moreover, exact algorithms obtained this way further can be used as building blocks for more complex algorithms (see next sections).

Table 3.5. Results of applying transformation methods to *PAIR_EQUALITY*$_4$ algorithm (half of the set *QFunc4*)

X	*PAIR EQUALITY*	2nd method			3rd method & 2nd method $\sigma_{VarNum} = \begin{pmatrix}1234\\1324\end{pmatrix}$				3rd method & 2nd method $\sigma_{VarNum} = \begin{pmatrix}1234\\3124\end{pmatrix}$			
	$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$	$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$
0000	**1**	0	0	0	0	0	0	**1**	**1**	0	0	0
0001	0	**1**	0	0	0	**1**	0	0	0	**1**	0	0
0010	0	0	**1**	0	0	**1**	0	0	0	0	**1**	0
0011	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**
0100	0	**1**	0	0	0	0	**1**	0	0	0	**1**	0
0101	**1**	0	0	0	**1**	0	0	0	0	0	0	**1**
0110	0	0	0	**1**	**1**	0	0	0	**1**	0	0	0
0111	0	0	**1**	0	0	0	**1**	0	0	**1**	0	0
1000	0	0	**1**	0	0	0	**1**	0	0	**1**	0	0
1001	0	0	0	**1**	**1**	0	0	0	**1**	0	0	0
1010	**1**	0	0	0	**1**	0	0	0	0	0	0	**1**
1011	0	**1**	0	0	0	0	**1**	0	0	0	**1**	0
1100	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**
1101	0	0	**1**	0	0	**1**	0	0	0	0	**1**	0
1110	0	**1**	0	0	0	**1**	0	0	0	**1**	0	0
1111	**1**	0	0	0	0	0	0	**1**	**1**	0	0	0
D(*f*)	4	4	4	4	4	4	4	4	4	4	4	4
Q_E(*f*)	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**

3.1.4 Exact Quantum Query Algorithm for Verifying Repetition Code

In this subsection, the author examines a specific problem: the verification of the codeword encoded by the repetition code for error detection. First, repetition codes are introduced and a Boolean function is defined for their verification. Secondly, the author shows that classically, for an N-bit message, values of all N variables must be queried in order to detect an error. Finally, an exact quantum query algorithm for N-bit codeword verification is presented that uses only $N/2$ queries to the black box.

3.1.4.1 Error Detection and Repetition Codes

In this subsection, a problem related to information transmission across a communication channel is investigated. The bit message is transmitted from a sender to a receiver. Over the transmission process, the information may be corrupted. Because of the noise in the channel or adversary intervention, some bits may disappear, become reverted, or even added. Various schemes exist to detect errors during transmission. In any case, a verification step is required after transmission. The received codeword is checked using defined rules and, as a result, a conclusion is made as to whether errors are present.

The author examines a transmission error detection scheme known as repetition codes. A repetition code is a (r, N) coding scheme repeating each N-bit block r times [35].

Example.

- Using a (3,1) repetition code, the message m = 101 is encoded as c = 111000111.
- Using a (2,2) repetition code, the message m = 1011 is encoded as c = 10101111.
- Using a (2,3) repetition code, m = 111000 is encoded as c = 111111000000.

Verification procedure for the repetition code involves checking weather in each group of r the consecutive blocks of size N all blocks are equal.

There are related steps taken in [30] for error correcting codes such as Hamming codes and Reed-Solomon codes. 25% complexity improvement has been achieved quantumly by making only $3/4 \cdot m$ queries when detecting Hamming codewords of length $m = 2^n - 1$. 50% complexity improvement has been demonstrated for Reed-Solomon codes of even length $m = n(2^n - 1)$.

The author starts with the verification of the (2,1) repetition code. The verification process can be expressed naturally as computing a Boolean function in a query model. It is assumed that the codeword to be checked is located in a black box. The Boolean function to be computed by the query algorithm is defined as follows.

Definition 3.5. *The Boolean function $VERIFY_N(X)$, where $N = 2k$, $X = (x_1, x_2, ..., x_{2k})$ is defined to have a value "1" if and only if variables are equal by pairs.*

$$VERIFY_{2k}(X) = \begin{cases} 1, \textit{if } (x_1 = x_2) \wedge (x_3 = x_4) \wedge (x_5 = x_6) \wedge ... \wedge (x_{2k\text{-}1} = x_{2k}) \\ 0 \text{ , } \textit{otherwise} \end{cases}$$

Example.

The Boolean function $VERIFY_4(X)$ has the following accepting inputs:

{0000, 0011, 1100, 1111}.

3.1.4.2 Deterministic Complexity of *VERIFY*$_N$

Fig. 3.8 demonstrates a classical deterministic decision tree, which computes Boolean function $VERIFY_4(x_1, x_2, x_3, x_4)$.

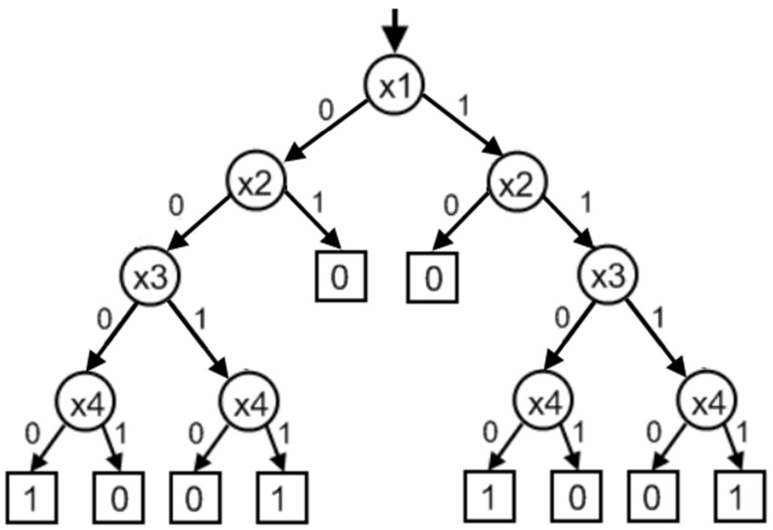


Fig. 3.8 A classical deterministic decision tree for computing $VERIFY_4$

Theorem 3.2. $D(VERIFY_N) = N$.

Proof. By function sensitivity on any accepting input, for instance, on X = 11...1. Inversion of any bit inverts the function value, because a pair of bits with different values appears:

$$s(VERIFY_N) = N \quad \Rightarrow \quad D(VERIFY_N) = N.$$

□

3.1.4.3 Computing the Function *VERIFY*$_N$ in the Quantum Query Model

The proposed approach to computing the Boolean function *VERIFY*$_N$ in a quantum query model is based on an exact quantum query algorithm for the *XOR* function.

Theorem 3.3. *There exists an exact quantum query algorithm that computes the Boolean function VERIFY*$_N$*(X) using N/2 queries:* $Q_E(VERIFY_N) = N/2$.

Proof. The definition of the *VERIFY*$_N$ function can be re-formulated as follows.

$$VERIFY_{2k}(X) = \begin{cases} 1, \text{ } if \text{ } \neg(x_1 \oplus x_2) \wedge \neg(x_3 \oplus x_4) \wedge \neg(x_5 \oplus x_6) \wedge ... \wedge \neg(x_{2k\text{-}1} \oplus x_{2k}) \\ 0 \text{ }, \text{ } otherwise \end{cases}$$

Exact quantum algorithm for computing the Boolean function $f(x_1, x_2) = \neg(x_1 \oplus x_2)$ with one query is presented in Fig. 3.9. An algorithm for *VERIFY*$_N$ is composed using an algorithm for $f(x_1, x_2) = \neg(x_1 \oplus x_2)$ as building blocks.

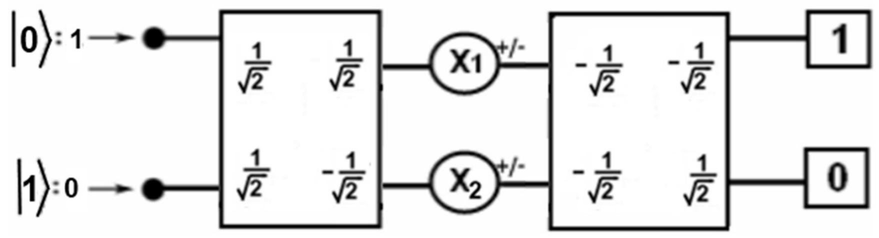


Fig. 3.9 Exact quantum query algorithm for computing $f(x_1,x_2)=\neg(x_1\oplus x_2)$

First, an algorithm for $f(x_1,x_2)=\neg(x_1\oplus x_2)$ is executed for variables x_1 and x_2. Then, the second instance of an algorithm for computing $f(x_1,x_2)=\neg(x_1\oplus x_2)$ is concatenated to the accepting output of the first algorithm instance (see Fig. 3.9). This time, the algorithm is executed for variables x_3 and x_4. The process is continued until all variables of *VERIFY*$_N$ are queried. The algorithm has only one accepting output, which is the first output of the last sub-algorithm.

A schematic view of the described approach is depicted in Fig. 3.10. It is evident that the total number of queries is *N*/2. □

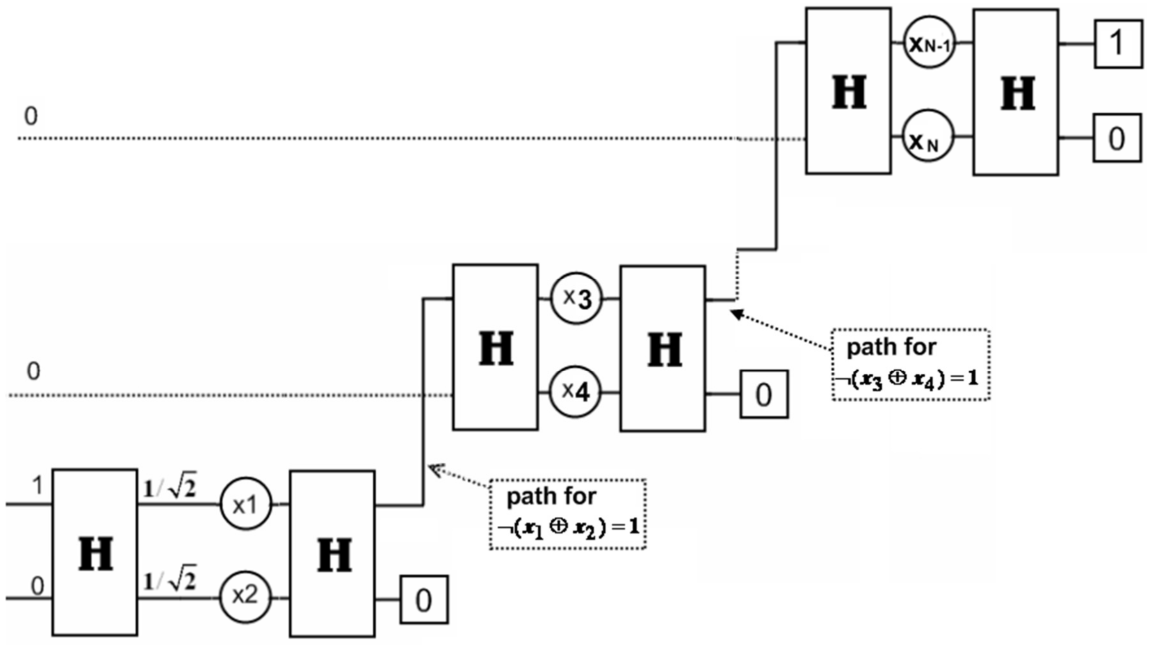


Fig. 3.10 Algorithm for computing the Boolean function *VERIFY*$_N$

3.1.4.4 Application of the above approach to the String Equality Problem

The described approach can be adapted for solving the computational problem of testing the equality of two binary strings. This is a well-known task, which can be used as a sub-routine in various algorithms.

A quantum query algorithm for the Boolean function *VERIFY*$_N$ checks the equality of variables by pairs, i.e. $(x_1=x_2)\wedge(x_3=x_4)\wedge\ldots\wedge(x_{N-1}=x_N)$. At the same time, this algorithm checks also the equality of two binary strings, $Y = x_1x_3x_5...x_{N-1}$ and $Z = x_2x_4x_6...x_N$. Therefore, the algorithm can be easily used not only to verify the repetition codes, but also for checking equality of the binary strings.

3.1.4.5 Verification of the (*r*,1) Repetition Code

Now, let us consider the $(r,1)$ repetition code, where each bit is repeated r times during encoding. The verification procedure for a codeword encoded using the given code consists of checking the equality of all bits in each sequence of r bits.

The Boolean function $EQUALITY_r$ is defined as

$$EQUALITY_r(X) = \begin{cases} 1, \textit{if} \ (x_1 = x_2) \wedge (x_2 = x_3) \wedge (x_3 = x_4) \wedge ... \wedge (x_{r-1} = x_r) \\ 0\ , \textit{otherwise} \end{cases}$$

The Boolean function that corresponds to the verification procedure is defined as

$$VERIFY^r_{r\cdot N}(X) = \begin{cases} 1, \textit{if} \ EQUALITY(x_1,...,x_r) \wedge EQUALITY(x_{r+1},...,x_{2r}) \wedge ... \\ \qquad\qquad ... \wedge \ EQUALITY\left(x_{(N-1)r+1},...,x_{Nr}\right) \\ 0\ , \textit{otherwise} \end{cases}$$

Theorem 3.4. *Deterministic complexity of the Boolean function $VERIFY^r_{r\cdot N}(X)$ is equal to the number of variables: $D(VERIFY^r_{r\cdot N}) = rN$.*

Proof. By function sensitivity on any accepting input. Inversion of any bit inverts the function value because a pair of bits with different values appears.

$$s(VERIFY^r_{r\cdot N}) = rN \quad \Rightarrow \quad D(VERIFY^r_{r\cdot N}) = rN .$$

□

Theorem 3.5. *There exists an exact quantum query algorithm that computes the Boolean function $VERIFY^r_{r\cdot N}(X)$ using $(r-1)N$ queries:*

$$Q_E(VERIFY^r_{r\cdot N}) = (r-1)N .$$

Proof. To speed up the verification procedure, the fact is used that *XOR* of two bits can be computed with a single quantum query. The Boolean function $EQUALITY_r$ can be expressed using operations $\oplus$, $\wedge$ and $\neg$:

$$EQUALITY_r(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3) \wedge \neg(x_3 \oplus x_4) \wedge ... \wedge \neg(x_{r-1} \oplus x_r) .$$

This logical formula contains (r-1) clauses consisting of *XOR* of two bits each. Using the approach described in the proof of Theorem 3.3, it is possible to compose an exact quantum query algorithm computing $EQUALITY_r$ using (r-1) quantum queries. This algorithm has a single accepting output. Next, an algorithm for $EQUALITY_r$ is used as a building block for composing an algorithm for computing $VERIFY^r_{r\cdot N}$. The resulting algorithm uses $(r-1)N$ queries to determine the value of the Boolean function $VERIFY^r_{r\cdot N}$ exactly. □

3.1.5 Algorithm Concatenation Methods

This section generalizes the previously described approaches for quantum query algorithm design and presents methods for generating complex quantum algorithms from simple building blocks. Specifically, these methods can be used for generating examples of *N* versus 2*N* gaps between quantum and classical query complexity of the function.

Theorem 3.6. *Given an exact quantum query algorithms QA_1, QA_2 computing Boolean functions $f_1(X), f_2(Y)$ with complexity $Q_E(QA_1) = m_1$, $Q_E(QA_2) = m_2$, it is possible to build a new exact quantum query algorithm QA_3 for computing the conjunction $f_3(XY) = f_1(X) \wedge f_2(Y)$ with complexity $Q_E(QA_3) = m_1 + m_2$.*

Proof. The approach is similar to the one demonstrated in Section 3.1.4. This time whole algorithms are concatenated together. To achieve computing a conjunction $f_3 = f_1 \wedge f_2$ on the part of the resulting algorithm, it is necessary to concatenate an instance of the second algorithm QA_2 to each accepting output of the first algorithm QA_1. Amplitudes of all rejecting states remain unchanged until the end of the computation.

The approach is schematically demonstrated in Fig. 3.11. The first state of the first algorithm QA_1 has an initial amplitude value 1. In this example, algorithm QA_1 has three accepting states. So, in the second stage of the new algorithm, there are three instances of the second algorithm QA_2 running in parallel. Each accepting state of QA_1 is connected to the first state of the associated second algorithm QA_2.

Let us show that obtained quantum algorithm computes a function:

$$f_3(XY) = f_1(X) \wedge f_2(Y).$$

If $f_1(X) = 0$, then after the last transformation of QA_1 only the rejecting basis states will have non-zero amplitude values. These amplitudes are not further modified until the end of the computation, so after the final measurement value "0" will be observed with probability $p = 1$.

If $f_1(X) = 1$, then after the last transformation of QA_1 only the accepting basis states will have non-zero amplitude values. Let us assume that after the last transformation, for some arbitrary accepting input *X* there are $k \geq 1$ basis states with positive amplitude values $\alpha_1, ..., \alpha_k$ $(\sum_{i=1}^{k} |\alpha_i|^2 = 1)$. Subsequently, *k* instances of the algorithm QA_2 will be initialized with these amplitudes and will continue execution by applying parallelized transformations.

- If $f_2(Y) = 0$, then for all *k* instances of QA_2 having received a positive amplitude value, after the last transformation only a subset of the rejecting states will have non-zero amplitude values. Amplitudes of all the accepting states of QA_2 will be equal to zero. The final measurement produces result "0" with probability $p = 1$.
- If $f_2(Y) = 1$, then vice versa - only the subset of the accepting states will have non-zero amplitude values, while amplitudes of all the rejecting states will be equal to zero. The final measurement produces result "1" with probability $p = 1$.

The described behavior ensures that the obtained algorithm QA_3 computes the conjunction of basic functions: $f_3 = f_1 \wedge f_2$. Total number of queries is $m_1 + m_2$.

□

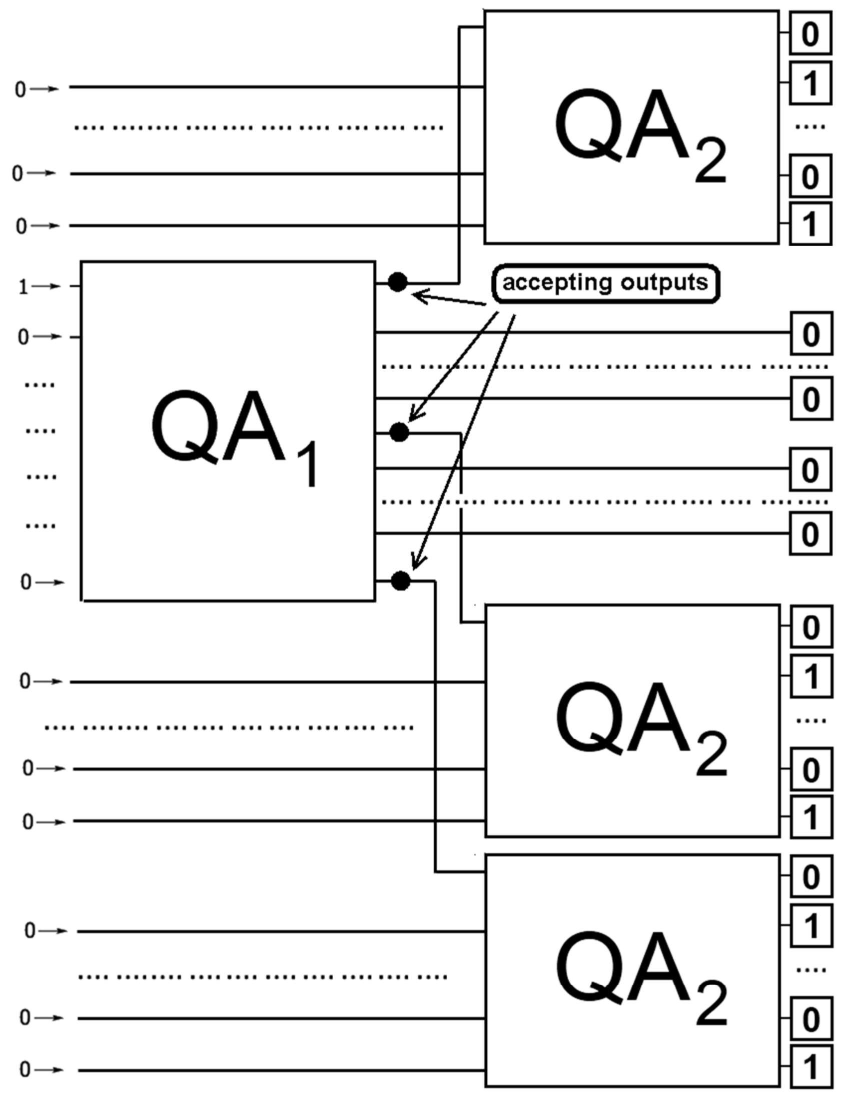


Fig. 3.11 Algorithm concatenation method for computing conjunction $f_3 = f_1 \wedge f_2$

An important property of the method is the opportunity to use the obtained algorithms as building blocks in the repeated application of a method for constructing even more complex algorithms. It is even possible to apply the method recursively.

Moreover, using the algorithm for XOR_N and exact quantum query algorithms for functions from the set *QFunc*4 (see Table 3.5) as building blocks, it is possible to construct an infinite set of exact algorithms. For all mentioned basic functions there is a gap of N versus $2N$ between the exact quantum and the classical deterministic query complexity. Consequently, for each algorithm with complexity K from an infinite set obtained by concatenation, the best possible classical deterministic algorithm for computing the same function requires $2K$ classical queries.

Theorem 3.7. *Given an exact quantum query algorithms QA_1, QA_2 computing Boolean functions $f_1(X)$, $f_2(X)$ with complexity $Q_E(QA_1) = m_1$, $Q_E(QA_2) = m_2$, it is possible to build a new exact quantum query algorithm QA_3 for computing the disjunction $f_3(XY) = f_1(X) \vee f_2(Y)$ with complexity $Q_E(QA_3) = m_1 + m_2$.*

Proof. The algorithm constructing approach is the same: algorithms are concatenated one by one. This time, the difference is that a separate instance of QA_2 has to be connected to each rejecting state of the first algorithm QA_1 (see Fig. 3.12).

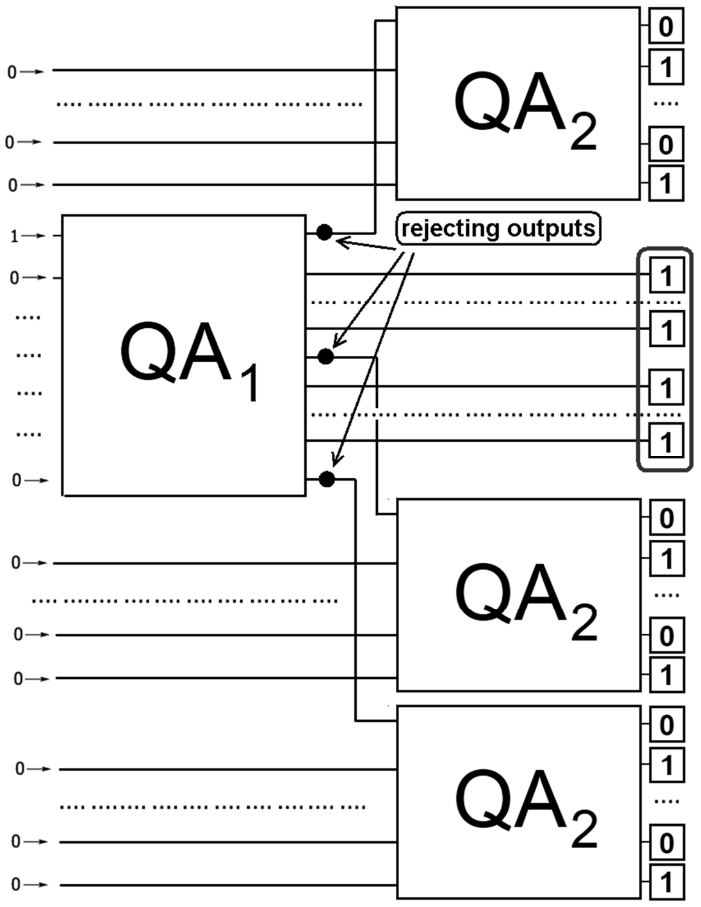


Fig. 3.12 Algorithm concatenation method for computing disjunction $f_3 = f_1 \vee f_2$

If $f_1(X) = 1$, then only the accepting states of QA_1 will have non-zero amplitude values. These amplitudes are not modified anymore until the end of the computation, so after the final measurement value "1" will be observed with probability $p = 1$.

If $f_1(X) = 0$, then only the accepting states of QA_1 will have non-zero amplitude values. Next, k instances of the algorithm QA_2 will be initialized with these amplitudes.

- If $f_2(Y) = 0$, then for all k instances of QA_2 only subset of the rejecting states will have non-zero amplitude values. The final measurement produces result "0" with probability $p = 1$.

- If $f_2(Y) = 1$, then vice versa, only the subset of the accepting states will have non-zero amplitude values. The final measurement produces result “1” with probability $p = 1$.

The described behavior ensures that the obtained algorithm QA_3 computes the disjunction of basic functions: $f_3 = f_1 \vee f_2$. The total number of queries is $m_1 + m_2$.

□

Again, an infinite set of exact algorithms can be constructed using this method for which a gap between quantum exact and classical deterministic query complexity is N versus $2N$.

Moreover, methods for conjunction and disjunction can be used interchangeably, thus obtaining even a larger variety of functions.

Theorem 3.8. *Given an exact quantum query algorithms QA_1, QA_2 computing Boolean functions $f_1(X)$, $f_2(X)$ with complexity $Q_E(QA_1) = m_1$, $Q_E(QA_2) = m_2$, it is possible to build a new exact quantum query algorithm QA_3 for computing XOR: $f_3(XY) = f_1(X) \oplus f_2(Y)$ with complexity $Q_E(QA_3) = m_1 + m_2$.*

Proof. In a similar way as in case of conjunction and disjunction, instances of the algorithm QA_2 have to be concatenated to algorithm’s QA_1 outputs. However, this time the second algorithm has to be concatenated to all outputs – both, the accepting and the rejecting. Additionally, for QA_2 instances concatenated to accepting outputs of QA_1, the result values assigned to the states have to be inversed. The quantum query algorithm obtained in this way will compute $f_3 = f_1 \oplus f_2$ exactly and complexity will be $Q_E(QA_3) = m_1 + m_2$.

