UNIVERSITY OF LATVIA

FACULTY OF COMPUTING

TAISIJA MIŠČENKO-SLATENKOVA

QUANTUM QUERY ALGORITHMS

Doctoral Thesis

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

Scientific advisor:
Professor, Dr. habil. math.
Rūsiņš Mārtiņš Freivalds

Riga 2012

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 way of computation based on the laws of quantum mechanics. The main subject of this research is a quantum query algorithm, where we pursued a major aim to make quantum algorithm design as straightforward as possible.

This survey presents quantum query algorithms computing Boolean functions with a small number of queries and algorithms computing multivalued functions. Numerous quantum algorithms efficient for certain problems are described in the thesis. Bounded-error quantum algorithms are the most impressive, for example, a single-query algorithm for conjunction of two bits with the correct answer probability 9/10. Quantum versus classical algorithm complexity gap is discussed thorougly for each scope of functions.

The last part of the thesis is devoted to Boolean functions with low-degree representing polynomials. Approaches presented in this work allow to design a Boolean function with a large gap between the deterministic complexity and the degree of a representing polynomial.

Anotācija

Kvantu skaitļošana ir datorzinātnes apakšnozare, kas balstās uz kvantu mehānikas likumiem. Kvantu vaicājošais algoritms ir galvenais pētāmais objekts. Lielais darba mērķis ir padarīt kvantu algoritma konstruēšanu pēc iespējas vienkāršāku.

Pētījumā ir aprakstīti kvantu vaicājošie algoritmi, kas rēķina Būla funkcijas uzdodot maz vaicājumu, un tiek piedāvāti kvantu vaicājošie algoritmi daudzvērtīgu funkciju aprēķināšanai. Darbā ir aprakstīti vairāki efektīvi kvantu algoritmi konkrētu uzdevumu veikšanai. Paši nozīmīgākie ir kvantu vaicājošie algoritmi ar ierobežotu kļūdu, piemēram, piedāvāts algoritms divu bitu AND Būla funkcijai, kas izmanto vienīgu vaicājumu un izdod pareizu atbildi ar varbūtību 9/10. Katrai aprakstītajai funkciju kopai ir veikta pamatīga kvantu un klasiskās sarežģītības analīze.

Pēdējā pētījuma daļa ir veltīta Būla funkcijām ar zemas pakāpes polinomiem, kuri reprezentē dotās funkcijas. Darbā piedāvātie paņēmieni ļauj uzkonstruēt Būla funkcijas ar pietiekami lielu intervālu starp funkcijas determinētu sarežģītību un reprezentējošā polinoma pakāpi.

Preface

This thesis assembles the research performed by the author and reflected in the following publications:

1. 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
2. T. Mischenko-Slatenkova. *"Low Degree Boolean Functions: Application in Quantum Computation"*. Book of Abstracts "The 8th International Conference on Quantum Communication, Measurement and Computing", NICT and Tamagawa University, Japan, 2006, p. 220.
3. A. Dubrovska, T. Mischenko-Slatenkova, A. Rivosh. "*Quantum Query Algorithms for Certain Problems and Generalization of Algorithm Designing Techniques"*. Proc. of the Satelite Workshop of DLT 2007, ISBN 978-952-12-1921-4, Turku, Finland, pp. 65.-80, 2007
4. 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
5. 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
6. T. Mischenko-Slatenkova, A. Skuskovniks, A. Vasilieva, R. Tarasovs, R. Freivalds. *"Quantum queries on permutations"*. Accepted to publishing in proceeding of SOFSEM 2013.

The results of the thesis were presented at the following international conferences and workshops:

1. SOFSEM 2006, 32nd International Conference on Current Trends in Theory

and Practice of Computer Science. Merin, Czech Republic; 21-27 January 2006. Presentation "*Computing Boolean Functions: Exact Quantum Query Algorithms and Low Degree Polynomials*"

2. The Ninth Workshop on Quantum Information Processing (QIP 2006), Paris, France, January 16-20, 2006. Poster *"Computing Boolean Functions: Exact Quantum Query Algorithms and Low Degree Polynomials"*
3. The 8th International Conference on Quantum Communication (QCMC 2006), Measurement and Computing, 28th November - 3rd December, 2006, Tsukuba, Japan. Poster "*Low Degree Boolean Functions: Application in Quantum Computation*"
4. Developments in Language Theory (DLT 2007), Turku, Finland, July 3-6, 2007. Presentation "*Quantum Query Algorithms for Certain Problems and Generalization of Algorithm Designing Techniques*"
5. ACM International Conference on Computing Frontiers (ACM 2010), New York, USA, March 22 - 26, 2010. Presentation "*High Precision Quantum Query Algorithm for Computing AND-based Boolean Functions*"
6. IEEE International Conference on Fuzzy Systems (IEEE 2010), Barcelona, Spain, July 18-23, 2010. Presentation "*An Improved Quantum Query Algorithm for Computing AND Boolean Function*"
7. Unconventional Computation (UC 2010), Tokyo, Japan, June 21-25, 2010. Presentation "*Quantum Query Algorithms for Conjunctions"*
8. Computing Frontiers (CF 2010), Bertinoro, Italy, May 17-19, 2010. Presentation "*An Improved Quantum Query Algorithm for Computing AND Boolean Function"*
9. The Sixth Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2011), Madrid, Spain, May 24-26, 2011 Poster *"Quantum Query Model: Application to Computing Mathematical Relations"*

Acknowledgements

First I would like to thank my supervisor Rusins Freivalds for familiarizing me with quantum computation and for his support, straightforward advice and bright ideas. Secondly I want to thank professor Andris Ambainis and associated professor Juris Smotrovs for many useful scientific discussions, comments and quick responce throughout the period of my PhD-ship.

Furthermore I want to thank Alina Vasilieva - my smart, accurate and patient co-author of papers.

Finally, I gratefully thank my family for various things beyond the scope of this thesis.

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

Contents

1 Introduction

Quantum computing is an alternative way of computation based on the laws of quantum mechanics. We will be researching quantum query algorithms, which are proved to solve certain problems faster than their classical counterparts.

Given an explicit definition of some function and a black box oracle containing the values of the variables as an input of the query algorithm, the goal of the algorithm is to compute the function's value. The algorithm queries black box oracle about the values of variables. Queries are asked individually, and the result of any query influences the next query to be asked or the result to be output. [BW02] gives a formal definition of quantum query algorithm.

Complexity of a query algorithm is measured by the number of queries to a black box oracle on the worst case input. The classical version of this model is known as decision trees [BW02].

Shor's [Sh97] and Grover's [Gr01] fast algorithms are considered legendary classics, meanwhile a lot of successful quantum query algorithms for certain tasks have been developed recently ([Am04], [KLM07], [VM10], [ACRSZ10], [KN97]).
A. Ambainis ([Am02], [Am98]) developed powerful methods to prove lower bounds of quantum query complexity. A good reference is the survey by Buhrman and de Wolf [BW02].