□

An example of this approach is demonstrated in Fig. 3.13. The first algorithm is an algorithm for computing XOR_2 (see Section 0). The second algorithm is an algorithm for $PAIR\_EQUALITY_4$ (see Section 3.1.1.2). The resulting algorithm computes the function:

$$f(x_1,...,x_6) = XOR_2(x_1,x_2) \oplus PAIR\_EQUALITY_4(x_3,...,x_6) =$$
$$= (x_1 \oplus x_2) \oplus \left(\neg(x_3 \oplus x_4) \wedge \neg(x_5 \oplus x_6)\right)$$

Algorithm complexity is $Q_E(f) = 3$.

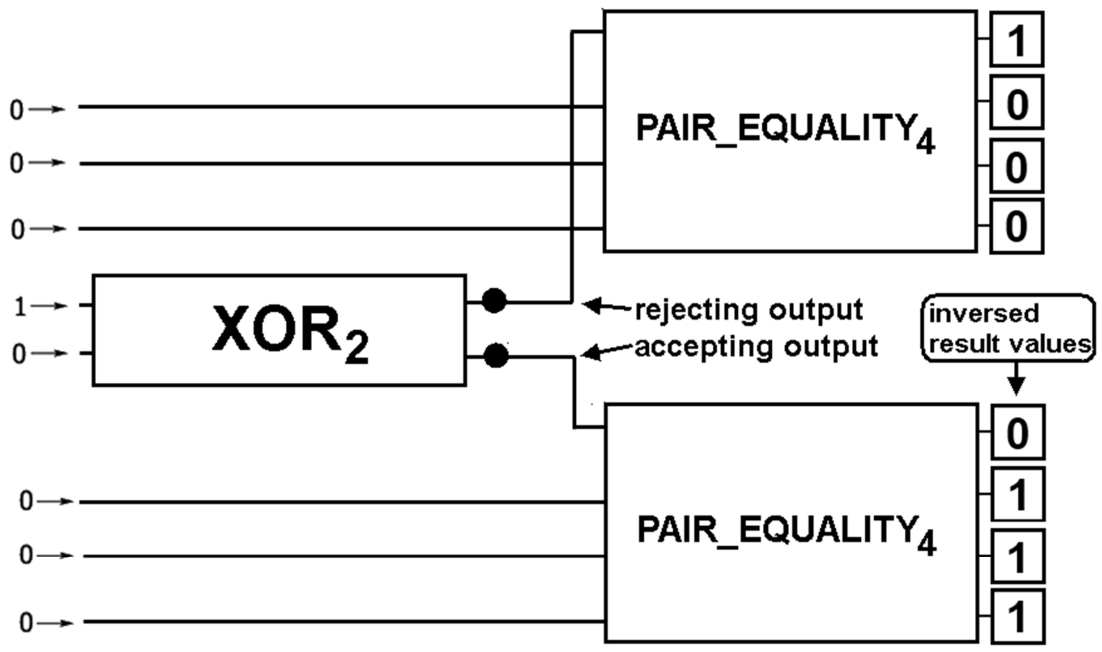


Fig. 3.13 Example of computing $f = XOR_2 \oplus PAIR\_EQUALITY_4$

3.1.6 Conclusion and Open Problems

In this section, the author examined the designing of exact quantum query algorithms.

First, two exact quantum query algorithms for three and four argument Boolean functions were presented. Both algorithms used less queries comparing to the best possible classical algorithms. Algorithms are used in further sections as a base for algorithm transformation and concatenation methods. The author also introduced the classification of exact quantum query algorithms.

Next, the techniques were presented that enable transformation of an existing quantum query algorithm for a certain Boolean function so that the resulting algorithm computes a function with a different logical structure. Methods were illustrated by applying them to two basic exact algorithms.

Subsequently, the verification of repetition codes for the error detection was examined. The author has presented the verification procedure as an application of a query algorithm to an input codeword contained in a black box. An exact quantum query algorithm is presented allowing verifying a codeword of length *N* using only *N*/2 queries to the black box. The algorithm saves exactly half the number of queries comparing to the classical case. For the total Boolean function, this result repeats the largest difference between the classical and the quantum algorithm complexity known today within the query model.

Finally, algorithm concatenation methods were presented giving an opportunity to construct composite algorithms for $f_1 \wedge f_2$, $f_1 \vee f_2$ and $f_1 \oplus f_2$ using existing algorithms for sub-functions as sub-routines. The techniques presented make the task of building efficient exact quantum query algorithms for the arbitrary Boolean functions easier. Using the described approach, it is possible to generate an infinite set of examples of *N* versus 2*N* gaps between the quantum and the classical query complexity for various functions.

There are many potential directions for the future research in the area of the exact quantum query algorithm design.

Given the useful properties of the above algorithms and regarding the presented results, it would be interesting to introduce general methods for designing exact quantum query algorithms from *Class* 1, *Class* 2+, *Class* 3. Applying transformation and concatenation methods to the base algorithms would significantly enlarge a set of efficiently computable Boolean functions. The ultimate goal is to develop a framework for building ad-hoc efficient quantum algorithms for arbitrary functions.

Then, it would be very interesting to find an example of *N* versus 2*N* complexity separation, where the structure of the computable Boolean function does not depend on *XOR* operation.

Finally, the most significant open question still remaining is as follows: is it possible to increase an exact algorithm performance for more than twice by use of the quantum tools?

3.2 Bounded-Error Quantum Query Algorithms

This section is based on the paper

- A. Vasilieva, T. Mischenko-Slatenkova. Quantum Query Algorithms for Conjunctions. *Proc. of the 9th International Conference UC 2010,* Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol. 6079/2010, ISBN: 978-3-642-13522-4, pp. 140-151, 2010

One of the most important open problems in quantum computing is the proper understanding why quantum algorithms can have any advantages over probabilistic algorithms. The author concentrates on the case when these advantages are least expected, namely, when the computing device is finite and so is the computation time. In this section, the quantum bounded-error versus classical probabilistic query algorithms is discussed.

3.2.1 Quantum Query Algorithms for Conjunctions

In this section, the results of constructing quantum query algorithms for a set of Boolean functions based on the *AND* Boolean operation are presented. Bounded-error algorithms are examined, which output a correct answer with some probability.

Every Boolean function can be presented as a logical formula in a conjunctive normal form (CNF). Formula is in CNF if it is a conjunction (*AND*s) of disjunctions (*OR*s) of variables or negated variables. Therefore, a fast and efficient algorithm for conjunction is necessary. While there exists an exact quantum algorithm for *XOR* that goes with *N*/2 queries, exact quantum algorithms for disjunction and conjunction require *N* queries in *N*-bit case, this is a proved lower bound [14]. To enlarge the complexity gap between the classical and quantum algorithms it is important to work out as efficient quantum algorithm for conjunction as possible.

Grover's search algorithm [2] potentially could be adjusted to compute *N*-bit conjunction using $O\left(\sqrt{N}\right)$ queries. However, such approach would be more efficient than the classical one just for sufficiently large *N*. In some tasks there is a need to evaluate conjunction of rather small number of variables, while the total number of such distinct evaluations may be huge (e.g. evaluation of conditional Boolean statements in computer programs). This reasoning motivated the author to search for other approaches for computing conjunctions, which would be preferable in cases when the number of variables is not very large.

In case of computing a two-variable function $AND(x_1, x_2)$, the previously obtained results are as follows:

1. In [36], an algorithm construction method, which allows to obtain a bounded-error quantum algorithm with one query and probability $p = 3/4$ has been demonstrated.
2. In [37] in the proof of Lemma 1, an algorithm for computing an arbitrary two-variable Boolean function has been presented, whose probability is $p = 11/14$. It is also mentioned that the authors can prove that probability $p = 9/10$ is optimal.

3. In [32] in Theorem 3-2, a bounded-error algorithm with one query and probability $p = 5/6$ has been presented. In the same thesis, in Theorem 2-2 approach for constructing an algorithm for computing $AND(f_1, f_2)$ with smaller probability $p = 2/3$ has been demonstrated.

The author presents a new bounded-error algorithm with one query for $AND(x_1, x_2)$, which improves the correct answer probability comparing to 1 and 2 till $p = 4/5$. Comparing to 3, probability is smaller (-1/30), but in exchange for it proposed algorithm has important properties that allow to generalize it for computing conjunction of sub-functions: $AND_2(f_1, f_2)$. Finally, it should be noted that in the author's joint paper with T. Mischenko-Slatenkova another algorithm with one query for $AND(x_1, x_2)$ has been presented, which achieves an optimum (according to [37]) probability $p = 9/10$. This algorithm was designed by T. Mischenko-Slatenkova. Unfortunately, it is not possible to generalize this algorithm for computing $AND_2(f_1, f_2)$.

Within the organizational structure of this section the author discusses, first, the classical complexity of the two-argument Boolean function $AND(x_1, x_2)$. Subsequently, a bounded-error quantum query algorithm is demonstrated computing $AND(x_1, x_2)$ with a probability $p = 4/5$. Finally, the author generalizes the approach applied and presents a method for constructing efficient quantum algorithms for computing a *composite function* $\overline{AND_2}[f_1, f_2]$, where f_1 and f_2 are Boolean functions.

Definition 3.6 ***$\overline{AND_n}$* structure** *($n \in \mathbb{N}$) is a composite Boolean function where arguments are arbitrary Boolean functions f_i and which is defined as*

$$\overline{AND_n}[f_1, f_2, ..., f_n](X) = 1 \quad \Leftrightarrow \quad \sum_{i=1}^{n} f_i(X_i) = n,$$

$X = X_1 X_2 ... X_n$; X_i is input for i^{th} function[3]; f_i's are called base functions.

3.2.1.1 Classical Complexity of $AND(x_1, x_2)$

The classical deterministic complexity of the Boolean function $AND(x_1, x_2)$ is obviously equal to the number of variables: $D(AND_2) = 2$.

Subsequently, the author shows that the best probability for a classical randomized decision tree to compute this function with one query is $p = 2/3$.

Theorem 3.9 *The Boolean function $AND_2(x_1, x_2)$ can be computed by a randomized classical decision tree with one query with the maximum probability p=2/3.*

Proof. The general form of the optimal randomized decision tree is shown in Fig. 3.14.

[3] Variables may also overlap among inputs for different functions, i.e. for $X_i = (x_{i1},...,x_{in})$ and $X_j = (x_{j1},..,x_{jm})$ there may be variables with the same indices.

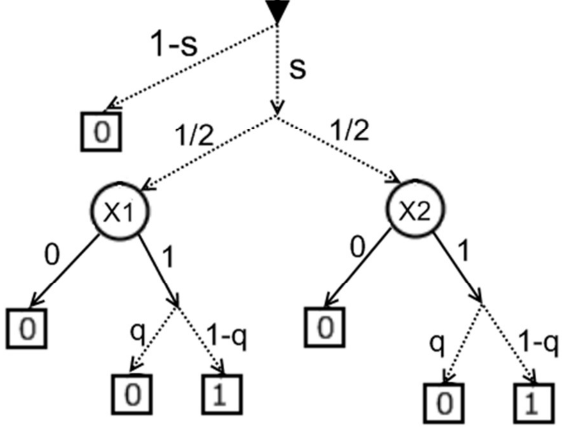


Fig. 3.14 The general form of the optimal classical randomized decision tree for computing $AND_2(x_1,x_2)$

The probability to see the result $b \in \{0,1\}$ after executing the algorithm on input X is denoted by $\Pr("b" \mid X)$. The correct answer probability calculation:

1) $\Pr("0" \mid X = 00) = (1-s) + \frac{1}{2}s + \frac{1}{2}s = 1$,
2) $\Pr("0" \mid X = 01 \ \vee \ X = 10) = (1-s) + \frac{1}{2}s + \frac{1}{2}sq = 1 - \frac{1}{2}(s - sq)$,
3) $\Pr("1" \mid X = 11) = \frac{1}{2}s(1-q) + \frac{1}{2}s(1-q) = s - sq$.

Let us denote $(s - sq) = z$. Then, the total probability of the correct answer is

$$p = min(\Pr("0"), \Pr("1")) = min(1 - \frac{1}{2}z, z).$$

The highest probability is obtained when $\Pr("0") = \Pr("1")$.

$$1 - \frac{1}{2}z = z$$
$$z = \frac{2}{3}$$

3.2.1.2 Quantum Query Algorithm for $AND(x_1, x_2)$

The author starts with a bounded-error quantum query algorithm for the simplest case of a two-variable function $AND(x_1, x_2)$.

Theorem 3.10 *There exists a quantum query algorithm Q1 computing the Boolean function $AND(x_1, x_2)$ with one quantum query and correct answer probability p=4/5:*

$$Q_{4/5}(AND_2) = 1.$$

Proof. The algorithm is presented in Fig. 3.15. The algorithm uses 3-qubit quantum system. Each horizontal line corresponds to the amplitude of the basis state. Computation starts with the state $|\varphi\rangle = \left(\frac{2}{\sqrt{5}},\ 0,\ 0,\ 0,\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$ (unitary transformation, converting the initial state $|\vec{0}\rangle = (1,0,0,..,0)^T$ into $|\varphi\rangle$, is omitted). Two large rectangles correspond to the 8×8 unitary matrices U_0 and U_1. The vertical layer of circles specifies the queried variable order for the single query Q_0. Finally, eight small squares at the end of each horizontal line define the assigned function value for each basis state. The main idea is to assign the amplitude value $\alpha = 1/\sqrt{5}$ to the basis state $|100\rangle$ and leave it invariable until the end of the execution.

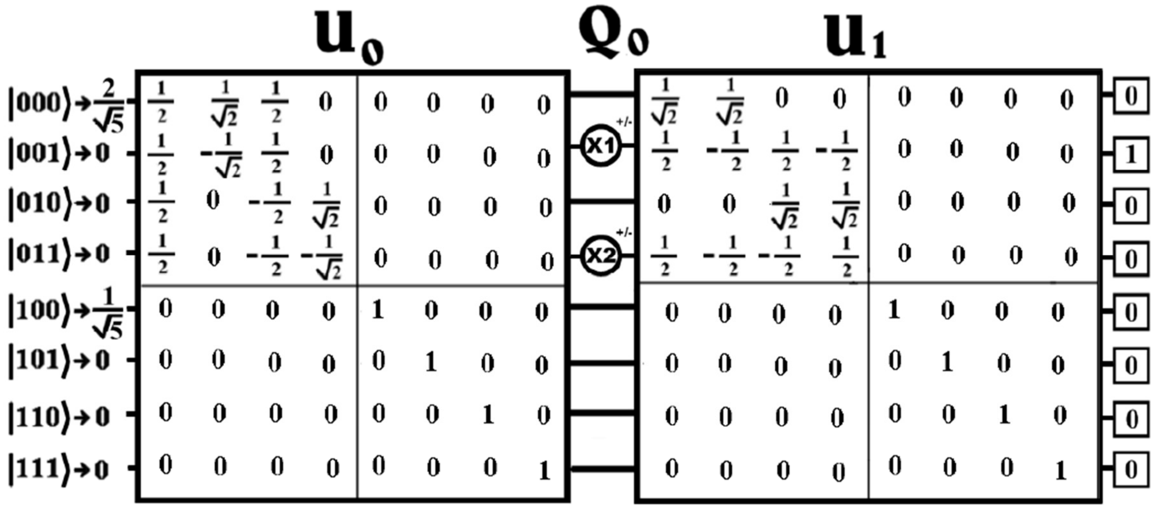


Fig. 3.15 Bounded-error quantum query algorithm $Q1$ for computing $AND(x_1, x_2)$

Quantum state after the first transformation U_0 becomes equal to

$$|\varphi_2\rangle = U_0|\varphi\rangle = U_0 \cdot \left(\frac{2}{\sqrt{5}},\ 0,\ 0,\ 0,\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T =$$

$$= \left(\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$$

Further evolution of the quantum system for each input X is shown in Table 3.6.

Table 3.6 Quantum query algorithm $Q1$ computation process for $AND(x_1, x_2)$

X	$\lvert\varphi_3\rangle = Q_0 U_0 \lvert\varphi\rangle$	$\lvert\varphi_{FINAL}\rangle = U_1 Q_0 U_0 \lvert\varphi\rangle$	p("1")
00	$\left(\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	$\left(\sqrt{\frac{2}{5}}, 0, \sqrt{\frac{2}{5}}, 0, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	0
01	$\left(\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, -\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	$\left(\sqrt{\frac{2}{5}}, \frac{1}{\sqrt{5}}, 0, -\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	$\frac{1}{5}$
10	$\left(\frac{1}{\sqrt{5}}, -\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	$\left(0, \frac{1}{\sqrt{5}}, \sqrt{\frac{2}{5}}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	$\frac{1}{5}$
11	$\left(\frac{1}{\sqrt{5}}, -\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, -\frac{1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	$\left(0, \frac{2}{\sqrt{5}}, 0, 0, \frac{1}{\sqrt{5}}, 0, 0, 0\right)^T$	$\frac{4}{5}$

3.2.1.3 Decomposing the $AND(x_1, x_2)$ Algorithm

This section is a transitional point on the way to achieve a generalized method for computing the structure $\overline{AND_2}$. Now, the author reveals the internal details of the algorithm *Q1* allowing adaptation of its structure to compute a much wider set of Boolean functions.

The fairly chaotic and asymmetric matrix U_0 actually is a product of two other matrices.

$$U_0 = U_0^B \cdot U_0^A = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \bullet \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix U_1, in turn, is a product of the following two matrices.

$$U_1 = U_1^B \cdot U_1^A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \bullet \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

A detailed algorithm structure now looks as follows:

$$|\varphi\rangle \to U_0^A \ , \ U_0^B \ , \ Q_0 \ , \ U_1^A \ , \ U_1^B \to [Measure]$$

The final vector is calculated as

$$|\varphi_{FINAL}\rangle = U_1^B \cdot U_1^A \cdot Q_0 \cdot U_0^B \cdot U_0^A \cdot |\varphi\rangle .$$

The most important point – the algorithm part represented by transformations U_0^B , Q_0 , U_1^A - actually executes two instances of an exact quantum query algorithm for $f(x) = x$ in parallel. The above significant detail is graphically demonstrated in Fig. 3.16 and Fig. 3.17.

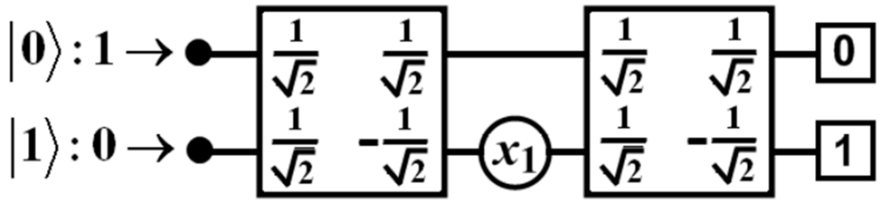


Fig. 3.16. Exact quantum query algorithm for computing $f(x) = x$

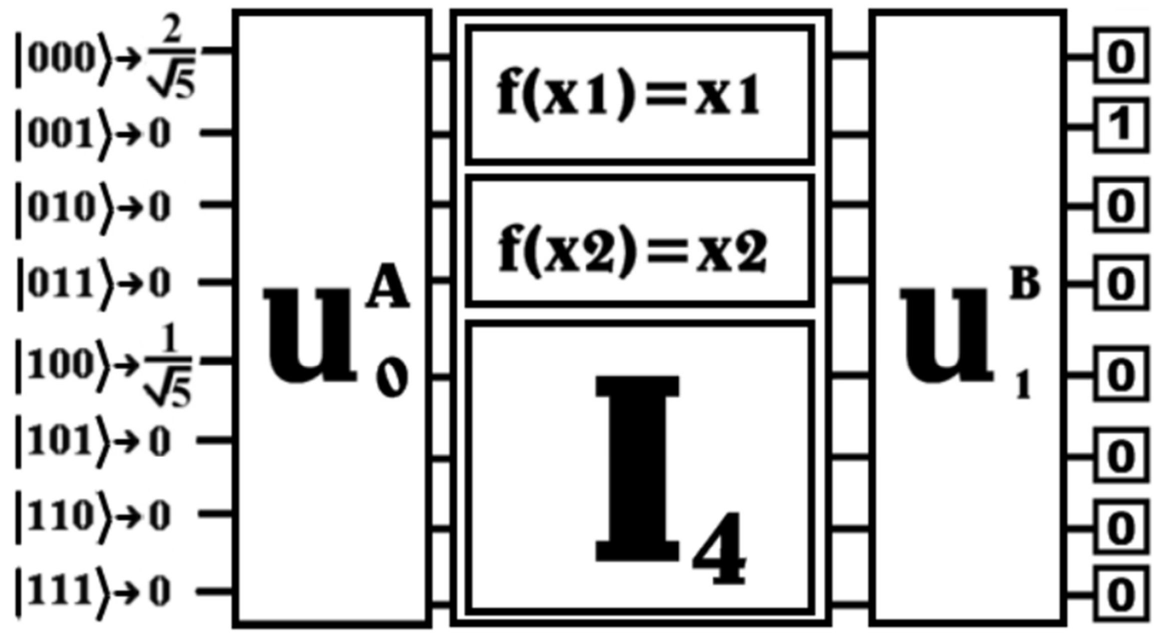


Fig. 3.17. Quantum algorithm for $AND(x_1, x_2)$, revised

In other words, firstly, quantum parallelism is employed to evaluate each variable. Secondly, the unitary transformation U_1^B is applied to correlate the amplitude distribution in such a way that the resulting quantum algorithm computes $AND(x_1, x_2)$ with acceptable error probability.

In the next section, this approach is generalized to allow the use of other Boolean functions as sub-routines.

3.2.1.4 A Method for Computing $\overline{AND}_2[f_1, f_2]$

It is possible to replace a sub-algorithm for $f(x) = x$ (in an algorithm construction demonstrated in the previous section) with any other quantum algorithm meeting specific properties. The author's method is applicable to base algorithms belonging to *Class* 2+.

Theorem 3.11 *Given exact quantum query algorithms A1 and A2 for computing Boolean functions $f_1(X_1)$ and $f_2(X_2)$ that belong to Class 2+, a composite Boolean function $\overline{AND}_2[f_1, f_2]$ can be computed with probability p=4/5 using $max(Q_E(A1), Q_E(A2))$ queries to the black box.*

Proof. A general algorithm construction method for computing the Boolean function $\overline{AND}_2[f_1, f_2]$ is presented in the next table below.

A method for computing $\overline{AND_2}[f_1, f_2]$

Input. Two exact quantum query algorithms $A1, A2 \in \mathit{Class}\ 2+$ that compute Boolean functions $f_1(X_1), f_2(X_2)$. The dimension of the Hilbert space utilized by the first algorithm is denoted by m_1 (the number of amplitudes), and for the second algorithm by m_2. The positions of accepting outputs of A1 and A2 are denoted with acc_1 and acc_2.

Constructing steps

1. If $m_1 = m_2$, then utilize a quantum system with $4m_1$ amplitudes for a new algorithm. First $2m_1$ amplitudes will be used for the parallel execution of A1 and A2. Additional qubit is required to provide separate amplitude for storing the value of $1/\sqrt{5}$.
2. If $m_1 \neq m_2$ (without loss of generality assume that $m_1 > m_2$), then utilize a quantum system with $2m_1$ amplitudes for a new algorithm. First $(m_1 + m_2)$ amplitudes will be used for the parallel execution of A1 and A2. Use the first remaining free amplitude for storing the value of $1/\sqrt{5}$.
3. Start the computation from the state

$$|\varphi\rangle = \left(\underbrace{\sqrt{2/5},\ 0,...,0}_{m_1}, \underbrace{\sqrt{2/5},\ 0,...,0}_{m_2}, \underbrace{1/\sqrt{5},\ 0,..,0}_{remaining\ amplitudes} \right)^T.$$

4. Combine unitary transformations and queries of A1 and A2 in the following way:

$$U_i = \begin{pmatrix} U_1 & O_{m_1\times m_2} & O_{m_1\times m_1} \\ O_{m_2\times m_1} & U_2 & O_{m_2\times m_1} \\ O_{m_1\times m_1} & O_{m_1\times m_2} & I_{m_1-m_2} \end{pmatrix}$$, here $O_{m_i\times m_j}$ are $m_i \times m_j$ zero-matrices,

$I_{m_1-m_2}$ is $(m_1-m_2)\times(m_1-m_2)$ identity matrix, U_i^1 and U_i^2 are either fixed or query unitary transformations of A1 and A2.

5. Apply transformations U_i. Before the final measurement apply an additional unitary transformation:

$$U = (u_{ij}) = \begin{cases} 1, & \text{if } (i=j)\,\&\,(i \neq acc_1)\,\&\,(i \neq (m_1 + acc_2)) \\ 1/\sqrt{2}, & \text{if } (i = j = acc_1) \\ 1/\sqrt{2}, & \text{if } (i = acc_1)\,\&\,(j = (m_1 + acc_2)) \text{ OR } (i = (m_1 + acc_2))\,\&\,(j = acc_1) \\ -1/\sqrt{2}, & \text{if } (i = j = (m_1 + acc_2)) \\ 0, & \text{otherwise} \end{cases}$$

6. Define as accepting output exactly one basis state $|acc_1\rangle$.

Output. A bounded-error quantum query algorithm A for computing a function $F(X) = f_1(X_1) \wedge f_2(X_2)$ with a probability $p = 4/5$ and complexity $Q_{4/5}(A) = \max(Q_E(A1), Q_E(A2))$.

The most significant advantage of this method is that the overall algorithm complexity does not exceed the greatest complexity of sub-algorithms. To compute a composite function, additional queries are not required. However, the cost for efficient computing is the error probability.

A very important aspect is that author has used a specific algorithm for the two-variable Boolean function $AND(x_1, x_2)$ as a base for the constructing method. If the correct answer probability for the $AND(x_1, x_2)$ algorithm, which would also use an algorithm for computing *f*(*X*)=*X* as a sub-routine, is improved to $p > 4/5$, then the probability of a general constructing method and all further results of this section would be improved as well.

According to Theorem 3.1 it is possible to place into $\overline{AND_2}$ structure any Boolean function having exactly one accepting input vector. Thus, the method is applicable to an infinite set of base functions.

Theorem 3.12. *For an infinite set of Boolean functions, quantum query algorithms can be constructed using a method for computing* $\overline{AND}2[f_1, f_2]$. *As a result, the following complexity gap can be achieved when computing the same function in quantum and classical deterministic models:* $Q_{4/5}(\overline{AND}2(f_1, f_2)) = \frac{1}{2} \cdot D(\overline{AND}2(f_1, f_2))$.

Proof For any Boolean function *f* having a single accepting vector, the sensitivity *s*(*f*) and, consequently, the deterministic complexity *D*(*f*) are equal to the number of variables. Let us suppose there are two Boolean functions f_1 and f_2, with the same number of variables *N*, and the task is to compute $\overline{AND_2}(f_1, f_2)$[4]. Obviously, the classical deterministic complexity of this function is $D(\overline{AND}2(f_1, f_2)) = 2N$. For each function a deterministic algorithm can be converted into an exact quantum query algorithm of the class *Class*2+, using the same *N* queries. Finally, the method is applied for constructing an algorithm for $\overline{AND_2}(f_1, f_2)$ which does not require additional queries:

$$Q_{4/5}(\overline{AND}2(f_1, f_2)) = N = \frac{1}{2} \cdot D(\overline{AND}2(f_1, f_2)).$$

□

In the theorem above, the classical deterministic and the quantum bounded-error query complexity is compared. It would be interesting to compare classical probabilistic and quantum bounded-error complexity for $\overline{AND_2}(f_1, f_2)$. As of today, the author does not have such estimation.

[4] It is assumed that variables do not overlap this time.

Theorem 3.13. *The Boolean function $AND_N(X)$ ($N = 2k,\ k \in \mathbb{N}$) can be computed by a bounded-error quantum query algorithm with a probability p = 4/5 using N/2 queries:*

$$Q_{4/5}(AND_N) = N/2.$$

Proof. Boolean function $AND_N(X)$ can be represented as

$$AND_N = \overline{AND_2}(AND_{N/2}, AND_{N/2}).$$

It means that by applying the construction method it is possible to obtain an algorithm with complexity $Q_{4/5}(AND_N) = Q_E(AND_{N/2})$.

The Boolean function $AND_{N/2}$ can be computed by a deterministic algorithm with complexity $D(AND_{N/2}) = N/2$, which has exactly one accepting output. It means that this deterministic algorithm can be converted into *Class* 2+ algorithm using the same *N*/2 number of queries.

$$Q_{4/5}(AND_N) = Q_E(AND_{N/2}) = N/2.$$

□

3.2.1.5 An Example of a Larger Separation: *D*(*f*)=6 versus $Q_{4/5}(f)$=2

This subsection demonstrates an example where the quantum algorithm complexity is more than twice lower than the classical deterministic algorithm complexity. It is possible in cases when an exact quantum algorithm for a sub-function is better than the best possible deterministic algorithm for the same function.

An exact quantum query algorithm for $EQUALITY_3(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)$ has been presented in Section 3.1.1.1. The algorithm uses only two quantum queries while classically all three queries are required. The algorithm belongs to class *Class*2+ and can be used as a sub-algorithm for $\overline{AND_2}$ structure.

To evaluate the deterministic complexity of $f = \overline{AND_2}[EQUALITY_3, EQUALITY_3]$, function sensitivity on any accepting input is used: $s(f) = 6 \Rightarrow D(f) = 6$.

A quantum bounded-error algorithm for $f = \overline{AND_2}[EQUALITY_3]$ constructed using the author's method requires only two queries: $Q_{4/5}(f) = 2$.

The same approach can be applied to any algorithm of *Class*2+ computing *N*-variable Boolean function.

3.2.1.6 Iterative Application of a Method for Computing $\overline{AND_2}[f_1, f_2]$

The useful properties of the algorithm construction method described in Section 3.2.1.4 enable a repeated application of this method.

Theorem 3.14. *Let $F_1 = AND_2[f_{11}, f_{12}]$ and $F_2 = AND_2[f_{21}, f_{22}]$ be composite Boolean functions. Let Q1 and Q2 be bounded-error quantum query algorithms that have been constructed using a method for computing $AND_2[f_1,f_2]$, and that compute F_1 and F_2 with probability p=*4/5*. Then, a bounded-error quantum query algorithm Q can be constructed to compute a composite Boolean function $F = AND_2[F_1, F_2]$ with probability p* = 16/25.

Proof. The method for computing $AND_2[f_1,f_2]$ is straightforwardly applied to algorithms $Q1$ and $Q2$ instead of instances of *Class* 2+. As a result, the obtained complex algorithm computes $F = AND_2[F_1, F_2]$ with a probability $p = \frac{4}{5} \cdot \frac{4}{5} = \frac{16}{25}$. □

Consequently, it is possible to compute a four-variable function $AND(x_1,\ldots,x_4)$ with a single quantum query with a probability $p = 16/25$.

The next iteration produces quantum algorithms computing functions in a form

$$F = AND_2[AND_2[AND_2[f_1, f_2], AND_2[f_3, f_4]], AND_2[AND_2[f_5, f_6], AND_2[f_7, f_8]]]$$

with probability $p = 64/125$, which is just slightly more than a half.

3.2.1.7 Enlarging a Set of Method Input Algorithms

In the description of a general method for computing $AND_2[f_1,f_2]$ a set of input algorithms is restricted to instances of *Class* 2+. That was done for simplicity of description of the main idea and details. However, it is possible to apply the proposed method to quantum algorithms outside *Class* 2+ as well.

Let us consider the following case. Let A1 and A2 be two bounded-error quantum algorithms that compute Boolean functions $f_1(X_1)$ and $f_2(X_2)$, each having exactly one accepting output. It is possible to apply the general method also to these algorithms. Combined algorithm computes a composite function $f(X) = f_1(X_1) \wedge f_2(X_2)$, where $X = X_1X_2$. Suppose that the combined algorithm is executed on input X, which is accepting for both sub-algorithms. Let us assume that algorithm A1 executed on corresponding input X_1 finishes with amplitude α_1 in its accepting state. A2 executed on corresponding input X_2 finishes with amplitude α_2 in its accepting state. Additional condition is that signs of α_1 and α_2 must be the same. Then, combined algorithm will have the following amplitude value in its accepting state:

$$\alpha = \frac{1}{\sqrt{2}}\sqrt{\frac{2}{5}}\alpha_1 + \frac{1}{\sqrt{2}}\sqrt{\frac{2}{5}}\alpha_2 = \frac{1}{\sqrt{5}}(\alpha_1 + \alpha_2).$$

Thus, probability to obtain result $f(X) = 1$ is:

$$p = |\alpha|^2 = \frac{1}{\sqrt{5}}|\alpha_1 + \alpha_2|^2 = \frac{1}{5}|\alpha_1 + \alpha_2|^2.$$

Moreover, it is also possible to apply general method for computing $AND_2[f_1,f_2]$ to basic algorithms with more than one accepting output. In such a case it is necessary to execute more instances of basic algorithms in parallel and join outputs between each other. If the first basic algorithm has N accepting outputs and second basic algorithm has M accepting outputs, then $N \cdot M$ parallel executions of paired algorithm instances are necessary.

3.2.1.8 Improved Quantum Query Algorithm for $AND(x_1, x_2)$

Lastly, the author is going to demonstrate another bounded-error quantum query algorithm for computing $AND_2(x_1,x_2)$ that achieves a higher (and optimum according to

[37]) correct answer probability $p = 9/10$. This algorithm designed by the co-author T. Mischenko-Slatenkova has been published in the joint paper of both authors. Unfortunately, the algorithm cannot be directly applied in the method for constructing an algorithm for the $AND_2[f_1,f_2]$ structure.

Theorem 3.15. *There exists a quantum query algorithm Q2 that computes Boolean function $AND_2(x_1,x_2)$ with one quantum query and the correct answer probability $p=9/10$: $Q_{9/10}(Q2)=1$.*

Proof. Algorithm is presented in Fig. 3.18. Computation process for each input X is shown in Table 3.7. □

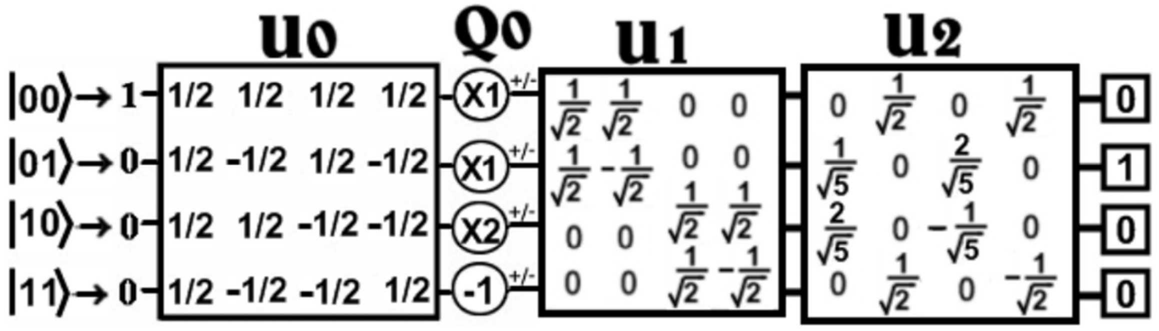


Fig. 3.18 Bounded-error quantum query algorithm *Q2* for computing $AND_2(x_1,x_2)$

The structure of the algorithm $Q2$ is different from the structure of the algorithm $Q1$ presented in Section 3.2.1.2. The fundamental difference lies in the query behavior and unfortunately this difference does not allow applying the algorithm to compute conjunction of the sub-functions.

On the other hand, the correct answer probability obtained by the algorithm $Q2$ is higher than that of $Q1$. It is closer to $p = 1$, so, in some applications it may be more advantageous to use this algorithm instead of the previous one. Algorithm $Q2$ also uses less qubits than algorithm $Q1$, and that may be considered an advantage as well.

Table 3.7. Computation process of the quantum algorithm *Q*2 for computing $AND_2(x_1,x_2)$

X	$\lvert\varphi_2\rangle = Q_0 U_0 \lvert\varphi_0\rangle$	$\lvert\varphi_2\rangle = U_2 U_1 Q_0 U_0 \lvert\varphi_0\rangle$	p("1")
00	$\frac{1}{2}(1,\ 1,\ 1,\ -1)^T$	$\left(\frac{1}{2},\ \frac{1}{\sqrt{10}},\ \sqrt{\frac{2}{5}},\ -\frac{1}{2}\right)^T$	$\frac{1}{10}$
01	$\frac{1}{2}(1,\ 1,\ -1,\ -1)^T$	$\left(0,\ -\frac{1}{\sqrt{10}},\ \frac{3}{\sqrt{10}},\ 0\right)^T$	$\frac{1}{10}$
10	$\frac{1}{2}(-1,\ -1,\ 1,\ -1)^T$	$\left(\frac{1}{2},\ -\frac{1}{\sqrt{10}},\ -\sqrt{\frac{2}{5}},\ \frac{1}{2}\right)^T$	$\frac{1}{10}$
11	$\frac{1}{2}(-1,\ -1,\ -1,\ -1)^T$	$\left(0,\ -\frac{3}{\sqrt{10}},\ -\frac{1}{\sqrt{10}},\ 0\right)^T$	$\frac{9}{10}$

3.2.2 Conclusion and Open Problems

In this section, the computing of conjunctions in the quantum bounded-error settings was examined. A quantum query algorithm was presented computing the conjunction of two bits by making only one query with the correct answer probability $p = 4/5$. Subsequently, the approach was extended and a general method was formulated for computing a conjunction of two Boolean functions with the same probability and a number of queries equal to $\max(Q_E(f_1), Q_E(f_2))$. The proposed approach provides for designing of the quantum algorithms for complex functions based on the already known algorithms. A significant advantage lies in the fact that the overall algorithm complexity does not increase; additional queries are not required to compute a composite function.

The proposed quantum algorithms are more efficient than the best possible classical deterministic or quantum exact algorithms and they provide higher accuracy than the best possible classical randomized decision trees.

Regarding computing conjunctions the author would like to extend the number of clauses from two to *N*. The author would also like to improve the probability of iterative application of the construction method. Another fundamental goal is to develop a framework for building efficient ad-hoc quantum query algorithms for arbitrary Boolean functions.

4 Quantum Query Algorithms for Multivalued Functions

This chapter is based on the papers

- A. Vasilieva. Quantum Query Algorithms for Relations. *Proc. of the MFCS & CSL 2010 Satellite Workshop Randomized and quantum computation*, ISBN 978-80-87342-08-4, pp. 78-89, 2010
- A. Vasilieva. Uniformly Distributed Quantum Query Algorithms for Multifunctions. *Proc. of the Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)* , pp. 86-93, 2011

Most often query model is used to compute Boolean functions. However, it is possible to apply query model to functions with larger domain and range as well. In this section, the author considers even more uncommon case – computing of *multivalued functions*. A multivalued function is more general type of problem than a usual function. Multifunction is a left-total binary relation that associates values from a domain set with one or more values from a range set. Function is simply a special case of a multifunction, where each value from a domain set is mapped to no more than one value from a range set.

The study of query complexity of multifunctions has been inspired by the book on communication complexity by Kushilevitz and Nisan [38]. The main part of this book discusses communication complexity of functions, but Chapter 5 is devoted to the communication complexity of relations.

The author applies traditional query model to compute multifunctions. In classical deterministic settings, however, it does not seem to be possible to employ the difference between a multifunction and a function to obtain new interesting results. A deterministic decision tree always follows one and the same fixed path for each certain input and outputs one and the same value each time. The situation is different in the quantum case. Quantum state before the measurement is in a superposition of the basis states, so it is not determined to which exactly basis state quantum system collapses after the measurement.

Various computational problems may be represented in terms of multifunctions. Let us consider, for instance, an online reservation system for a large renting company. Company provides various products for rent, for example, cars, flats, TV-sets etc. User fills in a reservation form on the Web page and submits it. According to user's request parameters (multifunction input) system has to find a set of satisfying and available items (value set for that input) and display them to user for further selection or even perform selection automatically. By designing an efficient algorithm for computing this kind of multifunction it is possible to speed-up processing significantly. Nowadays, in heavy-loaded systems with huge amount of concurrent requests, a lot of resources could be saved by performance improvement at the moment of selecting appropriate value set.

The greatest challenge in designing quantum query algorithm for a function is ensuring that the algorithm is exact (i.e. making it output correct result always with probability $p = 1$). The largest complexity separation between classical deterministic and quantum exact query algorithm complexity for the same total function known for today is N versus $N/2$ [12]. However, in the case of multifunction it is allowed to output values from a fixed set instead of one fixed value for a certain input. The author asserts that in such case the task of designing a non-trivial exact quantum query algorithm is achievable more easily. That could help to construct examples, where number of queries required by quantum algorithm is more than two times less than required by classical algorithm.

In this section, the query model is adapted for computing multifunctions. First, the author gives definitions related to multifunctions. Then, the author defines several types of query algorithms that compute multifunctions in different manners. Finally, examples of computing multifunctions in classical and quantum query models are demonstrated, where quantum algorithm achieves a speed-up comparing to classical algorithm.

4.1 Multivalued Functions

The main object studied in this section is a multivalued function.

A multivalued function (*multifunction*, other names: relation, set-valued function, multi-valued map, correspondence) is more general type of problem than a usual function. Multifunction is a left-total binary relation that associates values from a domain set with one or more values from a range set. So, the difference from a usual function is in element mapping: each element from a domain set may be mapped to multiple elements from a range set. Function (single-valued function) is simply a special case of a multifunction, where each value from a domain set is mapped to no more than one value from a range set. The author considers the following kind of multifunctions:

$$M(X):\{0,1\}^N \to \mathbb{N}, \text{ where } X=(x_1,x_2,...,x_N),\ x_i \in \{0,1\}.$$

The notation is the following:

- each value from the domain set is called – an *input X;*
- each x_i is called – a *variable*;
- a set of associated values from the range set is called – a *result set for input X* - and is denoted by $M(X)$.

A function is a special case of multifunction and it uniquely associates each value from the domain set with one value from the range set. Fig. 4.1 graphically demonstrates this difference.

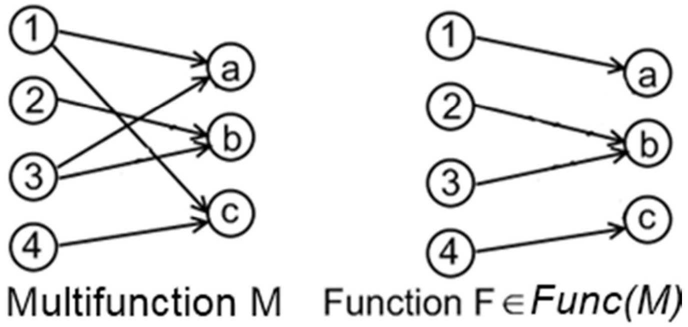


Fig. 4.1 Example of a multifunction and a function

Various functions can be selected in such a way from a single multifunction. Let us denote the set of all total functions that can be selected from multifunction M by $Func(M)$.

Example. The graph on the left side of Fig. 4.1 defines the multifunction:

$$M = \{ (1,a),(1,c),(2,b),(3,a),(3,b),(4,c) \}.$$

The set *Func*(*M*) consists of four total functions that may be selected as a subset of the multifunction:

$$Func(M) = \{ f_1 = \{ (1,a),(2,b),(3,a),(4,c) \}, f_2 = \{ (1,a),(2,b),(3,b),(4,c) \}, \\ f_3 = \{ (1,c),(2,b),(3,a),(4,c) \}, f_4 = \{ (1,c),(2,b),(3,b),(4,c) \} \}.$$

4.2 Computing Multifunctions in a Query Model

Computation of usual functions in a query model has been studied in detail: for each input, algorithm has to output correct function value with a certain probability. However, it is not obvious how to extend a query model to compute multivalued functions. The author proposes three different options to describe that a query algorithm computes a multifunction and defines three types of query algorithms based on these options. The author considers total multifunction only, where result set is not empty for every input *X*.

Definition 4.1. *Query algorithm computes multifunction M(X) in a* ***definite manner****, if for each X it outputs one certain correct value from a result set with probability p = 1. Classical query complexity is denoted by* $C_D(M)$*. Quantum query complexity is denoted by* $Q_D(M)$*.*

The type of classical decision tree that computes a multifunction in a definite manner is deterministic decision tree. In the quantum version, corresponding algorithm type is an exact quantum query algorithm.

Definition 4.2. *Query algorithm computes multifunction M(X) in a* ***randomly distributed manner****, if for each X it outputs arbitrary values from a result set with arbitrary probabilities (for each value such probability has to be positive) and never outputs an incorrect value. Classical query complexity is denoted by* $C_{RD}(M)$*. Quantum query complexity is denoted by* $Q_{RD}(M)$*.*

This definition is more natural and takes into account the essence of multifunction as a mathematical object. In a classical query model, probabilistic decision trees should be used to produce the described behavior. Quantum query algorithms seem to be better suited for computing multifunctions in a distributed manner because of the superposition principle. To make algorithm behavior correct on certain input it is necessary to prepare quantum system in a superposition, where only basis states associated with values from the result set have non-zero amplitude values. After the measurement, quantum system collapses to one of these basis states with a probability determined by its amplitude value.

Definition 4.3. *Query algorithm computes multifunction M(X) in a* ***uniformly distributed manner****, if for each X it outputs each value from a result set with equal probability and never outputs an incorrect value. Classical query complexity is denoted by* $C_{UD}(M)$*. Quantum query complexity is denoted by* $Q_{UD}(M)$*.*

This definition adds a serious constraint to design of a query algorithm. This time there is an important restriction – every value from the result set has to have equal probability to be produced by query algorithm on output. However, in author's opinion this definition is the most reasonable in a sense of computing a multifunction and algorithms of that type have most practical applications.

Each definition may be applied for solving specific real-world computational problems. Author is most interested in comparing complexity of computing multifunctions in the same manners in classical and quantum query models. The goal is to analyze algorithm implementation special features and differences to produce examples with large difference between classical and quantum query complexity.

4.2.1 A Note on Computing Multifunctions in a Definite Manner

First, the first type of query algorithms for multifunctions is discussed, which compute multifunctions in a definite manner. Are there prospects to obtain a large separation between classical and quantum complexity?

According to the definition, for each input X such algorithm always outputs one definite value. The only condition is that this value should be from the result set assigned to that input by multifunction M. It actually means that a definite query algorithm for multifunction M computes a function, which is a subset of multifunction.

When designing a query algorithm to compute multifunction M in a definite manner, we may choose some arbitrary function from a set $Func(M)$, which is better suited for computing in a query model, and construct an algorithm for that function. So, classical and quantum query complexities for computing multifunction definitely are expressed by formulas:

$$D(M) = \min_{f \in Func(M)} (D(f)) \qquad Q_E(M) = \min_{f \in Func(M)} (Q_E(f))$$

It appears that the task of enlarging the gap between classical and quantum query complexity to compute multifunctions in a definite manner is completely the same as when computing usual functions in a query model. Even more, the interesting moment is that the functions selected from the set $Func(M)$ for computing in classical and quantum cases may also be different. Unfortunately, it does not give us additional tool to enlarge the complexity gap when computing multifunction instead of function, quite contrary. For that reason, computing multifunctions in a distributed manner looks much more interesting.

4.3 First Example of Computing a Multifunction

In this subsection, the author illustrates the difference of computing multifunction in definite and distributed manners. Also a gap of N versus $2N$ is demonstrated between quantum and classical uniformly distributed query complexity of a multifunction.

Let us consider an online banking client service system. To receive specific kind of bank's services, client sends a request to the system. System has to analyze client's request, determine a set of appropriate agents and assign a request to some agent from this set.

In this example, four agents are assumed: Alice (id = 1), Bob (id = 2), Carol (id = 3) and Daren (id = 4). There are three factors that determine a set of appropriate agents for each client – location, client status and loan history. Table 4.1 describes these parameters. Second column contains a reference to the system function that has to be invoked to calculate parameter value. Invocation of each function can be interpreted as querying a black box and internal calculations may involve various database requests and other costly operations. Third column contains possible parameter values; fourth column contains corresponding numeric value returned by each function.

Table 4.1. Parameters that determine an agent that is able to serve client's request

Parameter		**Value**	
Description	**System function**	**Actual**	**Numeric**
Client location	`getLocation(client_id)`	Saldus	0
		Ventspils	1
Client status	`isVIP(client_id)`	Normal	0
		VIP	1
Does client have an active loan?	`hasLoan(client_id)`	No	0
		Yes	1

Table 4.2 defines the three-variable multifunction with Boolean domain and four-valued range: $M_1 : \{0,1\}^3 \to \{1,2,3,4\}$.

Rows of Table 4.2 have to be interpreted as the following statements:

- If a request is received from an ordinary client from Saldus, which does not have an active loan (X = 000), then a request should be served by either Alice or Carol;
- If a request is received from an ordinary client from Saldus, which has an active loan (X = 001), then a request should be served by either Alice or Daren;
- If a request is received from a VIP client from Saldus, which does not have an active loan (X = 010), then a request should be served by either Bob or Daren;
- etc.

Table 4.2. Definition of the multifunction M_1

X	$M_1(X)$	X	$M_1(X)$
000	{ 1 , 3 }	100	{ 2 , 4 }
001	{ 1 , 4 }	101	{ 2 , 3 }
010	{ 2 , 3 }	110	{ 1 , 4 }
011	{ 2 , 4 }	111	{ 1 , 3 }

Now, let us discuss the computational complexity of the multifunction M_1.

4.3.1 Definite Query Complexity of the Multifunction M_1

When computing a multifunction in a definite manner, algorithm has to output one arbitrary correct value from a result set with probability p = 1. It means that it is necessary just to ensure that client's request is not forwarded to incompetent agent, while the work distribution among the competent agents is not handled.

Theorem 4.1. $C_D(M_1) = 2$.

Proof. It is easy to see that one query is not enough to compute this multifunction classically in a definite manner. However, two queries are sufficient to reach the goal – it is necessary to know the values of the first two variables only. Deterministic decision tree is shown in Fig. 4.2.

Actually, the task in this case it to compute *XOR* of the first two bits. If $XOR(x_1, x_2) = 0$, algorithm outputs “1”. Otherwise, algorithm outputs “2”. □

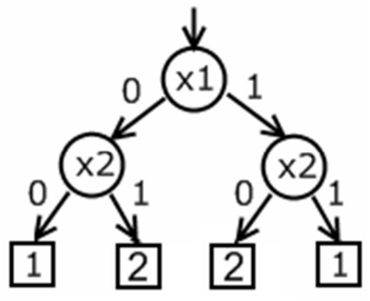


Fig. 4.2 Deterministic decision tree that computes M_1 in a definite manner

Theorem 4.2. $Q_D(M_1) = 1$.

Proof. It is well known that *XOR* of two bits can be computed exactly in the quantum query model by asking one query (see detailed description in Section 2.4). It immediately implies that multifunction M_1 can be computed with one query in a quantum query model in a definite manner. Quantum algorithm adapted for computing M_1 is shown in Fig. 4.3. □

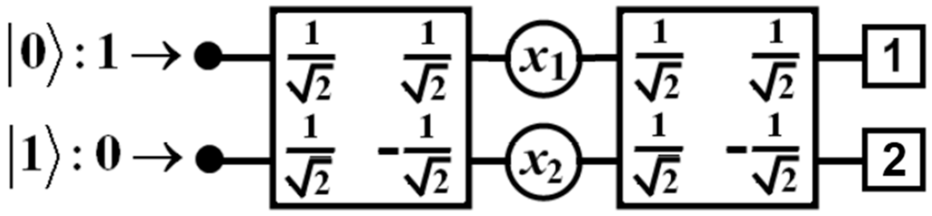

$|0\rangle : 1 \rightarrow$
$|1\rangle : 0 \rightarrow$
$\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$
$\frac{1}{\sqrt{2}}$ $-\frac{1}{\sqrt{2}}$
x_1
x_2
$\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$
$\frac{1}{\sqrt{2}}$ $-\frac{1}{\sqrt{2}}$
1
2

Fig. 4.3 Quantum query algorithm that computes M_1 in a definite manner

Although the algorithm is correct, the problem with such implementation of work distribution algorithm is that all requests are forwarded to Alice and Bob only, but Carol and Daren are bored without work.

4.3.2 Uniformly Distributed Query Complexity of the Multifunction M_1

Now, let us consider computing M_1 in a uniformly distributed manner, which seems to be much more practical in the context of this task. This time algorithm has to output each value from the result set with equal probability and should never output incorrect value.

Obviously, one query is not enough in the classical case. However, this time again, two queries suffice.

Theorem 4.3. $C_{UD}(M_1) = 2$.