One of the most important open problems in quantum computing is whether quantum algorithms could be more advantageous than probabilistic ones. Here we will concentrate on the case when these advantages are least expected, namely, when the computing device and the computation time are finite. We will discuss quantum versus classical query algorithm complexities: we are interested in classical and quantum algorithm complexity gap as large as possible for the same computational problem. The largest known gap between quantum exact and classical deterministic query algorithm complexity is N versus $2N$ ([BW02], [Am11]), and a long standing open question is whether *it is possible to enlarge this gap with no error allowed*. The first result with a 50% improvement for an exact algorithm was a quantum algorithm for XOR Boolean function presented in [CEMM98].

Meanwhile the best improvements are presented by quantum algorithms with certain error probability, making quantum computing a subject for numerous researches.

We classify query algorithms by the probability of returning a correct result:

- exact algorithms always output the correct result with probability p=1
- probabilistic algorithms may be of different types depending on the subject of the probability: either the result is correct with some probability, or there exists a probability for the outcome “unknown”, while the correct answer is returned precisely.

The quantum query model differs from the quantum circuit model ([Am04], [KLM07], [VM10], [ACRSZ10], [KN97]), and algorithm design techniques for this model are less developed. There are still no step-by-step instructions to design a quantum query algorithm for some computational problem – neither exact nor probabilistic. Given some function, it is a non-trivial task to design a quantum query algorithm for it, so the goal of this research is to find some new approaches and good examples of quantum query algorithms.

The research on quantum query algorithms will go in several directions:

- algorithms computing Boolean functions, especially, AND-function
- algorithms computing multivalued functions

In the first part of the research we will construct one-query bounded-error quantum query algorithms for three-, four- and six-bit equality functions and compare complexities between classical and quantum versions of these algorithms. We will construct one-query bounded-error quantum algorithms for the conjunction of two, three or five bits. We will propose a bounded-error quantum algorithm for the conjunction of $2n$ bits by asking n queries.

We will generalize some of the previously described algorithms and get new fast bounded-error quantum algorithms:

- a bounded-error algorithm with n quantum queries computing a function dependent on 3n arguments

- a bounded-error algorithm with n quantum queries computing a function dependent on 4n arguments

These algorithms are interesting on their own, but they also can be used as a basis for definition of another specific computational problem preserving the quantum algorithm complexity.

In the second part of the research we will focus on multivalued functions, which could be found in mathematics when observing the inverse of various functions, for example $\cos^{-1}(\varphi)$ or sqrt(x). We will present

- a quantum algorithm with n queries for a specific 4n-argument multivalued function
- a one-query quantum algorithm for a specific multivalued function with $N=2^n$ arguments.

Finally in the third part we will discuss low degree Boolean functions, the degree of algebraic polynomial representing a Boolean function and its relation to the complexity of a quantum query algorithm computing the function. We will present some approaches for definition of Boolean functions with sufficiently large gap between deterministic complexity and the degree of representing polynomial:

- an approach for a *3k*-argument polynomial of degree *2(k-1)* and corresponding Boolean function's deterministic complexity equal to the number of its arguments *3k*
- an approach for building a $3^{t+1}\cdot k$ -variable Boolean function f with deterministic complexity $D(f)=3^{t+1}\cdot k$ and the degree of the representing polynomial equal to $deg(f)=2^{t+1}\cdot(k-1)$, where $t\geq 1$ and any odd k.

We hope, algorithm examples presented in the thesis demonstrate some useful approaches of algorithm design, and will inspire newcomers of this area.

2 Preliminaries

In this section we consider notations, definitions and facts, either well-known or elementary, referenced directly or indirectly throughout the thesis, such as the classical decision tree, basics of quantum computing and the quantum query model.

We refer to [Am04] for most definitions in sections 2.1 and 2.2.

2.1 Quantum Computing

In this chapter we briefly outline the basic notions of quantum computing that are necessary to define the quantum query model. For more details, see ([KLM07], [ACRSZ10], [Am04]).

2.1.1 Quantum States

In finite dimensional quantum systems an *n-dimensional pure quantum state* is a unit vector in a 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 $|\psi\rangle = \sum_{i=0}^{n-1} a_i |i\rangle$ for some $a_i \in \mathbb{C}$.

In case of n=2 we have a one-qubit quantum system. A quantum bit, *qubit* for short, is a superposition of two basis states $|0\rangle$ and $|1\rangle$.

Since the norm of $|\psi\rangle$ is 1, $\sum_{i=0}^{n-1} |a_i|^2 = 1$. States $|0\rangle, |1\rangle, ..., |n-1\rangle$ are called *basis states.* Any state of the form $\sum_{i=0}^{n-1} a_i |i\rangle$ is called a *superposition* of $|0\rangle, |1\rangle, ..., |n-1\rangle$. The coefficient a_i is called *amplitude* of $|i\rangle$.

The quantum state can be changed by applying a unitary transformation or performing a measurement.

2.1.2 Unitary Transformation

A unitary transformation U is a linear transformation on $\mathbb{C}^n$ that maps each vector of unit norm to a vector of unit norm. Given a state $|\psi\rangle$, the state of the system after the transformation U is $U|\psi\rangle$.

2.1.3 Measurement

Here we use the simplest case of *quantum measurement*: the full measurement in the computational basis. Performing this measurement on a state $|\psi\rangle = a_0|0\rangle + ... + a_{n-1}|n-1\rangle$ gives the outcome i with probability $|a_i|^2$. The measurement changes state of the system to $|i\rangle$ and destroys the original state.

2.2 Query Models

The query model considers some explicitly defined function $f:\{0,1\}^n \to \{0,1\}^m$ with its arguments hidden in "black box". Algorithm has to output the value of the function correctly for an arbitrary input. The aim is to compute the value of function while making as less queries to "black box" as possible. Complexity of the algorithm is measured by number of queries on the worst-case input.

Decision trees are query algorithms described in terms of classical computation. For more details, see the survey by Buhrman and de Wolf [BW02] and Papadimitrou [Pa94].

2.2.1 Classical Decision Trees

We denote the *Hamming weight* of some input X by |X|.

A *deterministic decision tree* is a tree with internal nodes labeled with variables x_i, arrows exiting internal nodes labeled with possible variable values and leafs labeled with function values. A deterministic decision tree always follows the

same path for each input and produces the correct result with probability $p = 1$. Deterministic complexity of a function f is denoted by $D(f)$.

Definition 2.1. [BW02] *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. [BW02] *The **sensitivity $s_x(f)$** of f on input $(x_1, \ldots, x_i, \ldots, x_N)$ is the number of variables x_i with the following property: $f(x_1, \ldots, x_i, \ldots, x_N) \neq f(x_1, \ldots, 1-x_i, \ldots, x_N)$. The sensitivity of f is $s(f) = max_x s_x(f)$.*

It has been proved that $D(f) \geq s(f)$ [BW02]. In particular, if $s(f)$ is equal to the number of variables n, then $D(f)=n$.