Proof. Classical probabilistic decision tree that computes M_1 in a uniformly distributed manner is shown in Fig. 4.4. □

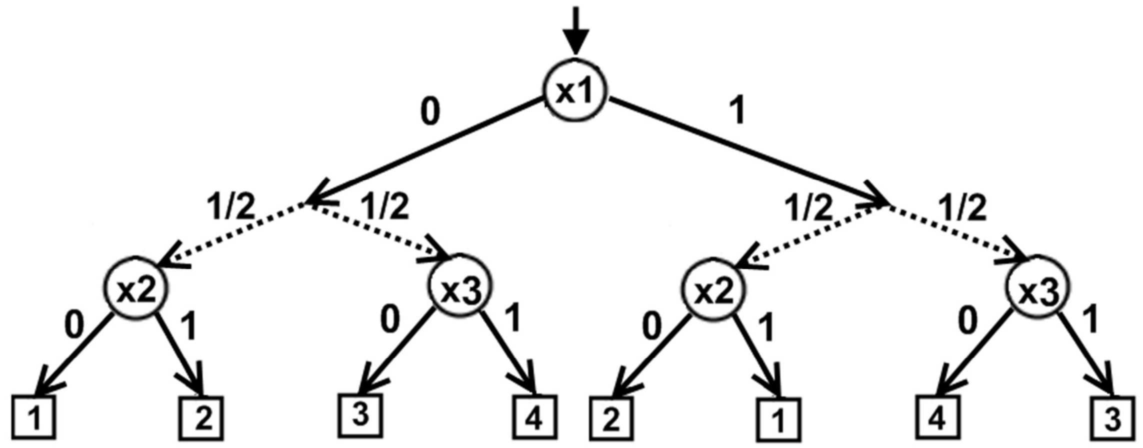


Fig. 4.4 Probabilistic decision tree that computes M_1 in a uniformly distributed manner

Theorem 4.4. $Q_{UD}(M_1) = 1$.

Proof. Quantum query algorithm QM1, which computes M_1 in the same uniformly distributed manner with one query, is presented in Fig. 4.5 and described below.

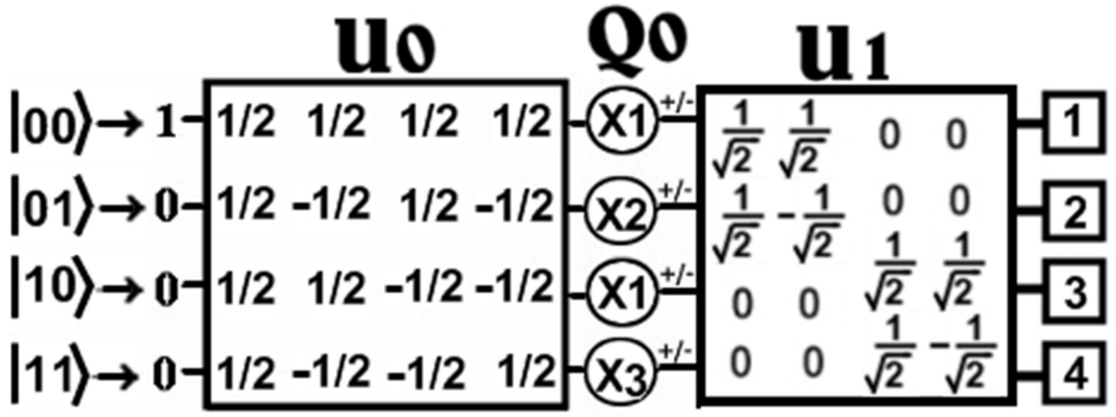


Fig. 4.5 Quantum query algorithm QM1 for computing M_1 in a uniformly distributed manner

Algorithm QM1 uses two-qubit quantum system. Single query Q_0 is defined by the unitary matrix:

$$Q_0 = \begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 \\ 0 & 0 & (-1)^{x_1} & 0 \\ 0 & 0 & 0 & (-1)^{x_3} \end{pmatrix}$$

Computational process for each input X is presented in a table in Appendix 4. □

This time all work items are equally distributed among agents.

With this basic example the author demonstrated query algorithms for computing multifunction in action. It is demonstrated that even in such a simple case of multifunction with three Boolean variables it is possible to obtain a gap between classical and quantum query complexity. In the next subsection, the author demonstrates a way for enlarging complexity gap in a uniformly distributed case.

4.3.3 Generalizations of the Multifunction M_1

In this subsection, the author demonstrates two generalizations of the multifunction M_1 with bigger number of variables and larger complexity separation between classical and quantum algorithms.

4.3.3.1 First Generalization of the Multifunction M_1

Definition 4.4. *Multifunction* $M_1^{GEN1} : \{0,1\}^N \to \{1,2,...,2(N-1)\}$ *associates each input element from the domain set with (N-1) output elements from the range set according to the following rule:*

$$\forall\ 1 < i \le N: \text{ if } (x_1 \oplus x_i = 0), \text{ then } M_1^{GEN1}(X) = 2(i-1)-1$$
$$\text{otherwise } M_1^{GEN1}(X) = 2(i-1)$$

It turns out that it is possible to compute multifunction M_1^{GEN1} classically in a uniformly distributed manner using two queries.

Theorem 4.5. $C_{UD}(M_1^{GEN1}) = 2$.

Proof. Classical probabilistic decision tree is demonstrated in Fig. 4.6. □

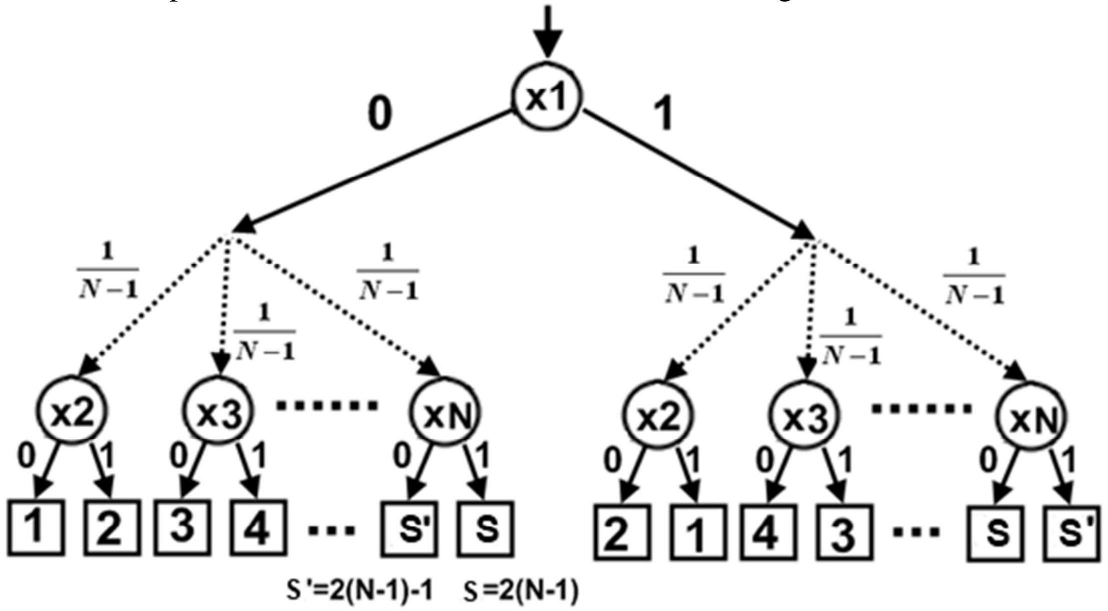


Fig. 4.6 Classical query algorithm for computing M_1^{GEN1} in a uniformly distributed manner

Theorem 4.6. $Q_{UD}(M_1^{GEN1}) = 1$.

Proof. To compute multifunction M_1^{GEN1} in a quantum query settings algorithm QM1 is extended to query all N multifunction variables in a single query. To be able to handle all variables, the quantum system is extended to have 2(N-1) basis states. Fig. 4.7 shows quantum algorithm QM1', which is an extended version of the algorithm QM1. H is the 2×2 Hadamard transformation, $\otimes$ denotes matrix tensor product operation. Quantum system consists of A qubits, where $A = \lceil \log_2(2(N-1)) \rceil$.

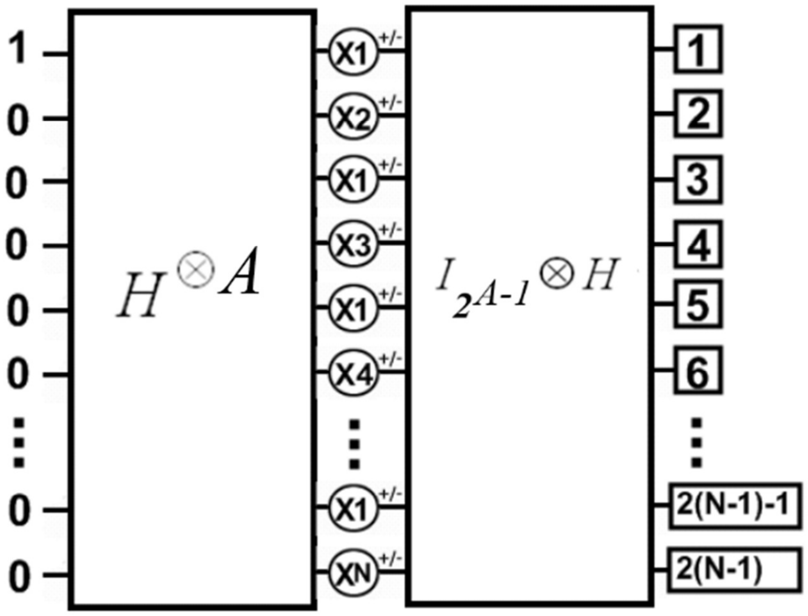


Fig. 4.7 Quantum query algorithm QM1' for computing M_1^{GEN1} in a uniformly distributed manner

Important moment is that variable x_1 is assigned to all odd amplitudes, but remaining variables $x_2,...,x_N$ are sequentially assigned to even amplitudes. □

In this example, the author enlarged the number of multifunction variables, but does not succeeded yet in enlarging the gap between classical and quantum query complexity.

4.3.3.2 Second Generalization of the Multifunction M_1

Subsequently, the author demonstrates another generalization of the multifunction M_1. This time a gap 2N versus N is achieved between the classical and the quantum query complexity.

The definition and behavior of multifunction M_1^{GEN2} is similar to multifunction M_1^{GEN1} - it associates each input element with (N-1) output elements from the range set. However, this time more variables are involved in the condition of the rule, which defines the multifunction.

Definition 4.5. *Multifunction* $M_1^{GEN2} : \{0,1\}^{N^2} \to \{1,2,...,2(N-1)\}$ *associates each input element from the domain set with (N-1) output elements from the range set according to the following rule:*

$$\forall\ 1 < i \le N: \text{ if } ((x_1 \oplus x_2 \oplus ... \oplus x_N) \oplus (x_{(i-1)N+1} \oplus x_{(i-1)N+2} \oplus ... \oplus x_{(i-1)N+N}) = 0),$$

$$\text{then } M_1^{GEN2}(X) = 2(i-1)-1$$

$$\text{otherwise } M_1^{GEN2}(X) = 2(i-1)$$

To compute multifunction M_1^{GEN2} in a classical query model 2*N* queries are required.

Theorem 4.7. $C_{UD}(M_1^{GEN2}) = 2N$.

Proof. In order to determine which range set element to include into the result set it is necessary to know values of all 2*N* variables involved into condition of the rule. A part of classical decision tree is depicted in Fig. 4.8. All sequentially queried variables are joined into one common query represented in the diagram by ellipses. Multiple arrows corresponding to common query outcomes are exiting these ellipses. □

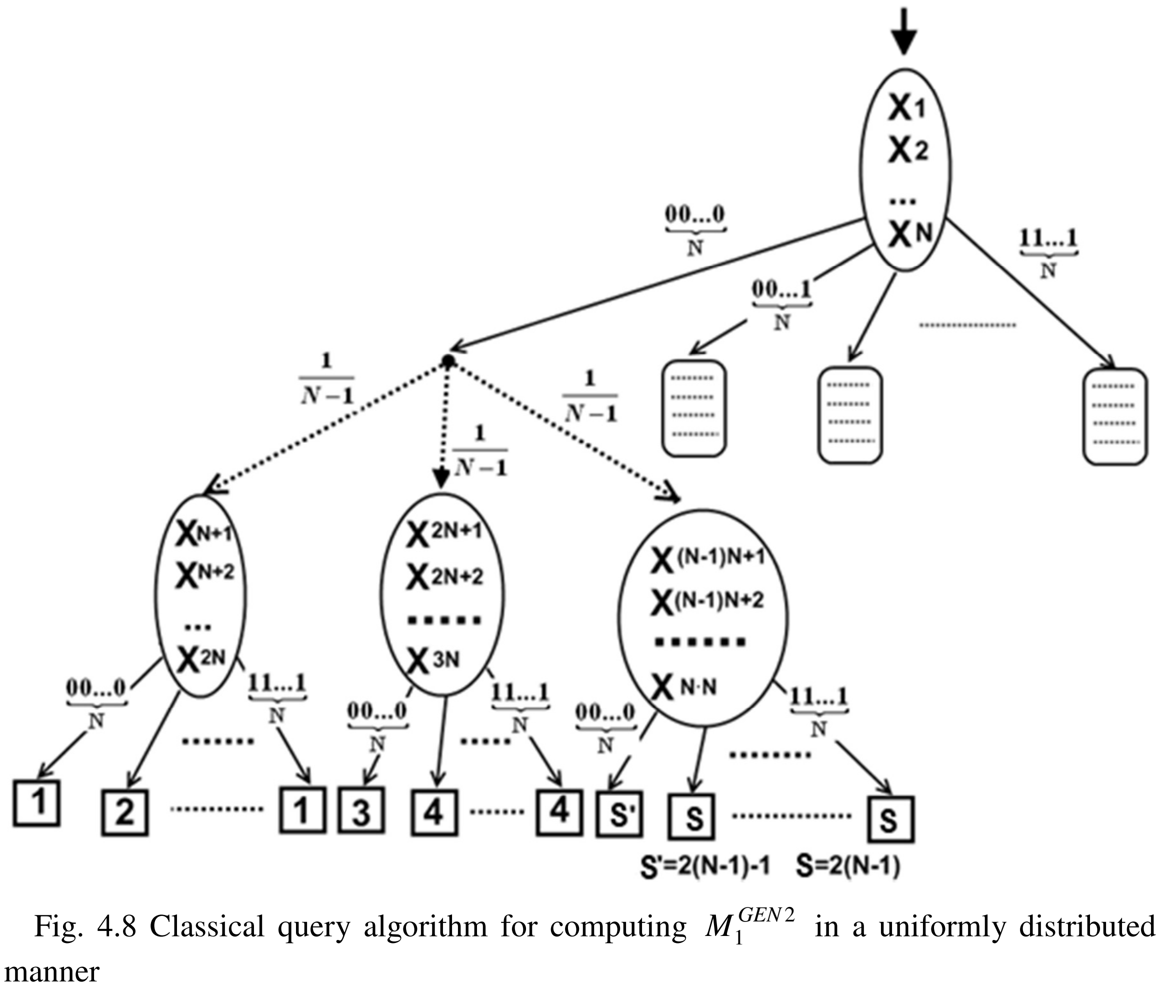
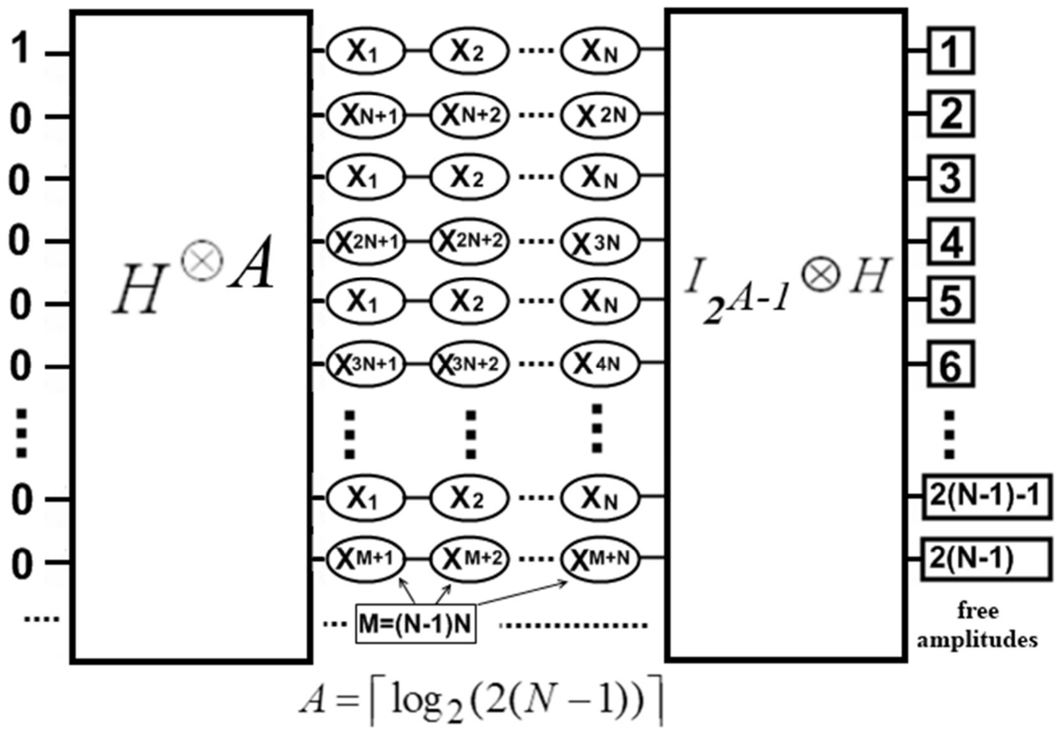


Fig. 4.8 Classical query algorithm for computing M_1^{GEN2} in a uniformly distributed manner

Theorem 4.8. $Q_{UD}(M_1^{GEN2}) \le N$.

Proof. General structure of the algorithm remains the same, but more queries are added. Algorithm QM1" is presented in Fig. 4.9 . Again, odd amplitudes all have the same set $x_1,...,x_N$ of queried variables assigned. Remaining variables are sequentially assigned to even amplitudes. □



1
0
0
0
0
0
0
0
....
$H^{\otimes A}$
X_1 X_2 .... X_N
X_{N+1} X_{N+2} .... X_{2N}
X_{2N+1} X_{2N+2} .... X_{3N}
X_{3N+1} X_{3N+2} .... X_{4N}
X_{M+1} X_{M+2} .... X_{M+N}
M=(N-1)N
$I_{2^{A-1}} \otimes H$
1
2
3
4
5
6
2(N-1)-1
2(N-1)
free amplitudes
$A = \lceil \log_2(2(N-1)) \rceil$

Fig. 4.9 Quantum query algorithm QM1' for computing M_1^{GEN2} in a uniformly distributed manner

The author demonstrated an approach for generalization of multifunctions to a larger number of variables. As a result, a complexity separation *N* versus 2*N* is obtained, which is the same as the largest separation between quantum exact and classical deterministic query algorithm for total Boolean functions known today. During computing multifunctions in a uniformly distributed manner there is no error probability and correct result is obtained with probability *p* = 1 as well (algorithm always outputs some correct value from the result set). However, the structure of considered multifunction is based on *XOR* operation. All examples of *N* versus 2*N* separations for functions, that the author is aware of, are directly based on *XOR* as well. The author is interested to find different cases, where *XOR* is not involved in obtaining a speed-up.

4.4 Second Example of Computing a Multifunction

Let us consider a TV company that offers minimal package and four more supplementary packages: movies, sports, social talk-shows and cartoons. Every client is free to choose any number of supplementary packages he is interested in. Company is willing to make a present for each client according to client's choice of packages. There are four different types of gift, let us mark them "1", "2", "3", "4".

Rule 1. If a client has one or three packages besides minimal package, company has to choose one from "1", "2", "3", "4" (probability to choose any gift from the scope has to be equally distributed between options, each having $p = 1/4$ to be selected).

Rule 2. If a client has only the minimal package or all four supplementary packages, company presents a gift of type "1".

Rule 3. If a client has chosen movies and social talk-shows or sports and cartoons, company presents a gift of type "2".

Rule 4. If a client has chosen movies and sports or social talk-shows and cartoons, company presents a gift of type "3".

Rule 5. If a client has chosen movies and cartoons or social talk-shows and sports, company presents a gift of type "4".

Table 4.3 defines a multifunction with Boolean domain and four-valued range: $M_2 : \{0,1\}^4 \to \{1,2,3,4\}$. Let us assign an index to each type of packages: 1 for movies, 2 for sports, 3 for social talk-shows and 4 for cartoons. Each bit in the input X gives the information whether i-th package is chosen by the client. 0000 means that only the minimal package is chosen, 1111 – full and so on.

Table 4.3. The definition of the multifunction M_2

X	$M_2(X)$	X	$M_2(X)$
0000	{1}	1000	{1,2,3,4}
0001	{1,2,3,4}	1001	{4}
0010	{1,2,3,4}	1010	{2}
0011	{3}	1011	{1,2,3,4}
0100	{1,2,3,4}	1100	{3}
0101	{2}	1101	{1,2,3,4}
0110	{4}	1110	{1,2,3,4}
0111	{1,2,3,4}	1111	{1}

4.4.1 Uniformly Distributed Query Complexity of the Multifunction M_2

Now, let us discuss the complexity of multifunction M_2. Computing M_2 is again examined in a uniformly distributed manner.

Theorem 4.9. $C_{UD}(M_2) \geq 3$.

Proof. Let us assume there exists a classical decision tree where all paths from root to leaves contain no more than two variables. When executing an algorithm on input

X=0000, result "1" has to be output with probability $p = 1$. It means that there exists a path from root to leaf with output value "1", which goes through two query nodes with some variables: $x_A = 0$ and $x_B = 0$. This path is depicted in Fig. 4.10. The fact is that it is not possible to select indices A and B to avoid contradictions with other inputs. For any choice of indices A and B there are four inputs that pass through the same path depicted in Fig. 4.10 and finish in a leaf with output value "1".

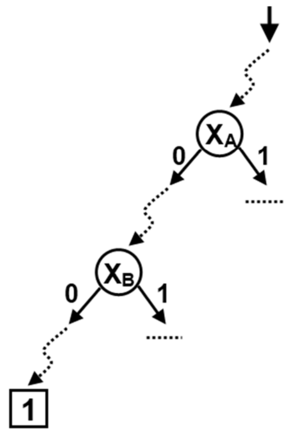


Fig. 4.10 Path for input X = 0000 in a potential classical randomized decision tree of depth d = 2 for computing M_2

Let us denote $X_i = x_1x_2x_3x_4$, A,B,C,D$\in \{1,2,3,4\}$, then these four inputs are as follows:

- X_1: $x_A = 0$, $x_B = 0$, $x_C = 0$, $x_D = 0$
- X_2: $x_A = 0$, $x_B = 0$, $x_C = 0$, $x_D = 1$
- X_3: $x_A = 0$, $x_B = 0$, $x_C = 1$, $x_D = 0$
- X_4: $x_A = 0$, $x_B = 0$, $x_C = 1$, $x_D = 1$

Let us consider the last input X_4, which has exactly two bits equal to "1". From Table 4.3 it is easy to see that for any input with exactly two "1" result set $M_2(X)$ consists of exactly one output value, which is always different from "1".

A contradiction is obtained: input X_4 passes through the path depicted in Fig. 4.10 and algorithm outputs incorrect value "1".

□

Theorem 4.10. $Q_{UD}(M_2) = 1$.

Proof. Quantum query algorithm QM2, which computes M_2 in a uniformly distributed manner with one query, is presented in Fig. 4.11. □

The author would like to note that definition of the multifunction M_2 and algorithm QM2 in some sense look similar to the definition and solution of the well-known Deutsch-Jozsa problem [25]. Careful reader could figure out this similarity by oneself. However, as is demonstrated further, generalization of that multifunction is not of that kind anymore.

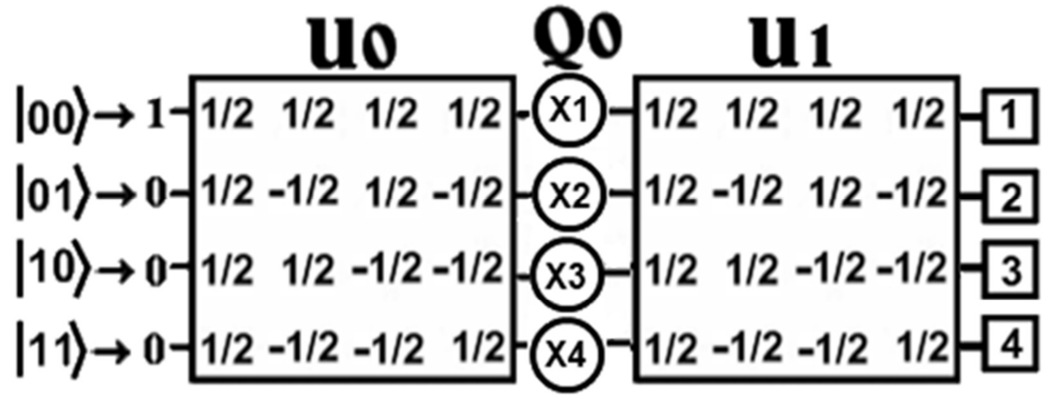


Fig. 4.11 Quantum query algorithm QM2 for computing M_2 in a uniformly distributed manner

4.4.2 Generalizations of the Multifunction M_2

Subsequently, the author presents two generalizations of the quantum query algorithm QM2. The first algorithm computes generalized multifunction in the same uniformly distributed manner, while the second one performs computation in randomly distributed manner.

4.4.2.1 First Generalization of the Multifunction M_2

Let us define the generalized version of multifunction $M_2^{GEN1}:\{0,1\}^{4N}\to\{1,2,3,4\}$.

Imagine that 4*N* variables are put on four vertical lines (*v*-lines) in such a way that:

$\forall i\in\{0,...N-1\},\forall k\in\{1,2,3,4\}: x_{4i+k}$ belongs to *v*-line number *k*.

For example, $x_1,x_5,x_9,x_{13},...$ are placed on the 1st *v*-line, $x_2,x_6,x_{10},x_{14},...$ - on the 2nd, and so on (see Fig. 4.12 for illustration).

The result set for each input *X* of the multifunction is defined as follows:

1. $M_2^{GEN1}(X)=\{1\}$, if all four *v*-lines of *X* contains either odd or even number of ”1”. For example, for the next inputs the multifunction’s result set is $\{1\}$:
   – input *X* = 00000000 has zero ”1” on each *v*-line
   – input *X* = 00010001 has zero ”1” on the first, second and third *v*-line and two ”1” on the fourth *v*- line
   – input *X* = 00001111 has one ”1” on each *v*-line
   – input *X* = 11111111 has exactly two ”1” on each *v*-line
2. $M_2^{GEN1}(X)=\{2\}$, if 1st and 3rd *v*-lines of *X* have odd number of ”1” and 2nd and 4th have even number of ”1”, or vice versa: 1st an 3rd – even and 2nd and 4th - odd. For example, for inputs 00000101, 00001010, 01011111, 11011000 multifunction’s result set is $\{2\}$.
3. $M_2^{GEN1}(X)=\{3\}$, if 1st and 2nd *v*-lines of *X* have odd number of ”1” and 3rd and 4th have even number of ”1”, or vice versa: 1st and 2nd – even and 3rd and 4th - odd. For example, for inputs 00000011, 00001100, 00111111, 10001011 multifunction’s result set is $\{3\}$.

4. $M_2^{GEN1}(X) = \{4\}$, if 1st and 4th v-lines of X have odd number of "1" and 2nd and 3rd have even number of "1", or vice versa: 1st and 4th - even and 2nd and 3rd - odd. For example, inputs 00000110, 00001001, 00111010, 10011111 multifunction's result set is $\{4\}$.
5. In all other cases, $M_2^{GEN1}(X) = \{1,2,3,4\}$.

Theorem 4.11. $Q_{UD}(M_2^{GEN1}) \leq N$.

Proof. Quantum algorithm that computes multifunction M_2^{GEN1} in a uniformly distributed manner with N queries is presented in Fig. 4.12. Each quantum query Q_i is defined by the following unitary matrix:

$$Q_i = \begin{pmatrix} (-1)^{x_{4i+1}} & 0 & 0 & 0 \\ 0 & (-1)^{x_{4i+2}} & 0 & 0 \\ 0 & 0 & (-1)^{x_{4i+3}} & 0 \\ 0 & 0 & 0 & (-1)^{x_{4i+4}} \end{pmatrix},\ i \in \{0,...,N-1\}.$$

□

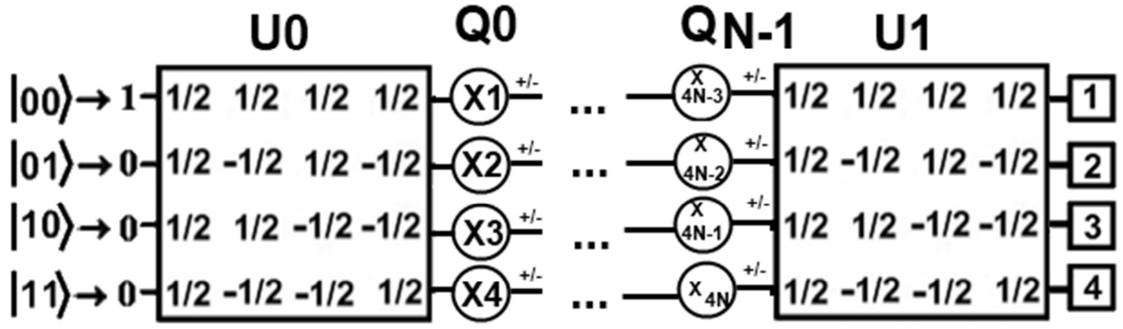


Fig. 4.12 Quantum query algorithm for computing M_2^{GEN1} in a uniformly distributed manner

Theorem 4.12. $C_{UD}(M_2^{GEN1}) \geq 3N$.

Proof. Let us assume there exists a classical decision tree that computes multifunction M_2^{GEN1} by asking 3N - 1 questions. All zeros input X = 00…0 is used to demonstrate a contradiction. Suppose arbitrary 3N - 1 variables are already queried, while N + 1 variables remain unquestioned.

On 4N-zeros input X = 00…0 algorithm has to output value "1" because all v-lines contain zero number of "1". Then, only such inputs are examined that have "0" in all queried 3N - 1 variables and exactly two "1" among remaining unquestioned variables. For all such inputs algorithm follows the same path in a tree and finishes in leaf with output value "1". However, all N + 1 unquestioned variables cannot be located on one v-line simply because each v-line consists of N variables. So, there is an input for which two "1" among unquestioned variables are located on different v-lines. According to the definition, result set in such case is {2} or {3} or {4}. Thus, algorithm outputs incorrect value for this input, this fact contradicts with the initial assumption, and implies $C_{UD}(M_2^{GEN1}) \geq 3N$. □

4.4.2.2 Second Generalization of the Multifunction M_2

Subsequently, the second way to generalize multifunction M_2 is demonstrated. In previous generalization more queries were added. This time the quantum system is extended and more variables are put in single query.

Suppose we are given a multifunction of N variables $M_2^{GEN2} : \{0,1\}^N \to \{1,2,...,N\}$, where N is power of 2. This time the author does not provide full definition of the multifunction; it follows from properties of quantum algorithm described below. The author just would like to demonstrate that such generalization is technically possible.

This time algorithm computes multifunction in a randomly distributed manner. Algorithm is allowed to output any value from result set with arbitrary probability, but probability for each value has to be positive: $p > 0$.

Theorem 4.13. $Q_{RD}(M_2^{GEN2}) = 1$.

Proof. The author adds more qubits and sequentially assigns variables to amplitudes. Given $N = 2^k, k \in \mathbb{N}$, quantum algorithm starts with k-qubit zero state $|0\rangle$, then applies $N \times N$ Hadamard transformation, N-variable query and finally applies $N \times N$ Hadamard transformation once again. Algorithm is depicted in Fig. 4.13. □

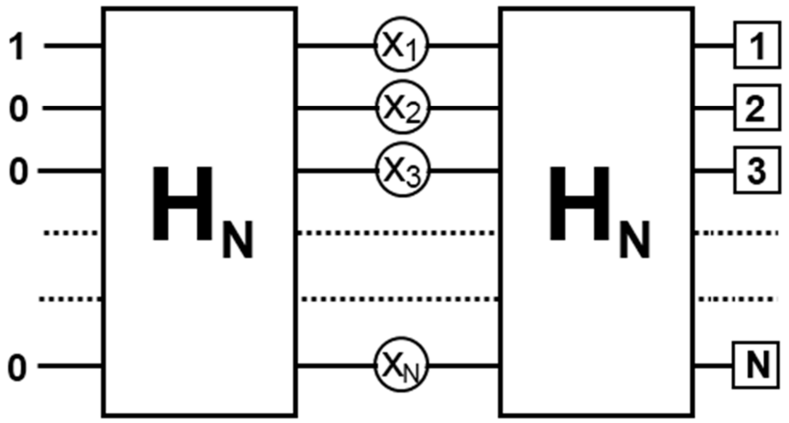


Fig. 4.13 Generalization of the quantum query algorithm for computing M_2^{GEN2}

Theorem 4.14. $C_{RD}(M_2^{GEN2}) \geq \frac{N}{2} + 1$.

Proof. Let us analyze the multifunction that is computable by the extended quantum algorithm. Imagine the first element of the quantum algorithm result vector (amplitude of the quantum basis state $|0\rangle$) right before the quantum measurement. It can be described by the formula:

$$\alpha_1 = \frac{(-1)^{x1} + (-1)^{x2} + ... + (-1)^{xN}}{N}.$$

If all $x_i = 0$, then $\alpha_1 = 1$, so for the input X = 00…0 algorithm outputs ”1” with probability p = 1. Let us suppose exactly N/2 variables are ”1” and N/2 are ”0”. In this case α_1 is precisely zero for all possible combinations. It means that probability to observe result value ”1” for any such input is p = 0.

Classical algorithm has to behave in the same way: for input X = 00…0 value ”1” has to be produced with probability p = 1, but for all inputs with exactly N/2 ”1” result value ”1” is not allowed to be output at all. This implies we are unable to recognize multifunction classically by asking only N/2 variable values, at least N/2+1 queries are required. □

4.5 Third Example of Computing a Multifunction

In this section, the author demonstrates the last example of computing a multifunction.

The eight-variable multifunction $M_3 : \{0,1\}^8 \to \{1,2,3,4\}$ is defined by the set of rules described in Table 4.4.

To figure out the result set for a certain input X it is necessary to find a matching rule in a table. Each rule $R(X)$ is described by the logical formula, which defines a condition that has to be satisfied by variable values. The last column contains the result set $S(R)$. Formally, $M(X) = S(R(X)) \Leftrightarrow R(X) = 1$.

Table 4.4. The definition of the multifunction M_3

Rule group	Suitable x_1, x_2, x_3, x_4	Matching rule $R(X)$ for $X = (x_1, ..., x_8)$	$S(R)$=$M_3(X)$
1	0000, 1111	$(x_1 = x_2 = x_3 = x_4) \& (x_5 = x_7)$	{1,2}
		$(x_1 = x_2 = x_3 = x_4) \& (x_5 \neq x_7)$	{3,4}
	0011, 1100	$(x_1 = x_2 \& x_3 = x_4 \& x_1 \neq x_3) \& (x_5 = x_7)$	{3,4}
		$(x_1 = x_2 \& x_3 = x_4 \& x_1 \neq x_3) \& (x_5 \neq x_7)$	{1,2}
2	0100, 1011	$(x_1 \neq x_2 \& x_3 = x_4 \& x_1 = x_3) \& (x_6 = x_7)$	{1,4}
		$(x_1 \neq x_2 \& x_3 = x_4 \& x_1 = x_3) \& (x_6 \neq x_7)$	{2,3}
	0111, 1000	$(x_1 \neq x_2 \& x_3 = x_4 \& x_1 \neq x_3) \& (x_6 = x_7)$	{2,3}
		$(x_1 \neq x_2 \& x_3 = x_4 \& x_1 \neq x_3) \& (x_6 \neq x_7)$	{1,4}
3	0001, 1110	$(x_1 = x_2 \& x_3 \neq x_4 \& x_1 = x_3) \& (x_5 = x_8)$	{1,4}
		$(x_1 = x_2 \& x_3 \neq x_4 \& x_1 = x_3) \& (x_5 \neq x_8)$	{2,3}
	0010, 1101	$(x_1 = x_2 \& x_3 \neq x_4 \& x_1 \neq x_3) \& (x_5 = x_8)$	{2,3}
		$(x_1 = x_2 \& x_3 \neq x_4 \& x_1 \neq x_3) \& (x_5 \neq x_8)$	{1,4}
4	0101, 1010	$(x_1 \neq x_2 \& x_3 \neq x_4 \& x_1 = x_3) \& (x_6 = x_8)$	{1,2}
		$(x_1 \neq x_2 \& x_3 \neq x_4 \& x_1 = x_3) \& (x_6 \neq x_8)$	{3,4}
	0110, 1001	$(x_1 \neq x_2 \& x_3 \neq x_4 \& x_1 \neq x_3) \& (x_6 = x_8)$	{3.4}
		$(x_1 \neq x_2 \& x_3 \neq x_4 \& x_1 \neq x_3) \& (x_6 \neq x_8)$	{1,2}

There is certain symmetry in the definition of the multifunction M_3. Input variables are logically split into two subsets, and values of variables in the first subset $\{x_1, x_2, x_3, x_4\}$ determine which values from the second subset $\{x_5, x_6, x_7, x_8\}$ are included in the second clause of the rule.

The important property is that the structure of the multifunction and algorithm for computing it are not based on *XOR* operation. In the area of quantum query algorithms for computing total functions, all examples that the author is aware of, in which quantum query complexity is twice smaller than classical query complexity (the largest known gap), are directly based on *XOR* operation.

Another important moment is that in this example the result set for each input consists of two elements and there is no input for which the result set consists of all possible output values.

4.5.1 Uniformly Distributed Query Complexity of the Multifunction M_3

Theorem 4.15. $C_{UD}(M_3) = 6$.

Proof. Straightforward classical algorithm with six queries is a deterministic decision tree with 50/50 randomization in output value after the last query. Four variables $x_1, \ldots, x_4$ are sequentially queried first, and then the last two variables for each path are selected based on results of the first four queries.

The lower bound proof is based on the following two properties of the multifunction.

Property 1. For any input, value inversion of exactly one variable present in the second clause of the corresponding rule changes the result set to its complement in the range set (e.g. from $\{1, 2\}$ to $\{3, 4\} = \{1, 2, 3, 4\} \setminus \{1, 2\}$).

Property 2. Every two inputs with the Hamming distance between first four variable values equal to 1 (value of exactly one variable is different) belong to different Rule Groups in the table which defines logical rules of M_3. Consequently, different two variables are included in the second clause of the rule and determine the result set.

Let us assume there exists a randomized decision tree of depth less than 6. Let us examine the computation process for all zeros input X = 00…0. According to the assumption, every computation path contains no more than five variable nodes and finishes in a leaf with result value either ”1” or ”2”. The following statements ensure that algorithm is not able to figure out the correct result value for X = 00…0 using five queries only:

1. In any case, there is no reason to include x_6 and x_8 in a path for X = 00...0 simply because these variables are not present in the rule.
2. Let us assume that either x_5 or x_7 is not included in a computation path (e.g. path consists of x_1, x_2, x_3, x_4, x_7 in arbitrary order). Then, computation for input with inversed value of that variable (e.g. X' = 00001000) would pass through the same path and finish in incorrect leaf ”1” or ”2” (result set for X' is $\{3, 4\}$). For all cases such behavior is ensured by the Property 1.
3. Let us assume that variable x_1, x_2, x_3 or x_4 is not included in a path. According to the Property 2, input with exactly one inversed variable among x_1, x_2, x_3, x_4 belongs to different Rule Group and the result set depends on two variables in the second clause of the rule different from x_5 and x_7 (e.g. for input X = 10000000 variables x_6 and x_7 are present in the second clause).

At least one of these variables is not present in considered path, so algorithm will not be able to determine result set correctly without additional query.

Above statements show that exclusion of any variable except x_6 and x_8 from the computation path for X = 00…0 results in uncertainty and break algorithm's ability to output correct value. So, assumption is wrong and six queries are required to compute M_3 in a uniformly distributed manner. □

Theorem 4.16. $Q_{UD}(M_3) \leq 2$.

Proof. Quantum query algorithm QM3 that computes M_3 with two queries is presented in Fig. 4.14.

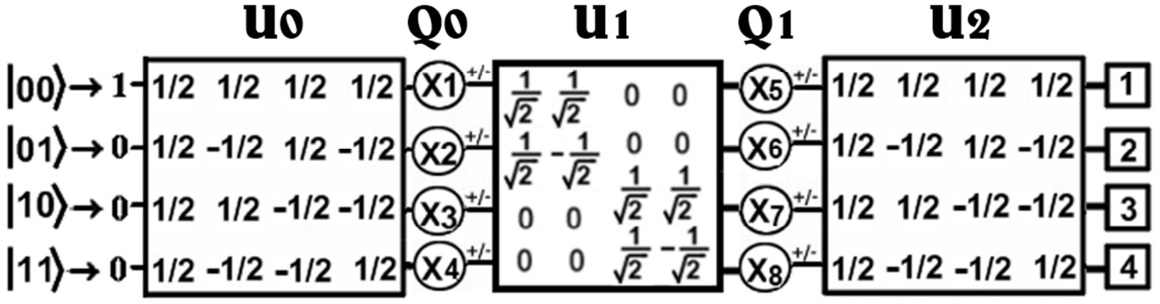


Fig. 4.14 Quantum query algorithm QM3 for computing M_3 in a uniformly distributed manner

Details of algorithm behavior are summarized in Table 4.5. The structure of the quantum algorithm completely corresponds to the rules that define multifunction M_3, thus it invariably produces the correct result. □

Table 4.5. Details of algorithm QM3 behavior on inputs matching different groups of rules

Rule group	Description of the state after U_1	Variables in 2nd query
1.1	Amplitudes of 1st and 3rd states have values $1/\sqrt{2}$ with equal signs	x_5 and x_7
1.2	Amplitudes of 1st and 3rd states have values $1/\sqrt{2}$ with different signs	
2.1	Amplitudes of 2nd and 3rd states have values $1/\sqrt{2}$ with equal signs	x_6 and x_7
2.2	Amplitudes of 2nd and 3rd states have values $1/\sqrt{2}$ with different signs	
3.1	Amplitudes of 1st and 4th states have values $1/\sqrt{2}$ with equal signs	x_5 and x_8
3.2	Amplitudes of 1st and 4th states have values $1/\sqrt{2}$ with different signs	
4.1	Amplitudes of 2nd and 4th states have values $1/\sqrt{2}$ with equal signs	x_6 and x_8
4.2	Amplitudes of 2nd and 4th states have values $1/\sqrt{2}$ with different signs	

4.5.2 Generalizations of the Multifunction M_3

Quantum query algorithm for computing multifunction M_3 is fairly interesting itself; moreover, it can be generalized to compute infinite family of multifunctions. The author briefly describes two methods to generalize quantum algorithm to compute multifunction with input of general size. Both methods can be applied to a variety of similar multifunctions, thus these are universal methods.

4.5.2.1 First Generalization of the Multifunction M_3

First generalization employs quantum parallelism - N instances of algorithm for M_3 are executed in parallel in a way depicted in Fig. 4.15.

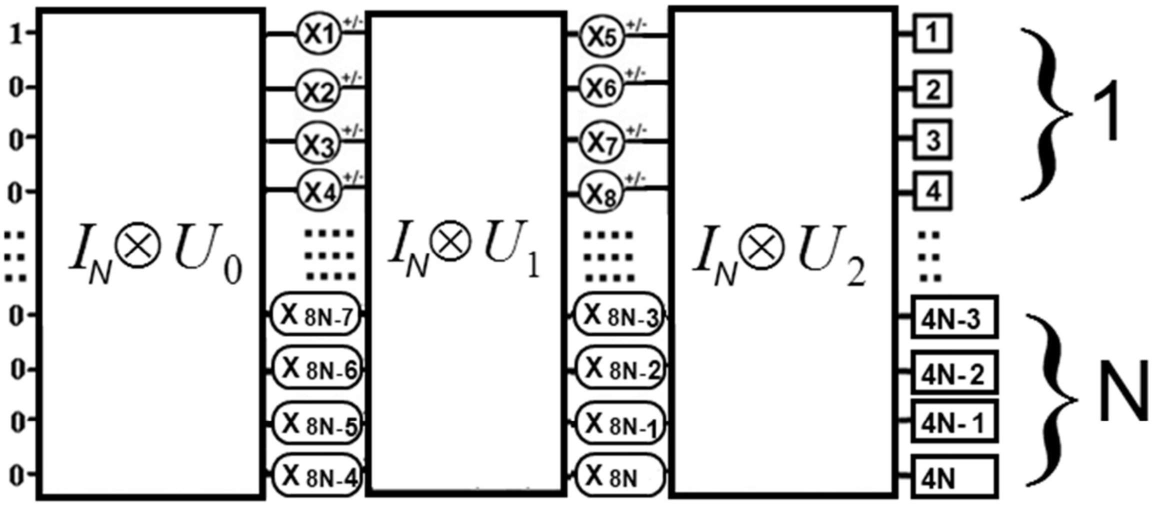


Fig. 4.15 First generalization of the quantum query algorithm QM3

As a result an algorithm is obtained, which computes multifunction with $8N$ variables. Range set consists of $4N$ values and every input is associated with $2N$ range set values. Quantum algorithm still uses two queries only. However, it is possible to compute the same multifunction classically using six queries. So, the number of multifunction variables is enlarged, but the quantum versus the classical complexity gap is not enlarged yet.

4.5.2.2 Second Generalization of the Multifunction M_3

The second generalization method is much more interesting. To generalize the multifunction the author associates each variable x_i with a sequence of new variables $x_i^1, x_i^2, ..., x_i^{N_i}$. In the definition of the multifunction an expression $x_i^1 \oplus x_i^2 \oplus ... \oplus x_i^{N_i}$ is substituted for each variable x_i. This way it is possible to define a multifunction with input of size:

$$S = N_1 + N_2 + ... + N_8.$$

Result set for each input is defined the same way using rules from Table 4.4. Just to figure out the value of each of eight input variables in a rule for M_3 first it is necessary to calculate XOR of the corresponding set of new variables.

Definition of new multifunction can be described by the following formula:

$$M_3^{GEN2}(x_1^1,...,x_1^{N_1},x_2^1,...,x_2^{N_2},......,x_8^1,...,x_8^{N_8}) =$$
$$= M_3(x_1^1 \oplus ... \oplus x_1^{N_1}, x_2^1 \oplus ... \oplus x_2^{N_2},......, x_8^1 \oplus ... \oplus x_8^{N_8}).$$

This way it is possible to generate an infinite family of multifunctions and corresponding quantum query algorithm for computing them.

In next theorems, the author assumes that $N_1 = N_2 = ... = N_8 = N$, so the total number of variables is $8N$. However, complexity estimations for arbitrary N_i values can be performed in a similar way.

Theorem 4.17. U*niformly distributed quantum query complexity of multifunction* M_3^{GEN2} *(in case when* $N_1 = N_2 = ... = N_8 = N$*) is:*

$$Q_{UD}(M_3^{GEN2}) \le 2N .$$

Proof. Quantum query algorithm can be easily adjusted to compute the generalized multifunction: on the algorithm diagram replace each query circle corresponding to variable x_i with a sequence of circles for new variables $x_i^1, x_i^2, ..., x_i^{N_i}$ inside (see Fig. 4.16).

According to the definition of the quantum query, each variable assigned to the horizontal line of the amplitude of the quantum basis state changes the sign of the amplitude to the opposite. Due to this property, sequential assignment of new variables in place of original variable x_i has the same effect as *XOR* operation in the definition of multifunction. □

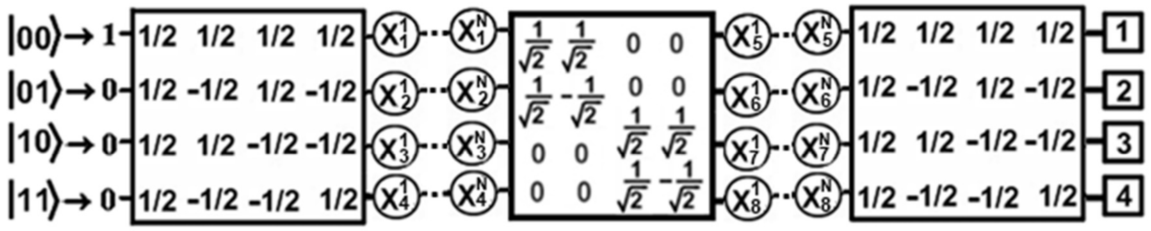


Fig. 4.16 Generalized quantum query algorithm for computing multifunction M_3^{GEN2} in a uniformly distributed manner

Theorem 4.18. *Classical uniformly distributed query complexity of multifunction* M_3^{GEN2} *(in case when* $N_1 = N_2 = ... = N_8 = N$*) is:*

$$C_{UD}(M_3^{GEN2}) = 6N .$$

Proof. The same as in the case of basic multifunction M_3, to figure out a matching rule for an arbitrary input it is needed to know the values of six sub-clauses $(x_i^1 \oplus x_i^2 \oplus ... \oplus x_i^{N_i})$ in the rule formula. For that reason, all corresponding $6N$ variables has to influence the computation of randomized decision tree and must be queried sequentially. Randomization has to be used at the final step only, to output uniform distribution of result set values. It is also not possible to save queries using randomization during the calculation of *XOR* of sub-clauses. □

4.6 Conclusion and Open Problems

In this section, the author examined computing multivalued functions in a query model. The author proposed three types of a query algorithm for computing multifunctions with the different output behavior. The author discussed the specifics of computing multifunctions in a definite manner and concluded that computing multifunctions in a distributed manner is more promising for enlarging a gap between classical and quantum query complexity. Finally, the author presented three examples of computing multifunctions in the classical and the quantum versions of a query model.

The following complexity gaps were obtained and presented in this section:

- $C_{UD}(M_1^{GEN2}) = 2N$ versus $Q_{UD}(M_1^{GEN2}) \le N$
- $C_{UD}(M_2^{GEN1}) \ge 3N$ versus $Q_{UD}(M_2^{GEN1}) \le N$
- $C_{RD}(M_2^{GEN2}) \ge \frac{N}{2} + 1$ versus $Q_{RD}(M_2^{GEN2}) = 1$
- $C_{UD}(M_3^{GEN2}) = 6N$ versus $Q_{UD}(M_3^{GEN2}) \le 2N$

Results presented build a foundation for further investigation. The main goal, which the author is looking to achieve, is constructing examples with even larger gap between the quantum and the classical algorithm complexity for the same multifunction. Important work direction is to improve techniques for proving complexity lower bounds for computing multifunctions in a classical query model.

5 Nondeterministic Query Algorithms

In this section, the nondeterministic algorithms are examined. A nondeterministic finite automaton, as introduced in [39], is a machine with many choices in its movements. On every stage it may choose one of several further internal states. The nondeterministic machine accepts a tape if there is at least one winning combination of choices of states leading to a designated final state. This is a traditional point of view on nondeterminism. In [40], nondeterministic algorithms are considered conceptual devices for simplifying the design of backtracking algorithms. The above study supports a view that algorithms are nondeterministic not in the sense of being random, but in the sense of having a free will. In [41], the authors present detailed definitions of nondeterministic finite automata, pushdown automata, Turing machine, and related results in complexity theory.

This section is organized as follows. In Section 5.1, a traditional quantum query model is examined, a notion of dual nondeterministic query algorithm is introduced and algorithm complexity in this model is studied. In Section 5.2, an alternative nondeterministic query model is introduced.

5.1 Traditional Nondeterministic Quantum Query Model

This section is based on the paper

- A. Dubrovska. Properties and Application of Nondeterministic Quantum Query Algorithms. *SOFSEM 2007: Theory and Practice of Computer Science*; Proc. Volume II; MatFyz Press; ISBN 80-903298-9-6; pp. 37-49, 2007

Nondeterministic quantum query algorithms (NQQA) were examined by de Wolf in [42]. For instance, it was proved that it is possible to compute a function

$$f(X)=1 \text{ iff } |X|\neq 1$$

using one query for all *N,* though it is proved that the best classical nondeterministic algorithm requires all *N* questions. This is the largest possible gap between complexities of two different kinds of algorithms permitted by the model.

Definition 5.1. [42] *A **nondeterministic quantum query algorithm** for f is defined to be a quantum algorithm that outputs 1 with positive probability if $f(X)=1$ and that always outputs 0 if $f(X)=0$.*

$NQ_1(f)$ denotes the query complexity of an optimal nondeterministic quantum algorithm for *f.*

5.1.1 Dual Nondeterministic Quantum Query Algorithms

The author introduces the concept of a *dual nondeterministic quantum query algorithm* and studies the relations between complexity of exact, nondeterministic and dual nondeterministic quantum query algorithms.

Definition 5.2 *A **dual nondeterministic quantum query algorithm** for f is defined to be a quantum algorithm that outputs 0 with positive probability if $f(X)=0$ and that always outputs 1 if $f(X)=1$.*

$NQ_0(f)$ denotes the query complexity of an optimal dual nondeterministic quantum algorithm for *f.*

5.1.2 Properties of NQQA

In this section, the author discusses and demonstrates certain properties of nondeterministic quantum query algorithms. First, the author describes the relation between complexities of nondeterministic and dual nondeterministic quantum query algorithms.