A power of randomization might be added to decision trees [BW02].

A *probabilistic (randomized) decision tree* may contain internal nodes with probabilistic branching, i.e., multiple arrows exiting from the same node, each one labeled with a probability for algorithm to follow that way. The total probability to obtain the result r after execution of an algorithm on certain input X equals to the sum of probabilities for each leaf labeled with r to be reached. Total probability of an algorithm to produce the correct result is the probability on the worst-case input.

2.2.2 Quantum Query Model

A *quantum query algorithm* is the quantum counterpart of the decision tree. The following quantum query definition is applicable to problems with {0,1}-valued arguments only:

the black box gets a state $\sum_i a_i |i\rangle$ as an input and outputs a state $\sum_i a_i (-1)^{x_i} |i\rangle$, where the assignment of x_i is arbitrary, corresponding to computational needs of the algorithm. If the value of the i-th argument is 1, then the sign of the i-th amplitude a_i changes to the opposite. This form of a query is better suited for using in quantum algorithms, while there is other, more general, definition of a query for functions with a larger domain set. The first form of a query could be simulated by the second one.

See [KLM07], [ACRSZ10], [Am04] for detailed description.

A quantum algorithm with T queries is a sequence of unitary transformations on a finitite-dimensional space $\mathbb{C}^n$:

$$U_0 \to O \to U_1 \to O \to \ldots \to U_{T-1} \to O \to U_T \ ,$$

U_i's can be arbitrary unitary transformations that do not depend on input bits. O's are query transformations. The computation starts in the initial state $|\vec{0}\rangle$. Then, we apply predefined unitary transformations $U_0,O_x,\ldots,O_x,U_T$ and measure the final state. We denote the query that corresponds to an input $x=(x_1,x_2,\ldots,x_n)$ by O_x. Quantum query transformation expressed in matrix form is as follows:

$$O_x = \begin{pmatrix} (-1)^{\phi_0} & 0 & \ldots & 0 \\ 0 & (-1)^{\phi_1} & \ldots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \ldots & (-1)^{\phi_{n-1}} \end{pmatrix} ,$$

where ϕ_i is some x_j from the input string $X=(x_1,x_2,\ldots,x_n)$.

Each amplitude of the final state is treated as an output of the algorithm equal to some value r_i from the range set of the function. Probability to get the output value r afer algorithm execution on some input X is the sum of squared moduli of all amplitudes of the final state that correspond to outputs with function value r.

The quantum algorithm computes a function $f(x_1,\ldots,x_N)$ if for every $x=(x_1,x_2,\ldots,x_N)$, for which f is defined, the probability that squared modulus of a certain amplitude of the final state $U_T O_x U_{T-1} \ldots O_x U_0 |0\rangle$ equals $f(x_1,\ldots,x_N)$ is at least $1-\epsilon$ for some fixed $\epsilon < 1/2$. The exact quantum algorithm computes a function with probability 1, i.e. $\epsilon = 0$.

There are several types of quantum algorithms [BW02]. The present survey uses the notion of an exact and a bounded-error quantum query algorithms.

Definition 2.3. [BW02] *A quantum query algorithm computes f* ***exactly*** *if the output equals f(X) with 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 2.4. [BW02] *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* $Q_p(f)$.

2.3 The Deutch Algorithm

The most referenced quantum query algorithm is the Deutch algorithm [De85], [CEMM98]. It computes the XOR Boolean function for two bits exactly using a single query, while any classical algorithm asks at least two. Next we explain how the algorithm works.

The algorithm uses one qubit quantum system with basis states $|0\rangle, |1\rangle$. We begin in the state $|\varphi_0\rangle = |0\rangle$ and then apply *H*, then *Q* and then *H*, where *H* is Hadamard matrix and Q is the query:

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

The algorithm is a sequence of transformations $V_x = H \cdot Q \cdot H$. Finally, we perform the measurement: if the final state is $|0\rangle$, we output 1, and otherwise 0.

To see how the described algorithm works, note that $H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, which is a superposition of two basis states. For the input string *x=00 Q* is the identity matrix, but for *x=11 Q* is minus identity, thus:

$$V_{00}|0\rangle = H \cdot I \cdot H|0\rangle = |0\rangle$$
$$V_{11}|0\rangle = H \cdot (-I) \cdot H|0\rangle = -|0\rangle$$

For *x = 10* $Q \cdot H|0\rangle = \frac{1}{\sqrt{2}}(-|0\rangle + |1\rangle)$ according to *Q* definition, so $V_{10}|0\rangle = -|1\rangle$.

Similar result holds for the string *x = 01*: $Q \cdot H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, $V_{01}|0\rangle = |1\rangle$.

Therefore for *x* with all bits equal the final state is $V_x|\varphi_0\rangle = |0\rangle$ with probability p=1.

For all other input strings the final state is $V_x|\varphi_0\rangle=|1\rangle$ up to a phase factor.

To compute the general form of $XOR(x_1,...,x_N)$ Boolean function, we have to extend the previous algorithm to the case of $N=2n$ arguments: the transformation sequence is equal to $V_x=H\cdot Q_1\cdot...\cdot Q_n\cdot H$, where Q_i queries values of the i-th and the $(i+1)$-th arguments, $1\le i\le n$.

3 Quantum Query Algorithms for Boolean Functions. Bounded-Error Quantum Query Algorithms.

There are different types of query algorithms, which are mainly classified by the probability of returning the correct result:

- exact algorithms are supposed to return the correct result on any input with probability $p = 1$
- probabilistic algorithms either return a result which is correct with some probability or always return the correct result, but there is a probability of returning "unknown"

Quantum computation is popular and widely supported due to the improvements in computation; the best results were achieved with probabilistic algorithms. A quadratic [Gr96] and exponential [Sh97] quantum and classical complexity gap is achieved by means of bounded-error quantum query algorithms. Unfortunately, difficulties appear when there is a need to reuse an algorithm by running it in terms of a more complex algorithm: only exact algorithms allow reusage without the huge loss of the correct answer probability.

Meanwhile, for the exact algorithms only N versus 2N quantum and classical complexity gap is known. There are lots of examples of algorithms with such a complexity gap, but the most famous algorithm for XOR Boolean function [BW02] was the first one. Quantum exact algorithm complexity has the following estimation in comparison with the deterministic algorithm complexity: $Q_E(f) \geq \frac{1}{2} D(f)$, which is not proved or refuted, but stands as an open question and a subject of numerous researches: nobody managed to get ahead of XOR algorithm and use less than N quantum queries as opposed to classical 2N queries.

Speaking about query algorithm complexity, we should mention algorithms for promise problems ([DJ92], [CEMM98], [Si94], [FI09]): these are exact quantum algorithms with restricted domain; behaviour of the algorithm outside the domain has to be neglected. Restriction of this kind allows to widen the complexity gap up to exponential, see Deutch-Jozsa algorithm description [DJ92].