Lemma 5.1 *A nondeterministic quantum query algorithm for a function f can be transformed in a dual nondeterministic algorithm for a function* $\overline{f}$ *by replacing the assigned values for each output* $j \in \{0,1\}$ *by* $(1-j)$. *The same is true for transforming algorithms in the opposite direction.*

Proof. Let $A1$ be a dual nondeterministic algorithm for a function *f*(*X*) with complexity $NQ_0(A1) = k$. The algorithm is executed on all inputs and depending on the result inputs are divided into three sets:

$$A = \{X \mid p(f(X)=1)=1\}$$
$$B = \{X \mid p(f(X)=0)=1\}$$
$$C = \{X \mid p(f(X)=0)>0 \,\&\, p(f(X)=1)>0 \,\&\, p(f(X)=0)+p(f(X)=1)=1\}$$

According to the definition of the dual nondeterministic algorithm, the result value of running $A1$ on pertinent input is assigned to each set (see Table 5.1).

Table 5.1. Results of running algorithm $A1$

X belongs to set:	Result of $A1$
A	1
B	0
C	0

Table 5.2. Results of running algorithm $A1$'

X belongs to set:	Result of $A1'$
A	0
B	1
C	1

Subsequently, the value assignment for each output is changed to the opposite: $0 \to 1$ and $0 \to 1$. The author denotes a new algorithm with $A1'$ and proves it to be a correct nondeterministic algorithm for $\overline{f}$.

After running $A1'$ author examines the same input sets *A*, *B* and *C* and obtains opposite probabilities for sets *A* and *B*:

$$A = \{X \mid p(f(X)=0)=1\}$$
$$B = \{X \mid p(f(X)=1)=1\}$$
$$C = \{X \mid p(f(X)=0)>0 \,\&\, p(f(X)=1)>0 \,\&\, p(f(X)=0)+p(f(X)=1)=1\}$$

Subsequently, establishing the function value obtained by running $A1'$ according to the nondeterministic algorithm definition, the results presented in Table 5.2 are obtained.

Comparing the results derived with values from Table 5.1 the author concludes that $A1'$ computes $\overline{f}$ as a nondeterministic algorithm.

The proof in the opposite direction is similar. □

Theorem 5.1 *For an arbitrary Boolean function f,* $NQ_0(f) = NQ_1(\overline{f})$.

Proof. The proof follows from Lemma 5.1. The best existing dual nondeterministic algorithm can be taken for function f and easily transformed into a nondeterministic algorithm for $\overline{f}$. Only the value assignment to outputs is changed; the number of questions $NQ_0(f)$ remains the same. In the opposite direction, it is possible to take the best existing nondeterministic algorithm for $\overline{f}$ and transform it into a dual nondeterministic for f staying with the same $NQ_1(\overline{f})$ queries. □

As the next step, the author examines composite functions and demonstrates a way to use an exact quantum query algorithm for a function to construct a nondeterministic quantum algorithm for a more complex function.

The first structure is the composite function *MULTI_AND*. The author denotes:

$$MULTI\_AND_m(x_1,...,x_{mn}) = \underbrace{f_n(x_1,...,x_n) \wedge f_n(x_{n+1},...,x_{2n}) \wedge ... \wedge f_n(x_{(m-1)n+1},...,x_{mn})}_{m}$$

$f_n(x_1,...,x_n)$ is called base function or sub-function. The composite function *MULTI_AND* is obtained using a base function structure as a pattern, joining several similar variable blocks by a logical *AND* operation.

The second structure is a composite function *MULTI_OR*. The author denotes:

$$MULTI\_OR_m(x_1,...,x_{mn}) = \underbrace{f_n(x_1,...,x_n) \vee f_n(x_{n+1},...,x_{2n}) \vee ... \vee f_n(x_{(m-1)n+1},...,x_{mn})}_{m}$$

Theorem 5.2 *Let Q1 be an exact quantum query algorithm that computes a Boolean function f with k queries. Consequently, a dual nondeterministic quantum query algorithm Q2 exists, computing function MULTI_AND$_m$(f) with the same k queries for all m.*

Proof. Let $Q1$ be an exact quantum algorithm with k queries for an N-variable function f. The author identifies the assignment of function values for outputs by $M = (q_1 \equiv k_1, q_2 \equiv k_2, ..., q_h \equiv k_h)$, where h is a number of amplitudes.

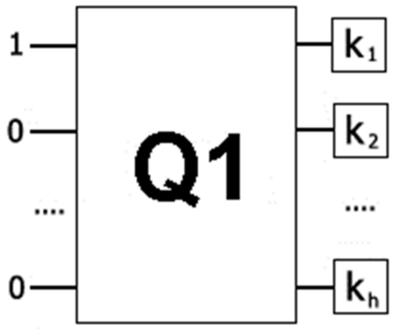


Fig. 5.1 Exact quantum algorithm $Q1$ for computing f

Further, the author constructs a quantum algorithm *Q*2, computing *MULTI_AND_m*(*f*) as a dual nondeterministic algorithm. The notation is the following:

$MULTI\_AND_m(X) = f(X_1) \wedge f(X_2) \wedge ... \wedge f(X_m)$, where $X = X_1 X_2 X_3 ... X_m$, $\forall i \in \{1..m\}$: $X_i = (x_{i1} = \alpha_{i1}, x_{i2} = \alpha_{i2}, ..., x_{iN} = \alpha_{iN})$ and $\alpha_{ij} \in \{0,1\}$, it is allowed that $X_i = X_j$, even if $i \neq j$.

To establish the value for each of *m* occurrences of *f*, algorithm *Q*1 is executed in parallel. To retain the total sum of squares of amplitudes equal to 1, the initial amplitude distribution is separated between all *m* parts of *Q*2. All *Q*1 transformations are unitary and from the structure of the algorithm it follows that *Q*2 transformations are also unitary. Algorithm *Q*2 is demonstrated in Fig. 5.2.

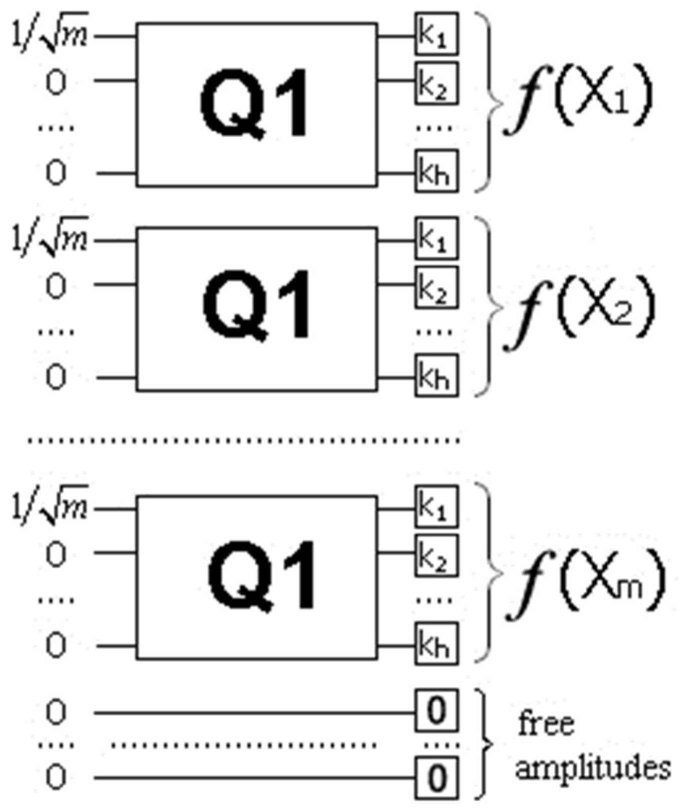


Fig. 5.2 Dual nondeterministic quantum query algorithm *Q*2

The author denotes the sum of squares of all amplitudes where the output value is "0" in the part of *Q*2 corresponding to $f(X_i)$ by $P_{f(X_i)}("0")$. In a similar way, by $P_{f(X_i)}("1")$, the sum of squares of all amplitudes of $f(X_i)$ parts corresponding to outputs with assigned "1" is identified.

If after running *Q*1 on some input *X_i*, a final distribution of amplitudes $(b_{i1}, b_{i2}, ..., b_{ih})$ appears, then computing $f(X_1) \wedge f(X_2) \wedge ... \wedge f(X_m)$ with *Q*2, the following final distribution of amplitudes will be obtained:

$$\left((b'_{11}, b'_{12}, ..., b'_{1h}), (b'_{21}, b'_{22}, ..., b'_{2h}), ..., (b'_{m1}, b'_{m2}, ..., b'_{mh}), 0, ..., 0\right), \text{ where } b'_{ij} = \frac{1}{\sqrt{m}} b_{ij}.$$

Lemma 5.2 *For an arbitrary input X_i:*

- *If the result of running Q1 is f(X_i)=0, then for Q2 the outcome is* $P_{f(X_i)}("0") = \frac{1}{m}$ *and* $P_{f(X_i)}("1") = 0$.
- *If the result of running Q1 is f(X_i)=1, then for Q2 the outcome is* $P_{f(X_i)}("0") = 0$ *and* $P_{f(X_i)}("1") = \frac{1}{m}$.

Proof. The proof follows from the properties of exact quantum query algorithm and the fact that the value assignment for outputs in each part of $Q2$ is the same as in $Q1$. □

For the entire algorithm $Q2$:

- If there exists at least one $i \in \{1..m\}$ for which $f(X_i) = 0$, then according to Lemma 5.2 in the part of $Q2$ corresponding to $f(X_i)$ $P_{f(X_i)}("0") = \frac{1}{m}$. The total probability of obtaining the result "0" is $p(f(X) = 0) > 0$ and in the sense of a dual nondeterministic algorithm definition, $MULTI\_AND(X) = 0$.
- If for all $i \in \{1..m\}$ $f(X_i) = 1$, then in all $Q2$ parts $P_{f(X_i)}("1") = \frac{1}{m}$ and the total probability is $p(f(X) = 1) = 1$, so in the sense of a dual nondeterministic algorithm definition, $MULTI\_AND(X) = 1$.

The above completely conforms to the essence of a function *MULTI_AND*; hence, $Q2$ computes this function as a dual nondeterministic algorithm. □

The next theorem can be used for complexity estimation.

Theorem 5.3 *For an arbitrary Boolean function f,*

$$NQ_0(MULTI\_AND_m(f)) \le Q_E(f).$$

Proof. Using the approach from Theorem 5.2, it is always possible to construct a dual nondeterministic algorithm for $MULTI\_AND_m(f)$ based on the best existing exact algorithm for *f*. In this case, the equality of complexities is achieved. It may be possible to find a better algorithm for $MULTI\_AND_m(f)$ using a completely different method, and therefore there is an inequality in total estimation. □

A similar result is obtained with a nondeterministic quantum query algorithm and the structure *MULTI_OR*.

Theorem 5.4 *Let Q1 be an exact quantum query algorithm that computes Boolean function f with k queries. Consequently, a nondeterministic quantum query algorithm Q2 exists computing the function MULTI_OR$_m$(f) with the same k queries for all m.*

Proof. The author uses the same algorithm $Q2$ from the proof of Theorem 5.2.
This time the results of running $Q2$ are interpreted as follows:

- If there exists at least one $i \in \{1..m\}$ for which $f(X_i)=1$, then according to Lemma 5.2 in a part of $Q2$ corresponding to $f(X_i)$ $P_{f(X_i)}("1")=\frac{1}{m}$. The total probability of obtaining result "1" will be $p(f(X)=1)>0$ and in the sense of a dual nondeterministic algorithm definition, $MULTI\_OR(X)=1$.
- If for all $i \in \{1..m\}$ $f(X_i)=1$, then in all $Q2$ parts $P_{f(X_i)}("0")=\frac{1}{m}$ and the total probability is $p(f(X)=0)=1$, so in the sense of a dual nondeterministic algorithm definition, $MULTI\_OR(X)=0$.

The above-stated completely agrees with the essence of the function *MULTI_OR*; hence, *Q*2 computes this function as a nondeterministic algorithm. □

Theorem 5.5 *For an arbitrary Boolean function f,* $NQ_1(MULTI\_OR_m(f)) \le Q_E(f)$.

Proof. Similar to the proof of Theorem 5.3. □

In the next two theorems, the author generalizes the obtained results to enable operations with compositions of arbitrary Boolean functions.

Theorem 5.6 *Let f_i be an arbitrary Boolean function. Let us examine a function $F=f_1 \wedge f_2 \wedge ... \wedge f_n$. A dual nondeterministic quantum query algorithm Q exists computing F with max($Q_E(f_1)$,$Q_E(f_2)$,...,$Q_E(f_n)$) queries.*

Proof. All exact algorithms for $f_1, f_2,...,f_n$ are executed in parallel, combining the queries (Fig. 5.3).

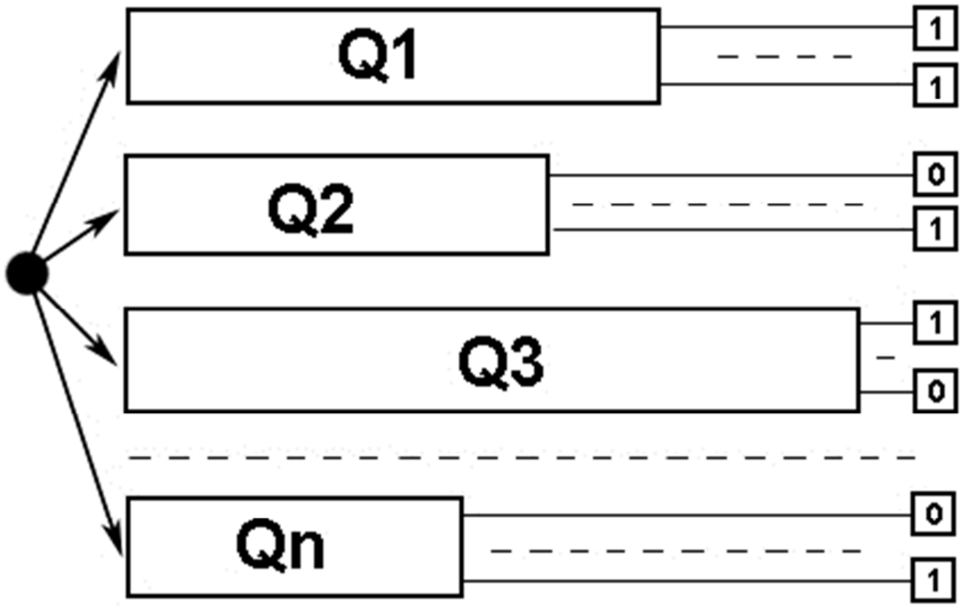


Fig. 5.3 Dual nondeterministic algorithm for $F=f_1 \wedge f_2 \wedge ... \wedge f_n$

The proposition of Lemma 5.2 is true for the algorithm Q; thus, it meets the properties of $F = f_1 \wedge f_2 \wedge \ldots \wedge f_n$. Exact algorithms are executed and questions are asked in parallel, so the complexity of the entire algorithm equals the largest number of queries of the corresponding exact algorithms. □

Theorem 5.7 *Let f_i be an arbitrary Boolean function. Let us examine a function $F = f_1 \vee f_2 \vee \ldots \vee f_n$. A nondeterministic quantum query algorithm Q exists computing F with $\max(Q_E(f_1), Q_E(f_2), \ldots, Q_E(f_n))$ queries.*

Proof. Similar to the proof of Theorem 5.6. □

5.1.3 Application of NQQA Properties

In this section, the author provides several examples on application of nondeterministic quantum query algorithms and their properties for efficient computation of Boolean functions.

5.1.3.1 Demonstration of the Approach

This subsection is dedicated to demonstrating the technical performance of constructing a dual nondeterministic quantum algorithm for a composite function using exact quantum algorithms for sub-functions.

Exact quantum query algorithms for the following two Boolean functions serve as the basis for the demonstration:

$$F_3(x_1, x_2, x_3) = \neg(x_1 \oplus x_2) \wedge (x_1 \oplus x_3)$$
$$G_4(x_1, x_2, x_3, x_4) = (x_1 \oplus x_2) \wedge (x_3 \oplus x_4)$$

The exact quantum query algorithms for these functions are presented in Fig. 5.4 and Fig. 5.5 and are obtained by applying transformation methods to base algorithms described in Section 3.1.1.

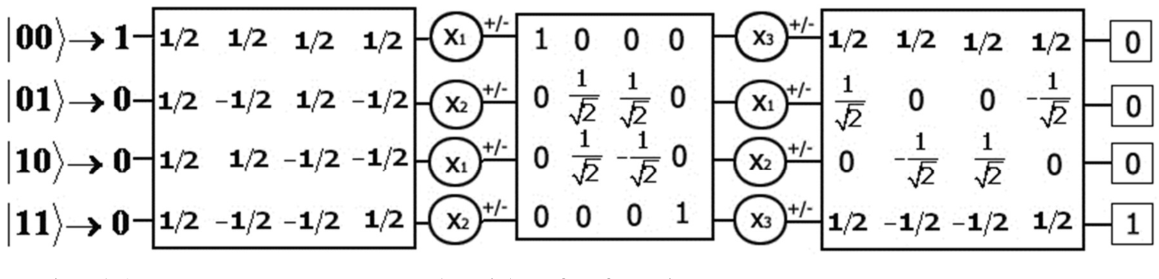


Fig. 5.4 Exact quantum query algorithm for function F_3

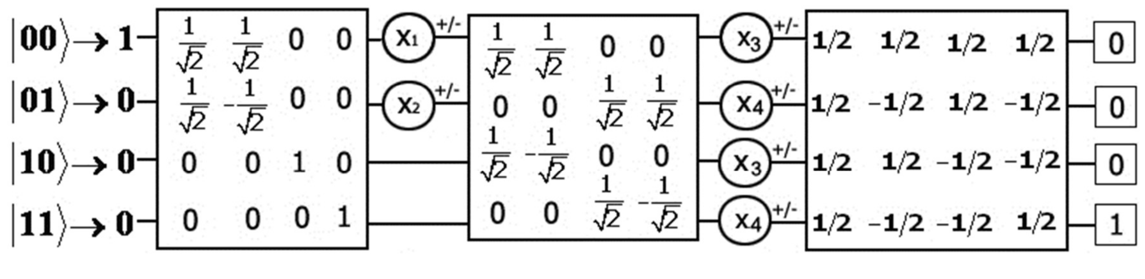


Fig. 5.5 Exact quantum query algorithm for function G_4

The author examines the composite function obtained by combining F_3 and G_4:

$$H_7(x_1,x_2,x_3,x_4,x_5,x_6,x_7)=(\neg(x_1\oplus x_2)\wedge(x_1\oplus x_3))\wedge((x_4\oplus x_5)\wedge(x_6\oplus x_7)).$$

The sensitivity of the base functions is $s(F_3)=3$ and $s(G_4)=4$. It is evident that the sensitivity of H_7 is equal to the number of variables, so $D(H_7)=7$.

According to Theorem 5.6, a dual nondeterministic algorithm exists with $\max(Q_E(F_3),Q_E(G_4))=2$ queries only.

Using a similar approach to one applied for the proof of Theorem 5.2, the author is able to completely describe a dual nondeterministic algorithm computing function H_7 (see Fig. 5.6).

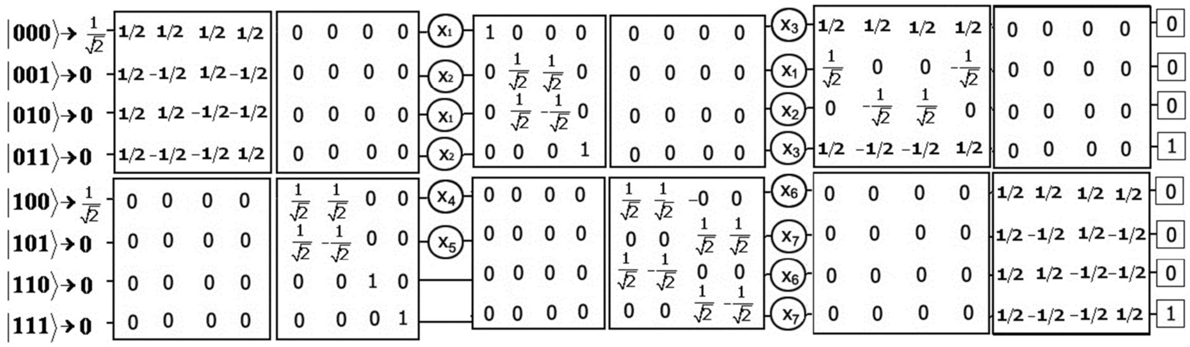


Fig. 5.6 Dual nondeterministic quantum query algorithm for H_7

5.1.3.2 Dual Nondeterministic Quantum Query Algorithm for *Control*$_N$ Function

The author introduces a function and proves a gap $O(N)$ versus $O(1)$ between the complexity of the deterministic and the dual nondeterministic quantum algorithms.

Control$_N$ is defined to be a Boolean function of $N=2k-1$ variables:

$$Control_N(x_1,x_2,...,x_k,x_{k+1},...,x_{2k-1})=1 \quad\Leftrightarrow\quad \begin{cases} x_{k+1}=x_1\oplus x_2 \\ x_{k+2}=x_1\oplus x_2\oplus x_3 \\ .... \\ x_{2k-2}=x_1\oplus x_2\oplus ...\oplus x_{k-1} \\ x_{2k-1}=x_1\oplus x_2\oplus ...\oplus x_{k-1}\oplus x_k \end{cases}$$

The essence of the function is that in each accepting input X, for which *Control*$_N$(X) = 1, values of the first j bits control the value of (k+j)'s bit.

Theorem 5.8 $D(Control_N)=N$.

Proof. Follows from the sensitivity of the function on zero input X = 00..0. □

Theorem 5.9 *There is a dual nondeterministic quantum algorithm computing Control*$_N$ *with two queries for all N.*

Proof. First, the equation system is transformed as follows:

$$\begin{cases} x_{k+1} = x_1 \oplus x_2 \\ x_{k+2} = x_1 \oplus x_2 \oplus x_3 \\ \dots\dots\dots\dots\dots\dots \\ x_{2k-1} = x_1 \oplus x_2 \oplus \dots \oplus x_k \end{cases} \Rightarrow \begin{cases} x_{k+1} = x_1 \oplus x_2 \\ x_{k+2} = x_{k+1} \oplus x_3 \\ \dots\dots\dots\dots\dots\dots \\ x_{2k-1} = x_{2k-2} \oplus x_k \end{cases} \Rightarrow \begin{cases} x_{k+1} \oplus x_1 \oplus x_2 = 0 \\ x_{k+2} \oplus x_{k+1} \oplus x_3 = 0 \\ \dots\dots\dots\dots\dots\dots \\ x_{2k-1} \oplus x_{2k-2} \oplus x_k = 0 \end{cases}$$

It is possible to rewrite the statement:

$$\begin{cases} x_{k+1} \oplus x_1 \oplus x_2 = 0 \\ x_{k+2} \oplus x_{k+1} \oplus x_3 = 0 \\ \dots\dots\dots\dots\dots\dots \\ x_{2k-1} \oplus x_{2k-2} \oplus x_k = 0 \end{cases} \Leftrightarrow Control_N(X) = 1$$

with equivalent logical formulas:

$$Control_N = \neg((x_{k+1} \oplus x_1 \oplus x_2) \vee (x_{k+2} \oplus x_{k+1} \oplus x_3) \vee \dots \vee (x_{2k-1} \oplus x_{2k-2} \oplus x_k))$$

$$Control_N = \neg(x_{k+1} \oplus x_1 \oplus x_2) \neg(x_{k+2} \oplus x_{k+1} \oplus x_3) \wedge \dots \wedge \neg(x_{2k-1} \oplus x_{2k-2} \oplus x_k)$$

$$Control_N = \neg(PARITY_3(X_1)) \wedge \neg(PARITY_3(X_2)) \wedge \dots \wedge \neg(PARITY_3(X_{k-1}))$$

$$\boldsymbol{Control_N = MULTI\_AND_{k-1}(\neg PARITY_3)}$$

From Theorem 5.2 follows that the dual nondeterministic quantum algorithm Q for the function $Control_N$ exists, so that $NQ_0(Q) = Q_E(\neg PARITY_3)$.

Fig. 5.7 demonstrates a quantum exact algorithm with two questions for a function $\neg PARITY_3(X) = \neg(x_1 \oplus x_2 \oplus x_3)$. Here H is Hadamard gate $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

Fig. 5.8 demonstrates the structure of a complete dual nondeterministic algorithm computing $Control_N$. □

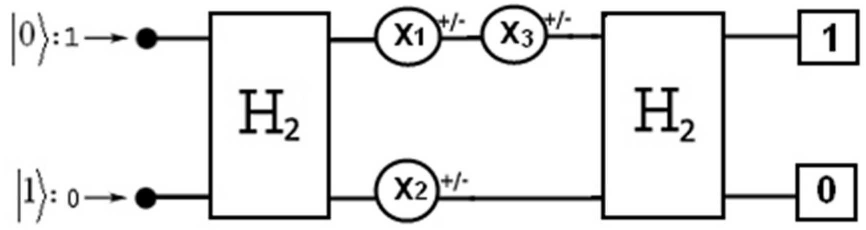


Fig. 5.7 Exact quantum algorithm for $\neg PARITY_3(X)$

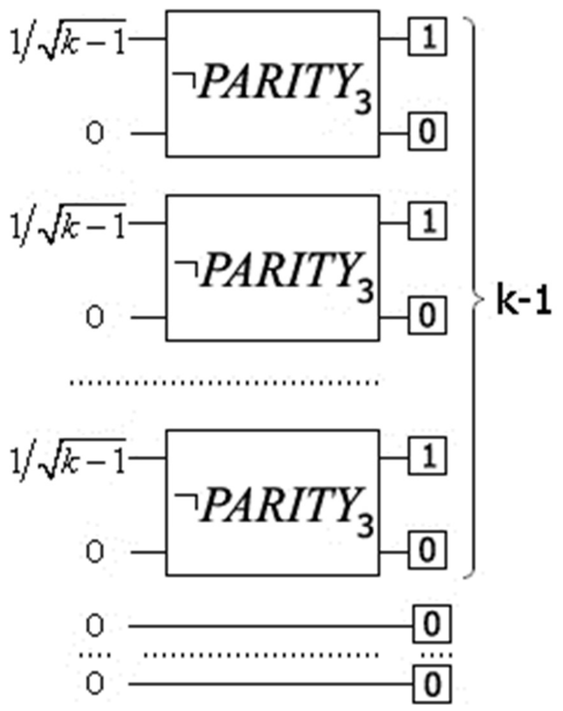


Fig. 5.8 Dual nondeterministic quantum query algorithm for $Control_N$ with two queries

5.1.3.3 Dual Nondeterministic Quantum Query Algorithm for *PAIR_EQUALITY$_N$* Function

Using the properties of dual nondeterministic algorithms it is easy to figure out an efficient algorithm for the *PAIR_EQUALITY$_N$* function (first defined in Section 3.1.1.2).