Let us return to the main object of this research section. Bounded-error algorithms output correct answer with some probability, consequently these algorithms should be compared to classical probabilistic query algorithms for estimation of the complexity gap.

The first subsection introduces an EQUALITY Boolean function for three, four and six arguments and offers a bounded-error one-query quantum algorithm for each case. Correct answer probabilities are 9/10, 3/4 and 9/16 correspondingly.

The second subsection presents a shift from n-bit EQUALITY function and its algorithm to (n-1)-bit conjunction and its algorithm preserving the same correct answer probability and asking one query: $Q_{9/10}(AND_2)=1$, $Q_{3/4}(AND_3)=1$ and $Q_{9/16}(AND_5)=1$.

The third subsection presents a Boolean function EOX (*Equality Of XORs*) designed on the basis of EQUALITY function, and its algorithm with 3n, 4n or 6n arguments and n quantum queries, preserving the correct answer probability of the basic algorithm. Some interesting applications of EOX-type function are offered in this subsection.

The fourth subsection stands apart from previous ones, but continues the research on effective algorithms for conjunctions. It shows a very unique quantum query algorithm for 2n-argument AND Boolean function with n quantum queries and the correct answer probability p = 3/4.

The fifth subsection presents a stand-alone quantum query algorithm for AZSO (*All Zeroes or Single One*) Boolean function of 6 arguments: true if the Hamming weight of the input string is 0 or 1. The algorithm has correct answer probability p = 9/16 and uses one quantum query.

A technical moment of a bounded-error algorithm design: while in case of an exact quantum algorithm we expect the measurement to give either 0 or 1 for the squared amplitude value, the situation is a bit different for a bounded-error algorithm. If we measure a quantum state $|0\rangle$, we observe an amplitude of the first element of the result vector. Suppose the amplitude is equal to some v (less or equal to 1; the sum of squared amplitudes over the vector are 1), then the algorithm's output is:

- 1 with the correct answer probability v^2, if $v^2 \geq \frac{1}{2}$,
- 0 with error probability v^2, if $v^2 < \frac{1}{2}$.

All results have been checked using *Wolfram Mathematica* tool. See *Appendices 1-5* for program code examples.

3.1 Quantum Query Algorithms for 3-, 4- and 6-bit EQUALITY Boolean Function

In this section we define an EQUALITY Boolean function, which is equal to 1 on all-ones or all-zeroes input strings, otherwise function is equal to 0. We offer bounded-error quantum query algorithms for the case of three, four and six arguments.

3.1.1 Quantum Algorithm for $EQUALITY_3$ Function: $Q_{9/10}(EQUALITY_3)=1$

This section is partly based on the author's Master Thesis [Mi07] . We define a 3-argument Boolean function $EQUALITY_3$ equal to *1* on vectors *000* and *111*, otherwise function is equal to *0*: $EQUALITY_3(x_1, x_2, x_3)=1 \Leftrightarrow [x_1=x_2=x_3]$

Theorem 3.1. *There is a bounded-error quantum query algorithm computing the Boolean function $EQUALITY_3$ (x_1,x_2,x_3) with one quantum query and error probability p = 1/10:* $Q_{9/10}(EQUALITY_3)=1$

Proof. The algorithm for $EQUALITY_3$ Boolean function uses 2-qubit quantum system with basis states $\{|0\rangle, |1\rangle, |2\rangle, |3\rangle\}$.

Define unitary matrices U_0 and U_1 by

$$U_0=H_{4x4}=\frac{1}{2}\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & -1\end{pmatrix}, U_1=\begin{pmatrix}\frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}}\\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2}\\ \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & -\frac{1}{\sqrt{10}} & -\frac{1}{\sqrt{10}}\\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2}\end{pmatrix}$$

respectively. U_0 is a 4x4 Hadamard matrix here. Define a query matrix Q by

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

We begin in the state $|\varphi_0\rangle=|0\rangle$ and then apply U_0, then Q, and then U_1. Finally, we perform the measurement consisting of a projection onto the state $|0\rangle$ and its orthogonal complement. If the output is $|0\rangle$, we output 1, otherwise 0. Consequently, the final quantum state, the state after applying a transformation sequence on the initial quantum state, is $|\varphi\rangle=U_1\cdot Q\cdot U_0|0\rangle$.

The claim is that the sequence $V_x=U_1\cdot Q\cdot U_0$ leaves $|\varphi_0\rangle$ unchanged up to the phase factor, when input string has all bits equal, otherwise maps $|\varphi_0\rangle$ into a subspace orthogonal to $|\varphi_0\rangle$. To see that the claim is correct, note first that $U_0|0\rangle=\frac{1}{2}\sum_{i=0}^{3}|i\rangle$, since U_0 is Hadamard matrix. For the input string *x=000 Q* is the identity matrix, but for *x=111 Q* is minus identity.

$$V_{000}|0\rangle=U_1\cdot I\cdot U_0|0\rangle=\left(\frac{3}{\sqrt{10}}\quad 0\quad \frac{1}{\sqrt{10}}\quad 0\right)^T,$$
$$V_{111}|0\rangle=U_1\cdot(-I)\cdot U_0|0\rangle=\left(-\frac{3}{\sqrt{10}}\quad 0\quad -\frac{1}{\sqrt{10}}\quad 0\right)^T,$$

which is the quantum state $|0\rangle$ with probability 9/10 after the measurement.

For *x = 100* we get $Q\cdot U_0|0\rangle=\frac{1}{2}(-|0\rangle-|1\rangle+|2\rangle+|3\rangle)$ according to *Q* definition , so

$V_{100}|0\rangle=\left(\frac{1}{\sqrt{10}}\quad 0\quad -\frac{3}{\sqrt{10}}\quad 0\right)^T$, which is not the quantum state $|2\rangle$ (or not the state $|0\rangle$)with probability 9/10. Similar result holds for the string *x = 011* :

$$Q\cdot U_0|0\rangle=\frac{1}{2}(|0\rangle+|1\rangle-|2\rangle-|3\rangle),\quad V_{011}|0\rangle=\left(-\frac{1}{\sqrt{10}}\quad 0\quad \frac{3}{\sqrt{10}}\quad 0\right)^T.$$

For *x = 001* :

$$Q\cdot U_0|0\rangle=\frac{1}{2}(|0\rangle+|1\rangle+|2\rangle-|3\rangle),\quad V_{001}|0\rangle=\left(\frac{1}{\sqrt{10}}\quad \frac{1}{2}\quad \frac{2}{\sqrt{10}}\quad -\frac{1}{2}\right)^T,$$ which is

not the state $|0\rangle$ with probability 9/10. Similar result holds for the string *x = 010*.

Thus, for *x* with all bits equal the final state is $V_x|\varphi_0\rangle=|0\rangle$ with probability 9/10. For all other input strings the final state is a superposition of states $|1\rangle,|2\rangle,|3\rangle$ with the sum of squared amplitudes equal to 9/10. Such a state is orthogonal to the initial state, proving our claim. □

All results have been checked using *Wolfram Mathematica* program (see *Appendix 1* for program code and output values)

What is the benefit of the quantum algorithm in comparison to the classical deterministic or probabilistic algorithm? By sensitivity on the input *000* $D(EQUALITY_3)=3$, which means there is no deterministic algorithm computing $EQUALITY_3$ by asking less than 3 questions . However, it is unfair to compare a bounded-error quantum algorithm with a deterministic one.

Let us find the highest possible probability for a classical randomized decision tree which computes this function with one query.

Theorem 3.2. *The Boolean function $EQUALITY_3(x_1,x_2,x_3)$ is computable by a randomized classical decision tree with one query with maximum probability p=½.*

Proof. The general form of the optimal randomized decision tree is shown in Fig. 3.1.

Let us denote

- Pr(r|X) to be the probability of an answer $r\in\{0,1\}$ after execution of the algorithm on the input string X
- p_0 to be the probability to reach the answer 1 in case one argument is known and equal to 0
- p_1 to be the probability to reach the answer 1 in case one argument is known and equal to 1

Due to symmetry of the function $p_0=p_1$, denote both of them by *p*.

Then: $Pr(1|X=000\cup X=111)=3\cdot\frac{1}{3}q\,p=q\cdot p$

$$Pr(0|X=001 \cup X=010 \cup X=100)$$
$$...=Pr(0|X=011 \cup X=101 \cup X=110)$$
$$...=1-q+\frac{1}{3}q(1-p)+\frac{1}{3}q(1-p)+\frac{1}{3}q(1-p)$$
$$...=1-q+q\cdot(1-p)=1-q\cdot p$$

The probability of the correct answer is the minimum of *Pr(0)* and *Pr(1)*, *min(qp, 1-qp)*, which has to be as large as possible to make the algorithm efficient. The highest probability is obtained in case when both answers are equally probable, *qp = 1 – qp*, *qp = ½,* which is the probability of the described algorithm. □

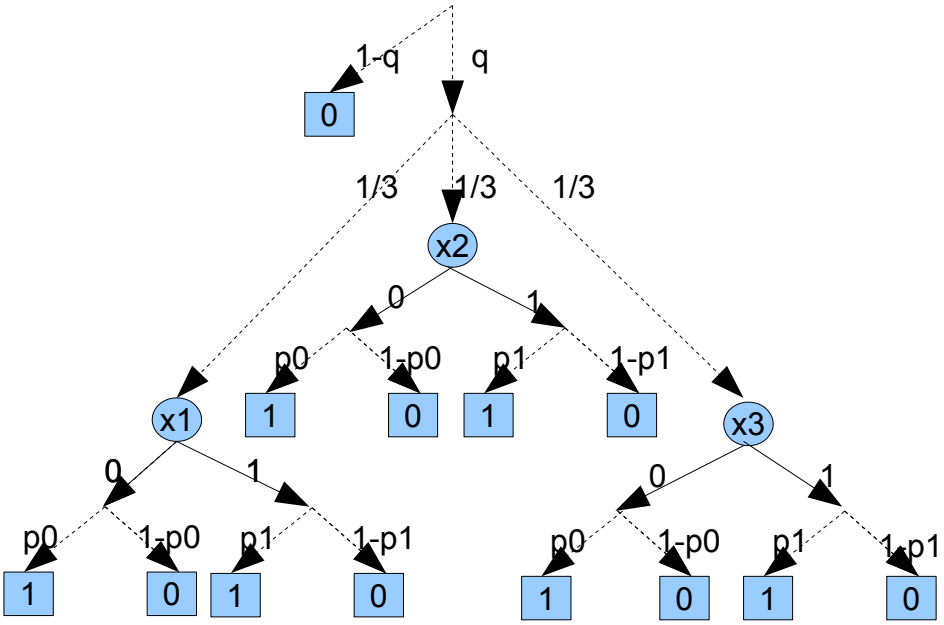


Fig. 3.1 The general form of the optimal classical randomized decision tree computing $EQUALITY_3(x_1,x_2,x_3)$. *p*0 is the probability to reach answer 1 if one argument is known and equal to 0, similarly for p1.

3.1.2 Quantum Algorithm for EQUALITY$_4$ Function: $Q_{3/4}$(EQUALITY$_4$) = 1

We define a 4-argument Boolean function EQUALITY$_4$ equal to 1 on vectors *0000* and *1111*, otherwise function is equal to 0: $EQUALITY_4(x_1, x_2, x_3, x_4) = 1 \Leftrightarrow [x_1 = x_2 = x_3 = x_4]$

Theorem 3.3. *There is a bounded-error quantum query algorithm computing the Boolean function EQUALITY$_4$ with one quantum query. Error probability is no more than ¼:* $Q_{3/4}(EQUALITY_4) = 1$

Proof. The algorithm for EQUALITY$_4$ Boolean function uses 2-qubit quantum system with basis states $\{|0\rangle, |1\rangle, |2\rangle, |3\rangle\}$.

Define a unitary matrix U_0, which is a 4x4 Hadamard matrix, and U_1 by

$$U_0 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, U_1 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & -1 & 1 \end{pmatrix}$$

respectively. Define a query matrix Q by

$$Q = \begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 \\ 0 & 0 & (-1)^{x_3} & 0 \\ 0 & 0 & 0 & (-1)^{x_4} \end{pmatrix}$$

We begin in the state $|\varphi_0\rangle = |0\rangle$ and apply U_0, then Q, and then apply U_1. Finally, we perform the measurement consisting of a projection onto the state $|0\rangle$ and its orthogonal complement. If the output is $|0\rangle$, we output 1, otherwise 0. The final quantum state is $|\varphi\rangle = U_1 \cdot Q \cdot U_0 |0\rangle$.

We claim that the sequence $V_x = U_1 \cdot Q \cdot U_0$ leaves $|\varphi_0\rangle$ unchanged up to the phase factor, when input string has all bits equal, otherwise maps $|\varphi_0\rangle$ into a subspace orthogonal to $|\varphi_0\rangle$.

The proof below is very similar to one presented in the section 3.1.1. Here we just repeat it shortly.

Since $U_0|0\rangle=\frac{1}{2}\sum_{i=0}^{3}|i\rangle$, for the input string *x=0000 Q* is the identity matrix, but for *x=1111 Q* is minus identity:

$$V_{0000}|0\rangle=U_1\cdot I\cdot U_0|0\rangle=|0\rangle,$$
$$V_{1111}|0\rangle=U_1\cdot(-I)\cdot U_0|0\rangle=-|0\rangle \quad .$$

For the input string *x = 1100*:

$Q\cdot U_0|0\rangle=\frac{1}{2}(-|0\rangle-|1\rangle+|2\rangle+|3\rangle)$, therefore $V_{1100}|0\rangle=|2\rangle$.