The Boolean function is defined by the formula ($N = 2k$):

$$PAIR\_EQUALITY_{2k}(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_3 \oplus x_4) \wedge \ldots \wedge \neg(x_{2k-1} \oplus x_{2k}) .$$

Theorem 5.10 $D(PAIR\_EQUALITY_N) = N$.

Proof. Follows from the sensitivity of the function on any accepting input. □

Theorem 5.11 *There is a dual nondeterministic quantum algorithm computing PAIR_EQUALITY$_N$ with one query for all N.*

Proof. It is possible to rewrite the logical formula as a *MULTI_AND* of *PARITY*:

$$PARITY_2(x_1, x_2) = \neg(x_1 \oplus x_2) ,$$

$$PAIR\_EQUALITY_{2k}(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_3 \oplus x_4) \wedge \ldots \wedge \neg(x_{2k-1} \oplus x_{2k}) =$$

$$= PARITY_2(x_1, x_2) \wedge PARITY_2(x_3, x_4) \wedge \ldots \wedge PARITY_2(x_{2k-1}, x_{2k}) = MULTI\_AND_k(PARITY_2)$$

It is well known that $Q_E(PARITY_2) = 1$, so from Theorem 5.2 follows:

$$NQ_0(PAIR\_EQUALITY_{2k}) = Q_E(PARITY_2) = 1 .$$

Algorithm structure is illustrated in Fig. 5.9. □

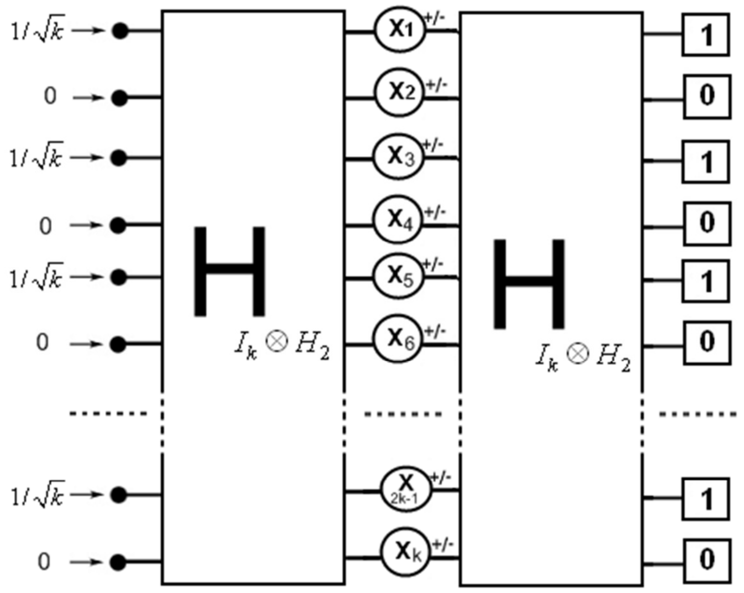


Fig. 5.9. Dual nondeterministic quantum query algorithm for *PAIR_EQUALITY*$_N$ with one query

5.1.4 Conclusion and Open Problems

In this section, nondeterministic quantum query algorithms were studied. First, a new concept of a dual nondeterministic quantum query algorithm was introduced followed by proving its relations with the complexity of nondeterministic and exact algorithms for several classes of Boolean functions. The results show the nondeterministic algorithms examined to be a powerful and an efficient model. A new function *Control*$_N$ was introduced and a dual nondeterministic algorithm for computing it with two queries for all *N* was presented.

The future direction of this research study is to prove stronger relations to other types of query algorithms, for example, to exact quantum algorithms of the same function, classical nondeterministic query algorithms and even classical deterministic algorithms. It would also be useful to discover efficient quantum nondeterministic algorithms for specific functions, revealing large gaps between complexities of different kinds of algorithms.

5.2 Alternative Nondeterministic Query Model

This section is based on the paper

- A. Vasilieva, R. Freivalds. Nondeterministic Query Algorithms. *Journal of Universal Computer Science (J. UCS)* 17(6): 859-873 (2011)

In [40], nondeterministic algorithms are considered conceptual devices to simplify the design of backtracking algorithms. The above study supports a view that algorithms are nondeterministic not in the sense of being random, but in the sense of having a *free will*.

The author investigates the nature of the above-mentioned nondeterministic free will. The author provides a way to measure the amount of nondeterminism in an algorithm. In the traditional nondeterministic query model the power of nondeterminism comes with no cost. The idea is that the algorithm must pay with additional queries for the nondeterministic help. The author introduces an alternative definition of the nondeterministic query model, which incorporates behavior described above.

Definition of the nondeterministic quantum query algorithms as first introduced in [42] seems a bit counter intuitive. This is another motivation to introduce a new approach for nondeterminism in query algorithms.

This section is organized as follows. In Section 5.2.1, an alternative model for nondeterministic query computation is introduced. In Section 5.2.2, an example of computing the Fano plane Boolean function in proposed model is demonstrated.

5.2.1 Definition of the Alternative Nondeterministic Query Model

In this subsection, the author introduces an alternative definition of a nondeterministic query model. The main idea is that in this variation the power of nondeterminism is not received free of charge, but the algorithm have to spend additional queries to obtain a nondeterministic help.

Suppose that the task is to compute some arbitrary Boolean function $F(X)$ in an alternative nondeterministic query model. Then, the first step is to define a nondeterministic helper function $G(X,Y)$. This function has to satisfy definite conditions, which will be precisely specified a little bit later. The second step is to design a deterministic query algorithm for the function $G(X,Y)$. Finally, the nondeterministic query complexity of the function $F(X)$ is equal to the complexity of the deterministic query algorithm for a nondeterministic helper function $G(X,Y)$.

Subsequently, the author provides formal definitions for the computational model informally described above.

Definition 5.3 *The* ***nondeterministic helper function G(X,Y)*** *for the Boolean function F(X) is a partial Boolean function, which satisfies the following conditions:*

1. $\forall x_1, ..., x_n, \exists y_1, ..., y_k,$ *such that* $G(x_1, ..., x_n, y_1, ..., y_k) = F(x_1, ..., x_n)$;
2. $\forall x_1, ..., x_n, \neg\exists y_1, ..., y_k,$ *such that* $G(x_1, ..., x_n, y_1, ..., y_k) \neq F(x_1, ..., x_n)$.

When computing $G(X,Y)$ deterministically an algorithm will output either an answer that $G(X, Y) = b$ ($b \in \{0, 1\}$) or indefinite answer "don't know". If some Boolean value b is retrieved during calculation, it implies that $F(X) = b$.

Definition 5.4 *The **nondeterministic query complexity of the function F(X) with the fixed helper function G(X,Y)** is denoted with $ND_G(F)$ and is equal to the deterministic complexity of the G(X,Y): $ND_G(F) = D(G)$.*

An additional restriction on the deterministic query algorithm for the helper function $G(X,Y)$ is that after computing this function deterministically it should be possible to re-calculate or verify the value of $F(X)$ independently, using variable values extracted from the black box during the calculation of $G(X,Y)$.

Definition 5.5 *The **nondeterministic query complexity of the function F(X)** is denoted with ND(F) and is equal to the minimal nondeterministic query complexity of the function F(X) over all possible fixed helper functions G(X,Y): $ND(F) = \min_{G(X,Y)} ND_G(F)$.*

5.2.2 Computing the Fano Plane Function

In this section, the author demonstrates the alternative nondeterministic query model in action and shows that a gap between deterministic and nondeterministic query complexity for a certain Boolean function can be large.

5.2.2.1 Definition of the Fano Plane Boolean Function

The Fano plane is the two-dimensional finite projective plane with the least number of points and lines [43]. This plane has seven points and seven lines with three points on every line. Fano plane has many applications including factoring integers via quadratic forms [44]. The author defines a 7-variable Boolean function based on the structure of the Fano plane. Each vertex of the Fano plane is labeled by a variable number x_i. Fig. 5.10 represents a variant of variables assignment and this fixed definition will be used in the remainder of this section.

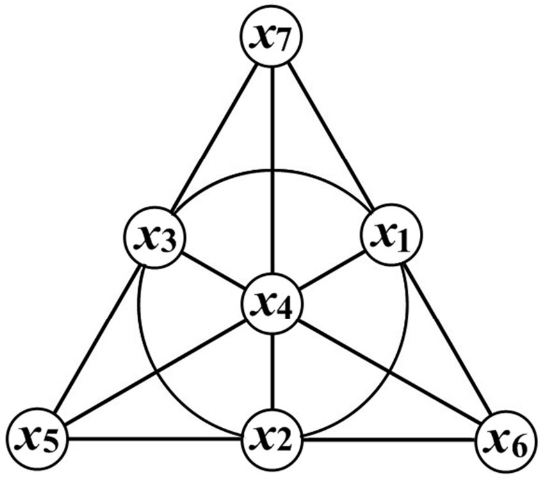


Fig. 5.10 Fano plane with vertices labeled by function *FANO*(*X*) variables

Definition 5.6 *A line of the Fano plane with Boolean values assigned to vertices is called **constant** if all vertices in a line have the same Boolean value assigned.*

There are two important properties of the Fano plane with Boolean values assigned to vertices. For any variable values assignment ($x_i \in \{0, 1\}$) there always is a constant line.

Second property is that for any variable values assignment there cannot be two constant lines assigned with the different Boolean value at the same time. These two properties allow to define a Boolean function based on the Fano plane.

Definition 5.7 *Boolean function* ***FANO(x_1, ..., x_7)*** *is defined as follows. For an arbitrary input X=(x_1, ..., x_7) find a constant line in the Fano plane. Value of the FANO(x_1, ..., x_7) function equals Boolean value assigned to vertices in that constant line.*

An example of *FANO*(*X*) function value assignment is illustrated in Fig. 5.11.

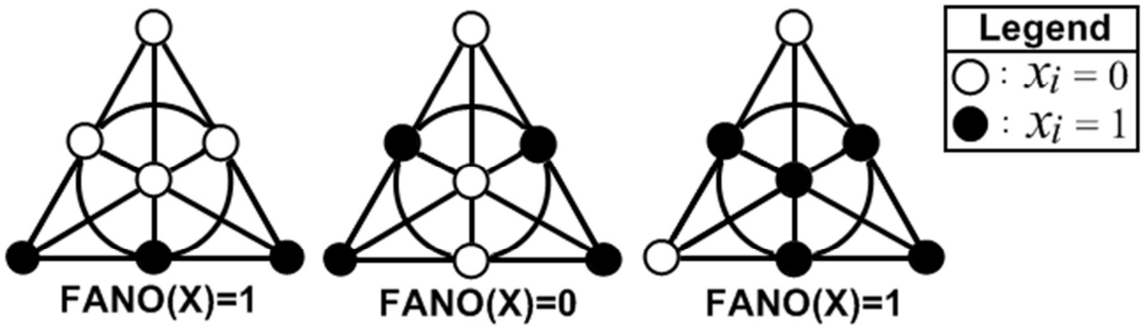


Fig. 5.11 Illustration of *FANO*(*X*) Boolean function value assignment

The Fano plane Boolean function can be represented also with a logical formula:

$$FANO(x_1, ..., x_7) = (x_1 \wedge x_2 \wedge x_3) \vee (x_5 \wedge x_3 \wedge x_7) \vee (x_7 \wedge x_1 \wedge x_6) \vee$$
$$\vee (x_6 \wedge x_2 \wedge x_5) \vee (x_3 \wedge x_4 \wedge x_6) \vee (x_1 \wedge x_4 \wedge x_5) \vee (x_2 \wedge x_4 \wedge x_7).$$

5.2.2.2 Deterministic Complexity of the Fano Plane Boolean Function

To determine *FANO*(*X*) function value using the deterministic decision tree it is necessary to query all variables.

Theorem 5.12 *Deterministic complexity of the Fano plane Boolean function is equal to the number of function variables: D(FANO)=7 .*

Proof. To prove this lower bound the author uses a kind of *adversary* method. The computation is examined as a game between an arbitrary algorithm and an adversary, which is playing against that algorithm. Algorithm is querying variables in order to determine function value, while adversary is providing variable values trying to use a strategy that forces an algorithm to query all variables. The author considers all possible scenarios and shows that for any deterministic algorithm always exists such adversary strategy that forces to query all variables. In other words, it means that for any fixed algorithm such input *X* always exists on which all variables must be queried.

First of all, the author defines a *winning game state* for an algorithm. In such a state next query will give function value for sure, either 0 or 1. We say that variable is *open* if its value is already known to an algorithm, otherwise variable is *closed*.

State is *winning* if there are two crossing lines, where crossing point is closed, two points on one line are open as “0”, while two points on other line are open as “1”. Query about crossing point surely will be the last. See Fig. 5.12 for an example of winning state.

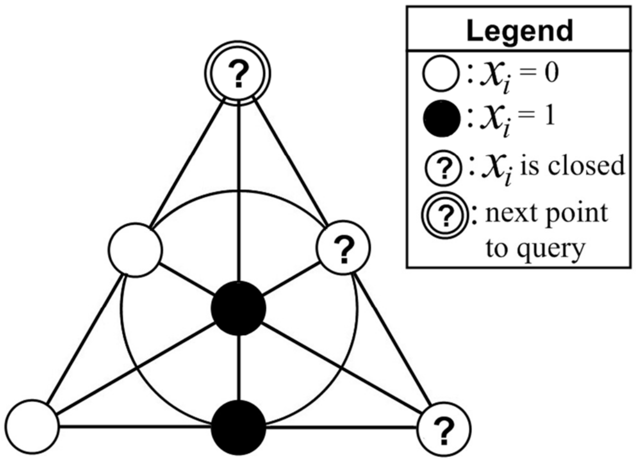


Fig. 5.12 Example of winning game state for an algorithm

Now let us examine all possible cases. Because of the symmetry of the Fano plane, number of cases to consider is rather small. After first three queries, only two different in essence states are possible: (1) three open points are located on one line, (2) three open points form a triangle. In both cases adversary strategy is to open two “0” and one “1”. See example in Fig. 5.13.

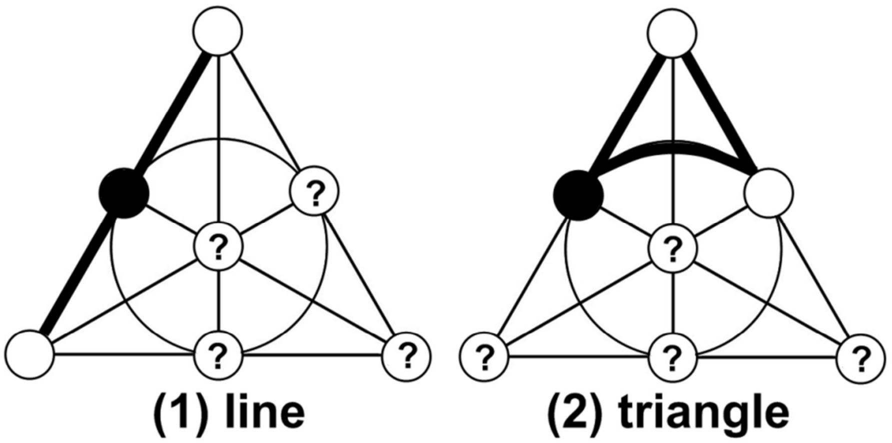


Fig. 5.13 Example of two possible in essence distinguishable states after three queries

Case 1 (line). All four choices of variable for the fourth query are equivalent because of the symmetry. Adversary strategy in any case is to open “1”. After such fourth query there remain four potential constant lines:

- L1, with two “1” already open;
- L2, with one “0” already open;
- L3, with one “0” already open;
- L4, with one “1” already open.

See example in Fig. 5.14.

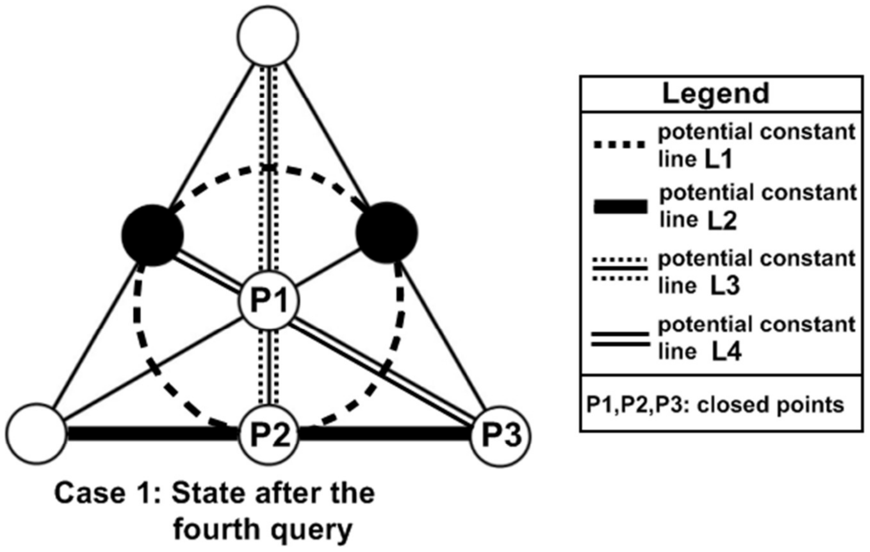

Legend
potential constant line L1
potential constant line L2
potential constant line L3
potential constant line L4
P1,P2,P3: closed points
P1
P2
P3
Case 1: State after the fourth query

Fig. 5.14 Case 1: illustration of potential constant lines and closed points after the fourth query

At the same time there remain three closed points:

- P1 ∈ L2, L4;
- P2 ∈ L1, L2, L3;
- P3 ∈ L3, L4.

Let us examine three possible cases for an algorithm to choose variable for the fifth query.

Case 1.1. If algorithm chooses point P1 for the fifth query, adversary strategy is to open "1". Result - there is no winning state and thus adversary is able to force algorithm to query all seven variables. See Fig. 5.15.A.

Case 1.2. If algorithm chooses point P2 for the fifth query, adversary strategy is to open "0". There again is no winning state. So, adversary will be able to give non-finishing variable value to any next query and seventh query will be required. See Fig. 5.15.B.

Case 1.3. For the last remaining option, point P3, adversary strategy is to open "0". No winning state, so adversary again is able to force algorithm to query all seven variables. See Fig. 5.15.C.

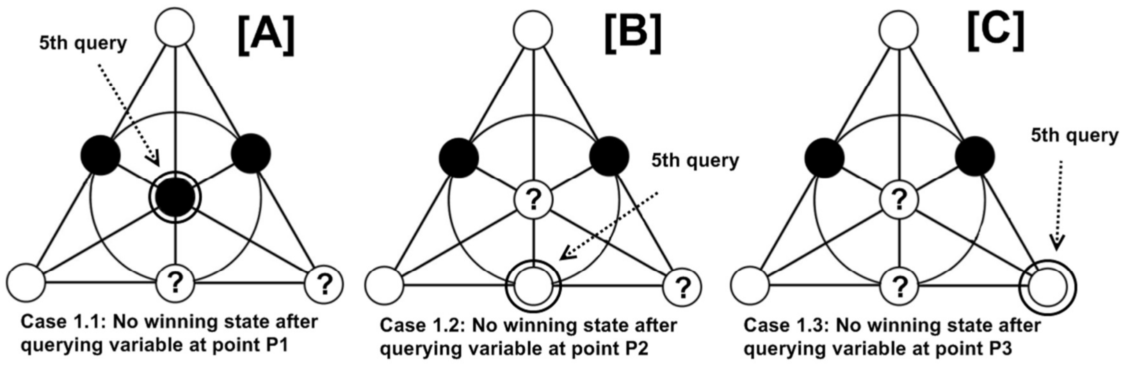

[A]
5th query
?
?
Case 1.1: No winning state after querying variable at point P1
[B]
5th query
?
?
Case 1.2: No winning state after querying variable at point P2
[C]
5th query
?
?
Case 1.3: No winning state after querying variable at point P3

Fig. 5.15 Cases 1.1, 1.2, 1.3: illustration of possible states after fifth query

Case 2 (triangle). Four closed points that remain after third query can be divided to two sets. Let us examine two different cases (2.1 and 2.2) based on this separation.

Case 2.1. There are three points (e.g. lower line in Fig. 5.13.1), opening of which after fourth query will bring a game to the state equivalent to that described in Case 1[5]. Adversary strategy for such fourth query is to open “1”. As shown above, in such situation adversary is able to force querying all seven variables.

Case 2.2. There is one point (e.g. central point in Fig. 5.13.2), after opening of which one line of the Fano plane will remain still fully closed. Adversary strategy is to open “0” for such fourth query. Otherwise, winning state would appear. Next, for any algorithm choice for fifth query adversary strategy is to open “1”. There will be no winning state, so, adversary is able to force algorithm to query all seven variables. See example in Fig. 5.16

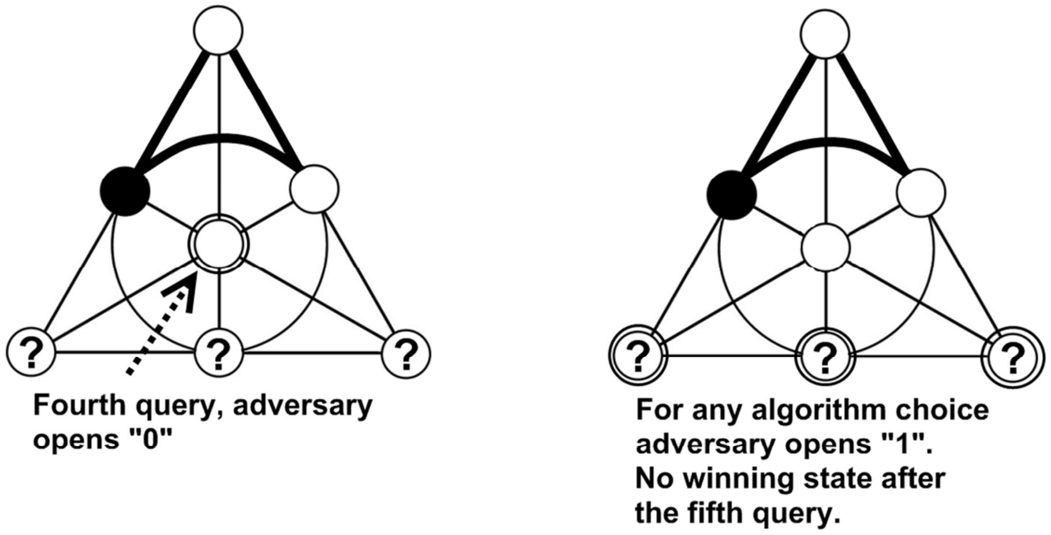

?
?
?
Fourth query, adversary opens "0"
?
?
?
For any algorithm choice adversary opens "1". No winning state after the fifth query.

Fig. 5.16 Case 2.2: illustration of possible states after fourth and fifth queries

All possible cases are analyzed and it is demonstrated that for any fixed deterministic algorithm such input X always exists on which all seven variables must be queried.

5.2.2.3 Nondeterministic Algorithm for the Fano Plane Boolean Function

In this subsection, the author demonstrates the computation of the Fano plane Boolean function in proposed alternative nondeterministic query model.

The first step is to define helper function $G_{FANO}(X, Y)$. The author adds three helper variables, so in total there are ten variables:

$$G_{FANO}(x_1, \ldots, x_7, y_1, y_2, y_3).$$

Subsequently, a binary sequence number is assigned to each line of the Fano plane. This assignment can be arbitrary, variant presented in Table 5.3 will be used further.

[5] The roles of ”0” and ”1” are interchanged in some cases.

Table 5.3 Binary numbering of the Fano plane lines

Line variables	Line number	Line variables	Line number
x_1, x_2, x_3	000	x_3, x_4, x_6	100
x_5, x_3, x_7	001	x_1, x_4, x_5	101
x_7, x_1, x_6	010	x_2, x_4, x_7	110
x_6, x_2, x_5	011		

Variables of the partial helper function $G_{FANO}(X,Y)$ are divided to two subsets. Variables of the X subset represent variable assignment of original $FANO(X)$ Boolean function. Variables of the Y subset represent the Fano plane line binary number. Since there is no line numbered with "111", function G_{FANO} is not defined for all inputs where $y_1 = y_2 = y_3 = 1$.

Definition 5.8 *Partial Boolean function $G_{FANO}(X, Y)$ is defined as:*

- *$G_{FANO}(X, y_1, y_2, y_3) = 1$, if the Fano plane line numbered with $y_1y_2y_3$ is constant and variables on that line are assigned Boolean value 1;*
- *$G_{FANO}(X, y_1, y_2, y_3) = 0$, if the Fano plane line numbered with $y_1y_2y_3$ is constant and variables on that line are assigned Boolean value 0;*
- *otherwise, function value is not defined.*

For the illustration purpose, partial truth table for inputs X = 0000001 and X=0110110 is given in Table 5.4.

Table 5.4 Partial truth table for $G_{FANO}(X,Y)$

$X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	$Y = (y_1, y_2, y_3)$	$FANO(X)$	$G_{FANO}(X,Y)$
0000001	000	0	0
	001		not defined
	010		not defined
	011		0
	100		0
	101		0
	110		not defined
	111		not defined
…	…	…	…
0110110	000	1	not defined
	001		not defined
	010		not defined
	011		1
	100		not defined
	101		not defined
	110		not defined
	111		not defined

Now, when the helper function $G_{FANO}(X, Y)$ is defined, the next step is to design an algorithm for computing this function. The author would like to remind that this algorithm has to be deterministic and variable values retrieved during computation process shall give us enough information to verify the value of $FANO(X)$ function independently.

The deterministic decision tree algorithm for computing $G_{FANO}(X,Y)$ consists of the following steps:

1. Sequentially query variables y_1, y_2 and y_3;
2. Find the Fano plane line numbered with $y_1y_2y_3$ and sequentially query all three variables composing this line;
3. If all variable values retrieved in the second step are equal - output this Boolean value. Otherwise, output "not defined".

All three variables composing the line must be queried in the second step because of the restriction that retrieved information should allow us to verify the value of $FANO(X)$ function. To perform such verification retrieved variable values are simply substituted to the logical formula of $FANO(X)$ and it is ensured that it evaluates to the correct value.

Theorem 5.13 *Nondeterministic query complexity of the Fano plane Boolean function FANO(X) with the fixed helper function $G_{FANO}(X, Y)$ is*

$$ND_{G_{FANO}}(FANO) = 6.$$

Proof. Algorithm described above performs three queries to the black box in the first step and next three queries to the black box in the second step.

5.2.2.4 Complexity of the Recursive Fano Plane Function

Finally, the author shows that a gap between deterministic and nondeterministic query complexity can be asymptotically large.

The definition of the Fano plane function can be applied recursively.