Similar results, the state $|1\rangle$, $|2\rangle$ or $|3\rangle$ up to the phase factor, hold for the other strings with Hamming weight 2.

For the input string *x = 1000*:

$Q\cdot U_0|0\rangle=\frac{1}{2}(-|0\rangle+|1\rangle+|2\rangle+|3\rangle)$, $V_{1000}|0\rangle=\frac{1}{2}(|0\rangle-|1\rangle-|2\rangle+|3\rangle)$.

Easy to see that for the input *x=0111* the outcome is $V_{0111}|0\rangle=-\frac{1}{2}(|0\rangle-|1\rangle-|2\rangle+|3\rangle)$, which is not the state $|0\rangle$ and therefore orthogonal to $|0\rangle$ with error probability *p = 1/4*. For all other input strings with Hamming weight 1 or 3 results are similar and lie in the span of $|0\rangle,|1\rangle,|2\rangle,|3\rangle$, where all coefficients are positive or negative ½ .

Thus, for *x* with all bits equal the final state is $V_x|\varphi_0\rangle=|0\rangle$ with probability *p = 1*. For all other input strings the result is orthogonal to the initial state, although with error probability *p = 1/4*, proving our claim. □

All results have been checked using *Wolfram Mathematica* program (see *Appendix 2* for program code and output values)

Let us now discuss the benefit of the quantum algorithm versus classical deterministic. By sensitivity on the input *0000,* $D(EQUALITY_4)=4$, which means a deterministic algorithm computing $EQUALITY_4$ requires 4 questions .

3.1.3 Quantum Algorithm for EQUALITY$_6$ Function: $Q_{9/16}(EQUALITY_6)=1$

We define a 6-argument Boolean function EQUALITY$_6$ equal to 1 on vectors 000000 and 111111, otherwise function is equal to 0: $EQUALITY_6(x_1, x_2, x_3, x_4, x_5, x_6)=1 \Leftrightarrow [x_1=x_2=x_3=x_4=x_5=x_6]$

Theorem 3.4. *There is a bounded-error quantum query algorithm computing the Boolean function EQUALITY$_6$ (x_1,x_2,x_3,x_4,x_5,x_6) with one quantum query and correct answer probabilty p = 9/16:* $Q_{9/16}(EQUALITY_6)=1$.

Proof. The algorithm for EQUALITY$_6$ Boolean function uses 3-qubit quantum system with basis states $\{|0\rangle,|1\rangle,|2\rangle,|3\rangle,|4\rangle,|5\rangle,|6\rangle,|7\rangle\}$.

Define a unitary matrix U_0 and U_1 by

$$U_0=H_{8x8}=\frac{1}{\sqrt{8}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix},$$

$$U_1=\frac{1}{\sqrt{8}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \end{pmatrix}$$

respectively. Define a query matrix Q by

$$Q=\begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & (-1)^{x_3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & (-1)^{x_4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (-1)^{x_5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & (-1)^{x_6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

We begin in the state $|\varphi_0\rangle=|0\rangle$, apply U_0, then Q, and then apply U_1. Finally, we perform the measurement consisting of a projection onto the state $|0\rangle$ and its orthogonal complement. If the output is $|0\rangle$, we output 1, otherwise 0. Consequently, the final quantum state, that is a state after applying a transformation sequence on the initial quantum state, is equal to $|\varphi\rangle=U_1\cdot Q\cdot U_0|0\rangle$.

The claim is that the sequence $V_x=U_1\cdot Q\cdot U_0$ leaves $|\varphi_0\rangle$ unchanged up to the phase factor, when the input string has all bits equal, otherwise maps $|\varphi_0\rangle$ into a subspace orthogonal to $|\varphi_0\rangle$.

Firstly, $U_0|0\rangle=\frac{1}{\sqrt{8}}\sum_{i=0}^{7}|i\rangle$, since U_0 is Hadamard matrix. For the input string *x=000000 Q* is the identity matrix, then

$$V_{000000}|0\rangle=U_1\cdot I\cdot U_0|0\rangle=\left(\frac{3}{4}\quad -\frac{1}{4}\quad \frac{1}{4}\quad -\frac{1}{4}\quad 0\quad 0\quad 0\quad \frac{1}{2}\right)^T.$$

While for *x = 111111*

$$Q\cdot U_0|0\rangle=\frac{1}{\sqrt{8}}(-|0\rangle-|1\rangle-|2\rangle-|3\rangle-|4\rangle-|5\rangle+|6\rangle+|7\rangle)$$

$$V_{111111}|0\rangle=\left(-\frac{3}{4}\quad \frac{1}{4}\quad -\frac{1}{4}\quad \frac{1}{4}\quad 0\quad 0\quad 0\quad \frac{1}{2}\right)^T.$$

After the measurement it is the quantum state $|0\rangle$ with probability 9/16.

Running the algorithm on any other input vector results in a quantum state orthogonal to $|0\rangle$ with error probability no more than ¼, proving our claim. See *Appendix 3* for other input and output vector correspondence. □

By sensitivity on the input *000000*, $D(EQUALITY_6)=6$.

Theorem 3.5. *The Boolean function $EQUALITY_6(x_1,...,x_6)$ is computable by a randomized classical decision tree with one query with maximum probability $p = ½$.*

Proof. The general form of the optimal randomized decision tree is shown in Fig. 3.2.

Let us denote the probability of an answer $r \in \{0,1\}$ after execution of the algorithm on the input string X by *Pr(r|X)*. Due to symmetry of the function, probabilities for both outputs 0 and 1 are equal after one query, denote both of them by *p*.

Then: $Pr(1|X=000000 \cup X=111111)=6\cdot\frac{1}{6}q \; p=q\cdot p$

$$Pr(0|X\neq 000000 \cup X\neq 111111)=1-q+6\cdot\frac{1}{6}q(1-p)=1-q+q\cdot(1-p)=1-q\cdot p$$

The probability of the correct answer is *min(qp, 1-qp)*. The highest probability is obtained in case when both answers are equally probable, $qp = 1 - qp$, $qp = ½$, consequently, the probability of the described algorithm is ½. □

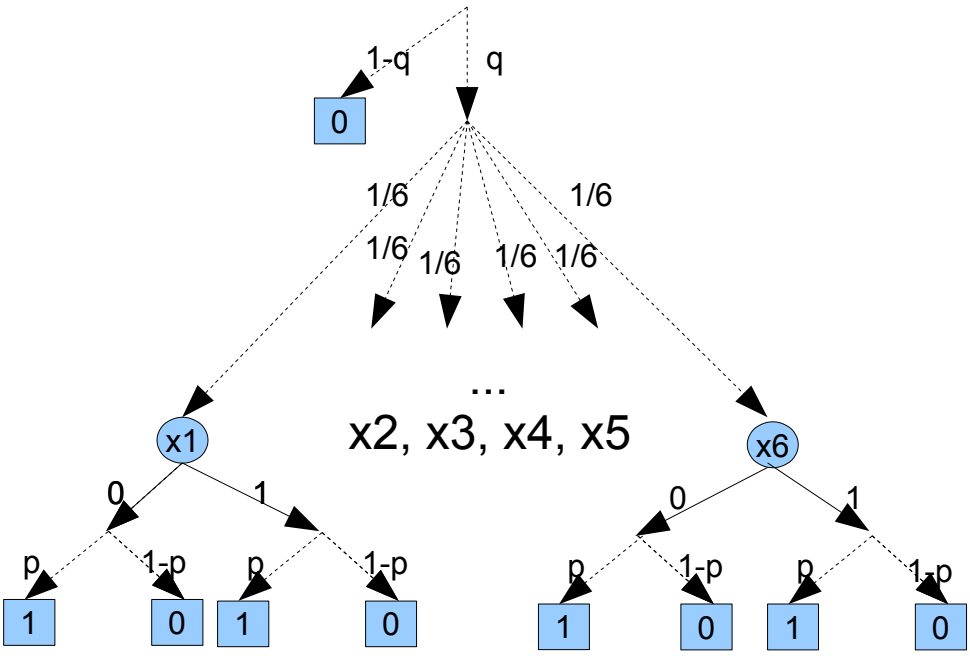


Fig. 3.2 The general form of the optimal classical randomized decision tree for computing $EQUALITY_6(x_1,...,x_6)$

3.2 Quantum Query Algorithms for 2-, 3-, 5- bit Conjunctions

Since every Boolean function is expressed as a logical formula, it can be normalized and rewritten in CNF(conjunctive normal form) or DNF (disjunctive normal form).

Conjunctive normal form is a conjunction of disjunctions of arguments or negated arguments, disjunctive normal form is a disjunction of conjunctions of arguments or negated arguments. AND and OR Boolean functions are basic elements which are used in every formula several times.

According to [Wo01], N queries are required in N-bit case of AND and OR, which is a proved lower bound. Improving the computation of primary functions, which are used to stick parts of a Boolean function in CNF/DNF to each other, will boost the computation of any complex formula.

Grover's search algorithm [Gr96] is one of the most well-known quantum algorithms, it computes N-bit OR by asking $O(\sqrt{N})$ queries. As far as

$AND(x_1,x_2) = NOT\ OR(NOT\ x_1,\ NOT\ x_2)$,

we can transform Grover's algorithm to compute AND Boolean function, which is obviously effective on sufficiently large N. In case of AND/OR defined on a small number of arguments much quicker algorithms exist there. Meanwhile, the number of AND/OR calls in one complex Boolean function might be large enough, consequently, the speed up of its computation is going to be significant.

This section presents algorithms for AND Boolean function for 2, 3 and 5 arguments. All quantum query algorithms are bounded-error, use one quantum query only having higher correct answer probability than their classical counterparts – randomized probabilistic algorithms. All algorithms operate faster than Grover's algorithm for the particular number of arguments.

3.2.1 Quantum Query Algorithm for AND_2: $Q_{9/10}(AND_2)=1$

This section is partly based on the article [VM10] and demonstrates a bounded-error quantum query algorithm for $AND_2(x_1,x_2)$ Boolean function with the best ever achieved correct answer probability $p = 9/10$.

According to [KW04], the proof of Lemma 1, there exists an algorithm for computing an arbitrary two-argument Boolean function with probability $p=11/14$. The same paper makes a contribution about an optimal probability equal to 9/10.

Theorem 3.6. *There is a quantum query algorithm computing the Boolean function $AND_2(x_1,x_2)$ with one quantum query and correct answer probability $p = 9/10$:* $Q_{9/10}(\mathrm{AND}_2)=1$.

Proof. Both function and algorithm are obtained from the function $EQUALITY_3(x_1,x_2,x_3)$ and its quantum algorithm respectively. Setting x_3 argument to be a constant value *1*, we force $EQUALITY_3(x_1,x_2,1)=1$ and therefore $x_1 = x_2 = 1$. For x_1x_2 equal to 00 or 01, or 10, $EQUALITY_3(x_1,x_2,1)=0$ exactly as function $AND(x_1,x_2)$ does. We get a quantum algorithm for $AND(x_1,x_2)$ by redefining the query operator in the $EQUALITY_3$ algorithm and leaving everything else unchanged:

- the algorithm uses 2-qubit quantum system, starts with |0> quantum state and stops in a state $|\varphi\rangle = U_1 \cdot Q \cdot U_0 |0\rangle$ after application of a transformation sequence on the initial quantum state
- finally, we measure the quantum state |0>, which probability is the probability of the correct value of $AND(x_1,x_2)$, i.e. $p=9/10$.
- U_0 is a 4x4 Hadamard matrix and U_1 is defined as follows

$$U_1 = \begin{pmatrix} \frac{1}{\sqrt{10}} & \frac{1}{\sqrt{10}} & \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \sqrt{\frac{2}{5}} & \sqrt{\frac{2}{5}} & -\frac{1}{\sqrt{10}} & -\frac{1}{\sqrt{10}} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}.$$

The new query matrix Q is:

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

All the processing repeats one from the section 3.1.1 with the only difference – the last of three arguments is constantly set to 1. □

All results have been checked using *Wolfram Mathematica* program (see *Appendix 1* for program code and output values)

Next, let us discuss the classical counterpart and compare quantum versus classical complexity. $D(\mathrm{AND}_2)=2$, which is obvious, however, the classical counterpart of our bounded-error quantum algorithm is a randomized decision tree.

Theorem 3.7. *The Boolean function AND_2 (x_1,x_2) is computable by a classical randomized 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. 3.3.

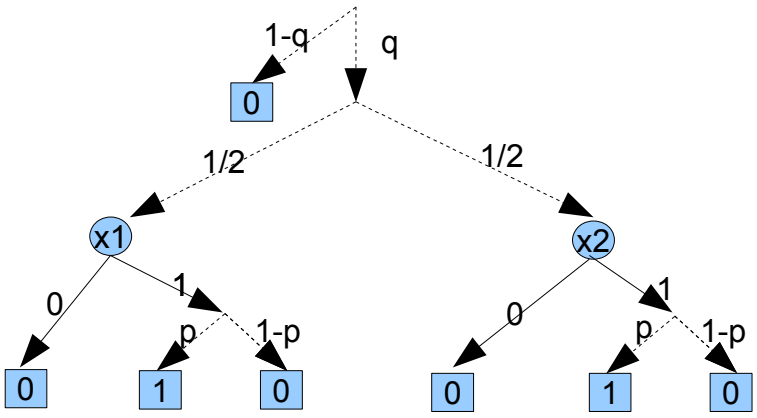


Fig. 3.3 The general form of the optimal classical randomized decision tree for computing AND_2 (x_1,x_2).