Definition 5.9. ***Recursive Boolean function*** *$FANO^i$ is defined as follows:*

- $FANO^1(X^1) = FANO(x_1, x_2, ..., x_7)$;
- $FANO^i(X^i) = FANO^1(FANO^{i-1}(X^{i-1}{}_1), FANO^{i-1}(X^{i-1}{}_2), ..., FANO^{i-1}(X^{i-1}{}_7))$, *where* $X^i = X_1^{i-1}X_2^{i-1}...X_7^{i-1}$.

Recursively defined Boolean function $FANO_N(X)$ has 7^N variables.

Theorem 5.14 *Deterministic decision tree complexity of the recursive Fano plane Boolean function $FANO_N$ is $D(FANO_N) = 7^N$.*

Proof. Since $D(FANO_1) = 7$, on each recursion level it is necessary to know values of all seven sub-functions. So, on the last level, the total number of variables to be queried is equal to 7^N. □

In a nondeterministic helper function $G_{FANO_N}(X,Y)$ for a recursive Fano plane function $FANO_N$ three additional helper variables are defined for each recursion level. These helper variables indicate which three sub-functions need to be computed in order to determine function value. The total complexity of the deterministic algorithm for the helper function evaluates to $O(3^N)$.

Theorem 5.15 *Nondeterministic query complexity of the recursive Fano plane Boolean function* $FANO^N$ *with the fixed helper function is* $ND_G(FANO^N) = O(3^N)$.

Proof. For each recursion level, first, algorithm queries three helper variables to determine three sub-functions that compose a line. Subsequently, algorithm goes one level deeper and calculates value of each sub-function. Calculation of number of queries is presented in Table 5.5.

Table 5.5 Calculation of nondeterministic query complexity for different recursion levels

Recursion level	Number of queries[6]
i=1	$ND_G(FANO^1) = 3_y + 3_x = 6$
i=2	$ND_G(FANO^2) = 3_y + 3\cdot(3_y + 3_x) = 3_y + 9_y + 9_x = 21$
i=3	$ND_G(FANO^3) = 3_y + 3\cdot(3_y + 3\cdot(3_y + 3_x)) = 3_y + 9_y + 27_y + 27_x = 66$
…	…
i=N	$ND_G(FANO^N) = 3_y + 3\cdot ND_G(FANO^{N-1}) = \left(\sum_{i=1}^{N} 3_y{}^i\right) + 3_x{}^N$

Finally, partial sum formula is used for the term $\sum_{i=1}^{N} 3^i$ and the complexity estimation is derived for the case of recursion level $i = N$ equal to $ND_G(FANO^N) = O(3^N)$:

$$ND_G(FANO^N) = \left(\sum_{i=1}^{N} 3^i\right) + 3^N = \frac{3}{2}\left(3^N - 1\right) + 3^N = \frac{5}{2}3^N - \frac{3}{2} = O(3^N) .$$

□

5.2.3 Conclusion and Open Problems

The author introduced an alternative definition of a nondeterministic query model. The main difference from traditional nondeterministic model is that nondeterministic behavior is not obtained free of charge, but additional queries must be spent to obtain nondeterministic help. The model proposed was demonstrated on example of computing the Fano plane Boolean function. When the definition of the function is applied recursively, a gap between deterministic and nondeterministic query complexity is 7^N versus $O(3^N)$.

Future work is to develop and improve the nondeterministic query model introduced in this paper. The scope of further investigation is very wide, from designing algorithms for certain problems in this model to performing a detailed complexity analysis and comparison to other computational models. Considering Boolean function based on projective finite geometries, similar to the Fano plane function, seems to be a promising direction for searching of interesting examples. The most important further step is to define a quantum counterpart of the alternative nondeterministic query model and to investigate its properties.

[6] Subscript s near each number i_s (e.g. 3_x or 3_y) indicates that variable from subset S (e.g. X or Y) is queried.

6 Conclusion

In the present thesis, the problems of quantum query algorithm design and complexity are investigated. Different types of quantum query algorithms are examined, such as exact, bounded-error and nondeterministic.

In the first part of the thesis, quantum query algorithms for computing Boolean functions are investigated.

Regarding exact quantum algorithms (for which the probability of a correct result is invariably $p = 1$), there is a long-standing open question and challenge: to establish whether a larger separation than N versus $2N$ can be achieved between the classical deterministic and the quantum exact query complexity for a total function? Even the design of examples with an N versus $2N$ complexity gap is known to be a complex task. The majority of such examples are directly based on the involvement of an *XOR* operation in the definition of a computable function. In the course of the present research study, efforts were made to improve and develop universal techniques for designing exact quantum query algorithms. To simplify the process of generation and verification of quantum algorithms, a Wolfram *Mathematica* software program was developed. Using this tool, two examples of Boolean functions were generated with a small number of variables, for which the complexity of the quantum exact query algorithm is lower than the complexity of the classical deterministic query algorithm. Subsequently, quantum algorithm transformation methods were proposed, providing for a significant enlargement of the set of efficiently computable Boolean functions. Transformation methods were successfully applied to quantum algorithms for two basic Boolean functions. The approach demonstrated may be applied to already familiar and new exact quantum query algorithms. Additionally, an exact quantum query algorithm for the problem of repetition code verification was presented. In this example, a gap of N versus $2N$ between quantum and classical complexity is achieved. Finally, a technique for generation of examples with an N versus $2N$ complexity gap was demonstrated. In the course of the study, the answer to the most important question was still not found: namely, is it possible to achieve a larger gap than N versus $2N$? However, by using the presented techniques, the task of designing efficient exact quantum query algorithms is significantly simplified.

Regarding bounded-error quantum query algorithms, an algorithm for computing conjunctions, i.e., the Boolean function $AND(x_1,\dots,x_N)$, was presented. This is a basic and widely applicable function, which requires an efficient algorithm. The quantum algorithm presented computes the conjunction of two bits through a single query with the correct answer probability $p = 4/5$. The approach was extended by formulating a general method for computing the conjunction of two Boolean functions with the same probability and the number of queries equal to $\max(Q_E(f_1), Q_E(f_2))$.

The second part of the thesis is devoted to computing multivalued functions in a query model. In author's opinion, this section contains the most interesting and important results. The author proposed three types of query algorithms for computing multifunctions: definite, randomly distributed and uniformly distributed query algorithms. Since quantum algorithms actively employ the power of parallelism and computing in superposition, they are naturally well suited for computing multifunctions in a distributed manner. The author presented three examples of computing multifunctions in classical and quantum versions of the query model. The most important result achieved is an N versus $3N$ separation between the quantum and the classical complexity of the multifunctions examined. There is no error probability, so in some sense it is comparable and exceeds the largest known N versus $2N$ gap between the quantum exact and the classical deterministic complexity for the total Boolean function. Another important point is that the definition of multifunction is not based on an *XOR* operation. Finally, the author described universal methods to generalize finite multifunctions to families of multifunctions with an input of general size.

In the third part of the thesis, the author examined nondeterministic query algorithms. Regarding the traditional quantum query model, the author introduced a new notion of a dual nondeterministic quantum query algorithm. This type of algorithms was studied, and the discovered properties were applied to design efficient algorithms. For the Boolean function $Control_N$, a dual nondeterministic quantum algorithm was designed for computing it with two queries for all N, while classically N queries are required. Investigating the traditional nondeterministic query model the author realized that it is counter-intuitive in some sense and does not employ the full power of nondeterminism. As an alternative, a different definition of a nondeterministic query model was introduced. The model was demonstrated on the basis of an example of computing the Fano plane Boolean function and obtaining a 7^N versus $O(3^N)$ gap between the deterministic and the nondeterministic query complexity.

The results presented are mostly of theoretical importance; however, in the future it will be possible to implement the algorithm designing methods proposed in a quantum computer, therefore achieving of practical benefits is also feasible.

The goals of the research are achieved on the whole; however, further improvements are possible by continuing the study in the following directions:

- *Exact quantum algorithms*. It is still unclear whether it is possible or not to build a quantum algorithm more than two times faster than the best possible classical deterministic algorithm. The author's idea for further work is to apply the combinatorial approach for algorithm design and to enlarge the size of the quantum system.
- *Computing multifunctions in the query model*. In author's opinion, this study direction is the most promising. To be able to build even more efficient algorithms, it is necessary to perform an exhaustive analysis of the properties of functions and algorithms discovered during the research study. A very important action line for further studies is development of convenient and efficient methods for proving the classical complexity lower bounds for multifunctions.
- *Nondeterministic query algorithm*s. The most important further step is to define a quantum counterpart of the alternative nondeterministic query model and to investigate its properties.

Bibliography

1. Shor, P. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5), pp.1484-1509, 1997
2. Grover, L. A fast quantum mechanical algorithm for database search. *Proc. of 28th STOC'96*, pp.212-219, 1996
3. DiCarlo, L., Chow, J., Gambetta, J., et al. Demonstration of two-qubit algorithms with a superconducting quantum processor. *Nature 460,* 460, pp.240-244, 2009
4. Politi, A., Matthews, J. C. F., O'Brien, J. L. Shor's Quantum Factoring Algorithm on a Photonic Chip. *Science*, 5945, vol. 325, p.1221, 2009
5. Johnson, M. W., Amin, M. H. S., Gildert, S., et al. Quantum annealing with manufactured spins. *Nature*, vol. 473, pp.194-198, 2011
6. Hiskett, P. A., Rosenberg, D., Peterson, C. G. Long-distance quantum key distribution in optical fibre. *New J. Phys.*, 193, vol. 8, 2006
7. Dixon, A. R., Dynes, J. F., Shields, A. J. Gigahertz decoy quantum key distribution with 1 Mbit/s secure key rate. *Optics Express*, 23, vol. 16, 2008
8. Jin, Xian-Min, Ren, Ji-Gang, Yang, Bin Experimental free-space quantum teleportation. *Nature Photonics*, 4, pp.376-381, 2010
9. Lee, N., Benichi, H., Takeno, Y., et al. Teleportation of Nonclassical Wave Packets of Light. *Science*, 6027, vol. 332, pp.330-333, 2011
10. ID Quantique SA - Network Encryption, Random Number Generators, Photon Counting. [online] [accessed: 31.10.2011]. Available: http://www.idquantique.com/
11. D-Wave Systems sells its first Quantum Computing System to Lockheed Martin Corporation. [online] [accessed: 31.10.2011].

    Available: http://www.dwavesys.com/en/pressreleases.html#lm_2011
12. Buhrman, H., de Wolf, R. Complexity Measures and Decision Tree Complexity: A Survey. *Theoretical Computer Science,* v. 288(1), pp.21–43, 2002
13. Nielsen, M., Chuang, I. Quantum Computation and Quantum Information. *Cambridge University Press*, 2000
14. de Wolf, R. Quantum Computing and Communication Complexity. *University of Amsterdam*, 2001
15. Ambainis, A., de Wolf, R. Average-case quantum query complexity. *Journal of Physics A 34*, pp.6741-6754, 2001
16. Ambainis, A. Quantum query algorithms and lower bounds (survey article). *Proc. of FOTFS III, Trends on Logic*, vol. 23, pp.15-32, 2004
17. Ambainis, A. Polynomial degree vs. quantum query complexity. *Journal of Computer and System Sciences 72*, pp.220-238, 2006

18. Kravcevs, V. Quantum algorithm complexity. *Doctoral Thesis*, *University of Latvia*, 2008

19. Ščeguļnaja-Dubrovska, O. Models of Quantum Computation. *Doctoral Thesis*, *University of Latvia*, 2010

20. Ambainis, A., Childs, A. M., Reichardt, B. W., et al. Any AND-OR Formula of Size N Can Be Evaluated in Time $N^(1/2+o(1))$ on a Quantum Computer. *SIAM J. Comput.*, vol. 39, pp.2513-2530, 2010

21. Freivalds, R. Query Algorithms. *Unpublished paper*

22. Feynman, R. Simulating Physics with Computers. *International Journal of Theoretical Physics*, No. 6/7, vol. 21, 1982

23. Deutsch, D. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. of the Royal Society of London A 400*, 1985

24. Childs, A. M., Cleve, R., Deotto, E., et al Exponential algorithmic speedup by quantum walk. *35th ACM Symposium on Theory of Computing*, pp.59-68, 2003

25. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M. Quantum algorithms revisited. *Proc. of the Royal Society of London*, vol. A 454, pp.339–354, 1998

26. Gruska, J. Quantum Computing. *McGraw-Hill*, 1999

27. Kaye, P., Laflamme, R., Mosca, M. An Introduction to Quantum Computing. *Oxford*, 2007

28. Deutsch, D., Jozsa, R. Rapid solutions of problems by quantum computation. *Proc. of the Royal Society of London*, vol. A 439, pp.553-558, 1992

29. Simon, I. String matching algorithms and automata. *Lecture Notes in Computer Science*, vol. 814, pp.386-395, 1994

30. Freivalds, R., Iwama, K. Quantum Queries on Permutations with a Promise. *Implementation and Application of Automata, Lecture Notes in Computer Science*, vol. 5642/2009, pp.208-216, 2009

31. Agadžanjans, R. Complexity of Quantum Query Algorithms. *Doctoral Thesis*, *University of Latvia*, 2010

32. Lace, L. Quantum Query Algorithms. *Doctoral Thesis, University of Latvia*, 2008,

33. Lace, L. Enlarging gap between quantum and deterministic query complexities. *Proc. of Baltic DB&IS*, Riga, Latvia, vol. 2, pp.81-91, 2004

34. Wolfram Mathematica computational software. [online] [accessed: 31.10.2011] Available: http://www.wolfram.com/mathematica/.

35. Cover, T., Thomas, J. Elements of Information Theory. *Wiley-Interscience*, 1991

36. Dubrovska, A. Application of quantum computing for efficient algorithm construction. *Master's Thesis, University of Latvia*, 2007

37. Kerenidis, R., de Wolf, R. Exponential Lower Bound for 2-Query Locally Decodable Codes via a Quantum Argument. *Journal of Computer and System Sciences*, pp.395-420, 2004

38. Kushilevitz, E., Nisan, N. Communication complexity. *Cambridge University* Press, 1997

39. Rabin, M., Scott, D. Finite automata and their decision. *IBM. Journal of Research and Development*, 3:2, pp.114-125, 1959

40. Floyd, R. Nondeterministic Algorithms. *Journal of the ACM (JACM)*, 4, vol. 14, pp.636-644, 1967

41. Hopcroft, J., Ullman, J.D. Formal Languages and their Relation to Automata. *Addison-Wesley, Reading, MA*, 1969.

42. de Wolf, R. Nondeterministic Quantum Query and Quantum Communication Complexities. *SIAM Journal on Computing*, 32(3), pp.681-699, 2003

43. Weisstein, E. Fano Plane. *From MathWorld - A Wolfram Web Resource.* [online] [accessed: 31.10.2011]. Available: http://mathworld.wolfram.com/FanoPlane.html.

44. Lehmer, D., Lehmer, E. A New Factorization Technique Using Quadratic Forms. *Mathematics of Computation*, 28, 126, pp.625-635, 1974

Appendix 1: Wolfram *Mathematica* program

The following simple program was developed for verification and generation of quantum algorithms. Quantum algorithm for $EQUALITY_3$ from Section 3.1.1.1 is simulated by this concrete program. However, it is very easy to simulate different algorithms by changing algorithm parameters in program code.

Program code

```
(* Define all intermediate fixed unitary transformations *)
U0 = {
   {1/2, 1/2, 1/2, 1/2},
   {1/2, -1/2, 1/2, -1/2},
   {1/2, 1/2, -1/2, -1/2},
   {1/2, -1/2, -1/2, 1/2}};
U1 = {
  {1, 0, 0, 0},
  {0, 1/√2, 1/√2, 0},
  {0, 1/√2, -1/√2, 0},
  {0, 0, 0, 1}};
U2 = {
   {1/2, 1/2, 1/2, 1/2},
   {1/√2, 0, 0, -1/√2},
   {0, -1/√2, 1/√2, 0},
   {1/2, -1/2, -1/2, 1/2}};

(* For convenience, print matrices *)
Print[U0 // MatrixForm];
Print[U1 // MatrixForm];
Print[U2 // MatrixForm];

(* Verify that all defined transformations are unitary *)
UnitaryQ[m_List?MatrixQ] := (Conjugate@Transpose@m.m == IdentityMatrix@Length@m);
Print["U0 is unitary: ", UnitaryQ[U0]];
Print["U1 is unitary: ", UnitaryQ[U1]];
Print["U2 is unitary: ", UnitaryQ[U2]];

(* Additional transformations *)
U2 = Transpose[U2];

(* Define initial quantum state *)
START = {1, 0, 0, 0};

(* Define a quantum measurement *)
MEASUREMENT = {{1}, {0}, {0}, {0}};
```

```
Print["------------------------------"];
Print["Input X | F(X) |  Final state"];
Print["------------------------------"];

(* Start algorithm verification, execute algorithm on all possible inputs
    Queries are defined inside.
 *)
For[X1 = 0, X1 ≤ 1, X1++,
  For[X2 = 0, X2 ≤ 1, X2++,
    For[X3 = 0, X3 ≤ 1, X3++,
      QUERY1 = {
        {(-1)^X1, 0, 0, 0},
        {0, (-1)^X2, 0, 0},
        {0, 0, (-1)^X1, 0},
        {0, 0, 0, (-1)^X2}};
      QUERY2 = {
        {(-1)^X3, 0, 0, 0},
        {0, (-1)^X1, 0, 0},
        {0, 0, (-1)^X2, 0},
        {0, 0, 0, (-1)^X3}};
      RESULT = U0.START;
      RESULT = U1.QUERY1.RESULT;
      RESULT = U2.QUERY2.RESULT;
      FINAL = RESULT.MEASUREMENT;
      F = FINAL[[1]]^2;
      Print["    ", X1, X2, X3, " | ", F, "    |  ", RESULT];
     ];
   ];
 ];
```

Program output

$$U0=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

$$U1=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$U2=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & 0 & 0 & -\frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

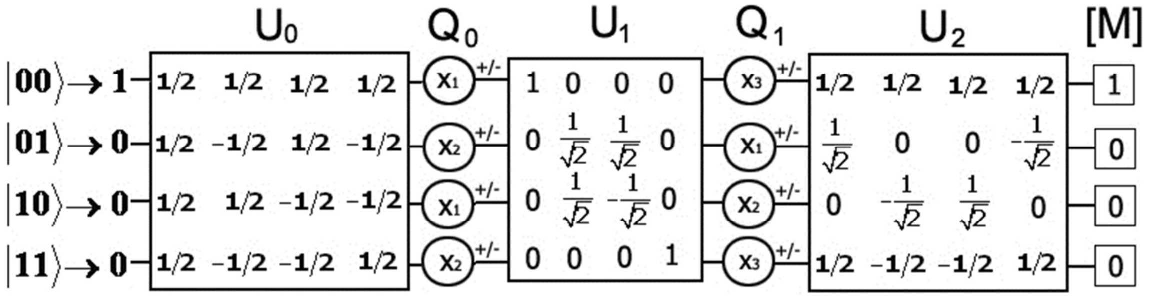
```
U0 is unitary: True

U1 is unitary: True

U2 is unitary: True

------------------------------

Input X | F(X) |  Final state

------------------------------

    000 | 1    |  {1, 0, 0, 0}

    001 | 0    |  {0, 0, 0, -1}

    010 | 0    |  {0, 0, 1, 0}

    011 | 0    |  {0, -1, 0, 0}

    100 | 0    |  {0, -1, 0, 0}

    101 | 0    |  {0, 0, 1, 0}

    110 | 0    |  {0, 0, 0, -1}

    111 | 1    |  {1, 0, 0, 0}
```

Appendix 2: Computation of the quantum query algorithm for *EQUALITY₃*

The quantum query algorithm for computing Boolean function

$$EQUALITY_3(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)$$

with two queries has been presented in Section 3.1.1.1.



The following table describes the computation process for each input.

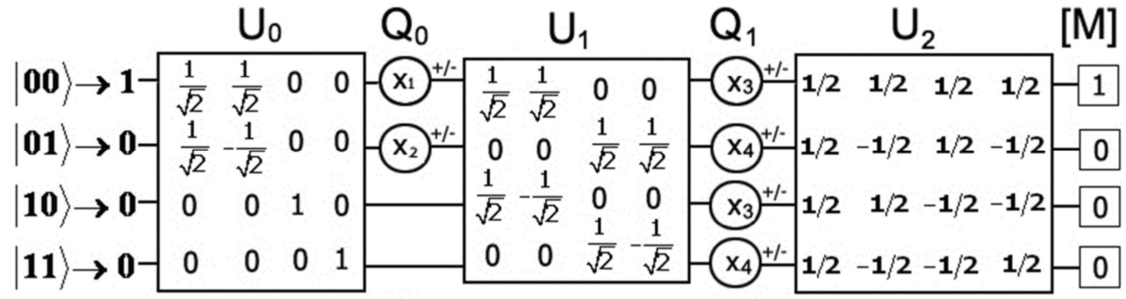
X	After $\langle\vec{0}\vert U_0Q_0$	After $\langle\vec{0}\vert U_0Q_0U_1$	After $\langle\vec{0}\vert U_0Q_0U_1Q_1$	Final state	Result
000	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{\sqrt{2}},0,\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{\sqrt{2}},0,\frac{1}{2}\right)^T$	$(1,0,0,0)^T$	**1**
001	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{\sqrt{2}},0,\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},\frac{1}{\sqrt{2}},0,-\frac{1}{2}\right)^T$	$(0,0,0,-1)^T$	**0**
010	$\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},0,-\frac{1}{\sqrt{2}},-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},0,\frac{1}{\sqrt{2}},-\frac{1}{2}\right)^T$	$(0,0,1,0)^T$	**0**
011	$\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},0,-\frac{1}{\sqrt{2}},-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},0,\frac{1}{\sqrt{2}},\frac{1}{2}\right)^T$	$(0,-1,0,0)^T$	**0**
100	$\left(-\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},0,\frac{1}{\sqrt{2}},\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},0,\frac{1}{\sqrt{2}},\frac{1}{2}\right)^T$	$(0,-1,0,0)^T$	**0**
101	$\left(-\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},0,\frac{1}{\sqrt{2}},\frac{1}{2}\right)^T$	$\left(\frac{1}{2},0,\frac{1}{\sqrt{2}},-\frac{1}{2}\right)^T$	$(0,0,1,0)^T$	**0**
110	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},-\frac{1}{\sqrt{2}},0,-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},\frac{1}{\sqrt{2}},0,-\frac{1}{2}\right)^T$	$(0,0,0,-1)^T$	**0**
111	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},-\frac{1}{\sqrt{2}},0,-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{\sqrt{2}},0,\frac{1}{2}\right)^T$	$(1,0,0,0)^T$	**1**

Appendix 3: Computation of the quantum query algorithm for *PAIR_EQUALITY₄*

The quantum query algorithm for computing Boolean function

$$PAIR\_EQUALITY_4(x_1,x_2,x_3,x_4)=\neg(x_1\oplus x_2)\wedge\neg(x_3\oplus x_4)$$

with two queries has been presented in Section 3.1.1.2.



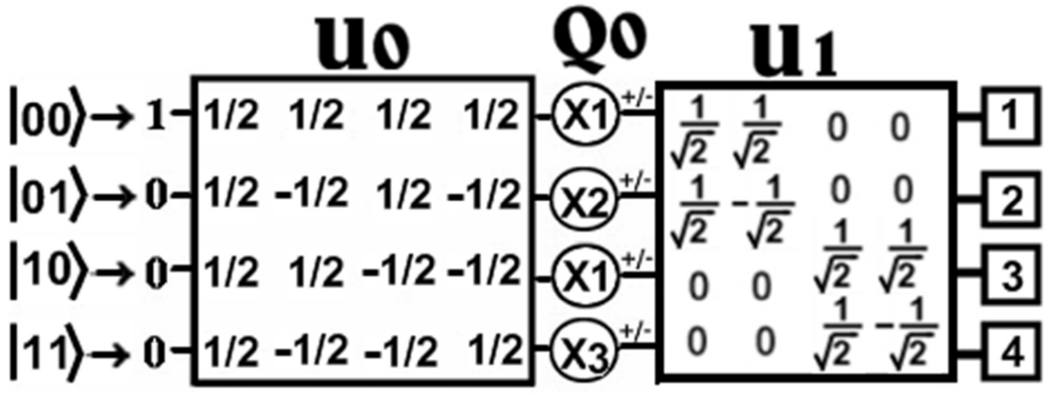
The following table describes the computation process for each input.

X	After $\langle\vec{0}\vert U_0Q_0$	After $\langle\vec{0}\vert U_0Q_0U_1$	After $\langle\vec{0}\vert U_0Q_0U_1Q_1$	Final state	Result
0000	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$(1,0,0,0)^T$	**1**
0001	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)^T$	$(0,1,0,0)^T$	**0**
0010	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)^T$	$(0,-1,0,0)^T$	**0**
0011	$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$(-1,0,0,0)^T$	**1**
0100	$\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$(0,0,1,0)^T$	**0**
0101	$\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},-\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)^T$	$(0,0,0,1)^T$	**0**
0110	$\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)^T$	$(0,0,0,-1)^T$	**0**
0111	$\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$(0,0,-1,0)^T$	**0**
1000	$\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$(0,0,-1,0)^T$	**0**
1001	$\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)^T$	$(0,0,0,-1)^T$	**0**

1010	$\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(\frac{1}{2},-\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)^T$	$(0,0,0,1)^T$	**0**
1011	$\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$(0,0,1,0)^T$	**0**
1100	$\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$(-1,0,0,0)^T$	**1**
1101	$\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(-\frac{1}{2},\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)^T$	$(0,-1,0,0)^T$	**0**
1110	$\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\right)^T$	$(0,1,0,0)^T$	**0**
1111	$\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}},0,0\right)^T$	$\left(-\frac{1}{2},-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\right)^T$	$\left(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)^T$	$(1,0,0,0)^T$	**1**

Appendix 4: Computation of the quantum query algorithm QM1

Quantum query algorithm QM1 for computing multivalued function M_1 in a uniformly distributed manner has been presented in Section 4.3.2.



The following table describes the computation process for each input.

X	State after the query	State before the measurement	Output
000	$\left(\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("1")=1/2 Pr("3")=1/2
001	$\left(\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("1")=1/2 Pr("4")=1/2
010	$\left(\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("2")=1/2 Pr("3")=1/2
011	$\left(\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("2")=1/2 Pr("4")=1/2
100	$\left(-\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("2")=1/2 Pr("4")=1/2
101	$\left(-\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("2")=1/2 Pr("3")=1/2
110	$\left(-\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("1")=1/2 Pr("4")=1/2
111	$\left(-\frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("1")=1/2 Pr("3")=1/2