The probability to get the correct answer on the input 11 is $Pr(1|X=11)=2\cdot\frac{1}{2}q\ p=q\cdot p$. The probability to get the correct answer on all other inputs is as follows:

$$Pr(0|X=00)=1-q+\frac{1}{2}\cdot q+\frac{1}{2}\cdot q=1$$

$$Pr(0|X=01\cup X=10)=1-q+\frac{1}{2}\cdot q+\frac{1}{2}\cdot q\cdot(1-p)=1-\frac{1}{2}\cdot q\cdot p$$

The probability of the correct answer is the minimum of *Pr(0)* and *Pr(1)*, *min(qp, 1- ½ qp)*. The highest probability is obtained in case when both answers are equally probable, *qp = 1 – ½ qp*, *qp = 2/3,* which is the probability of the described algorithm. □

3.2.2 Quantum query algorithm for AND_3: $Q_{3/4}(AND_3)=1$

This section demonstrates a bounded-error quantum query algorithm for the Boolean function $AND_3(x_1,x_2,x_3)$ with the correct answer probability p=3/4.

Theorem 3.8. *There is a quantum query algorithm computing the Boolean function $AND_3(x_1,x_2,x_3)$ with one quantum query and the correct answer probability p=3/4:* $Q_{3/4}(\mathrm{AND}_3)=1$

Proof. Both function and algorithm are obtained from the function $EQUALITY_4(x_1,x_2,x_3,x_4)$ and its quantum algorithm, correspondingly.

Setting x_4 argument to be a constant value *1*, we force $EQUALITY_4(x_1,x_2,x_3,1)=1$ and therefore $x_1=x_2=x_3=1$. For the input string $x_1x_2x_3$ not equal to 111 $EQUALITY_4(x_1,x_2,x_3,1)=0$ exactly as function $AND_3(x_1,x_2,x_3)$ does. We get a quantum algorithm for $AND_3(x_1,x_2,x_3)$ by redefining the query operator in the $EQUALITY_4$ algorithm, leaving everything else unchanged:

- the algorithm uses 2-qubit quantum system and starts with |0> quantum state, and stops in a state $|\varphi\rangle=U_1\cdot Q\cdot U_0|0\rangle$ after application of a transformation sequence on the initial quantum state
- finally, we get the algorithm's value by measuring the quantum state |0>. Error probability is no more than *p=1/4*.
- U_0 is a 4x4 Hadamard matrix and

$$U_1=\frac{1}{2}\begin{pmatrix}1&1&1&1\\1&-1&-1&1\\1&1&-1&-1\\-1&1&-1&1\end{pmatrix}.$$

The new query matrix Q is:

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

All the processing repeats one from section 3.1.2 with the only difference – the last of four bits is constantly set to 1. □

All results have been checked using *Wolfram Mathematica* program (see *Appendix 2* for program code and output values).

It is obvious, $D(\mathrm{AND}_3)=3$.

Theorem 3.9. *The Boolean function AND₃ (x₁,x₂,x₃) is computable by a classical randomized decision tree with one query with the maximum probability p= 3/5.*

Proof. The general form of the optimal randomized decision tree is shown in Fig. 3.4.

The probability to get the correct answer on the input 111 is

$$Pr(1|X=111)=3\cdot\frac{1}{3}q\ p=q\cdot p\ .$$

The probability to get the correct answer on all other inputs is as follows:

$$Pr(0|X=000)=1-q+3\cdot\frac{1}{3}\cdot q=1$$

$$Pr(0|X=001\cup X=010\cup X=100)=1-q+\frac{1}{3}q\cdot(3-p)=1-\frac{1}{3}qp$$

$$Pr(0|X=011\cup X=101\cup X=110)=1-q+\frac{1}{3}q\cdot(3-2\mathrm{p})=1-\frac{2}{3}qp$$

$$Pr(0)=min(1,1-\frac{1}{3}qp,1-\frac{2}{3}qp)=1-\frac{2}{3}qp,\ \ qp\le 1$$

The probability of the correct answer is the minimum of *Pr(0)* and *Pr(1)*,

min(qp, 1 - 2/3 qp), the highest probability is obtained in case when both answers are equally probable, *qp = 1 – 2/3 qp*, consequently, *qp = 3/5,* which is the probability of

the described algorithm. □

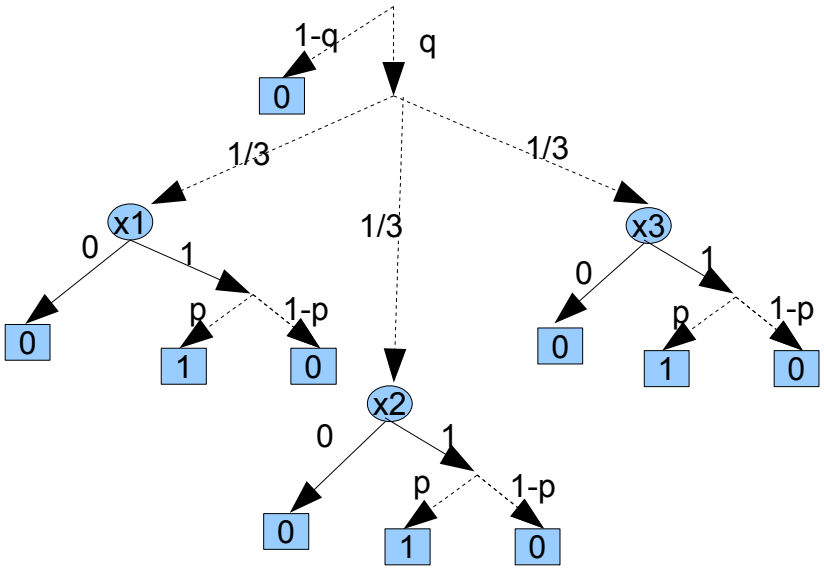


Fig. 3.4 The general form of the optimal classical randomized decision tree for computing $AND_3(x_1,x_2,x_3)$ Boolean function

3.2.3 Quantum query algorithm for AND_5: $Q_{9/16}(AND_5)=1$

This section demonstrates a bounded-error quantum query algorithm for $AND_5(x_1,x_2,x_3,x_4,x_5)$ Boolean function with the correct answer probability $p = 9/16$.

Theorem 3.10. *There is a quantum query algorithm computing the Boolean function $AND_5(x_1,x_2,x_3,x_4,x_5)$ with one quantum query and the correct answer probability $p= 9/16$:* $Q_{9/16}(\mathrm{AND}_5)=1$

Proof. Both function and algorithm are obtained from the function $EQUALITY_6(x_1,x_2,x_3,x_4,x_5,x_6)$ and its quantum algorithm respectively.

Setting x_6 argument to be a constant value *1*, we force the value of the function to be $EQUALITY_6(x_1,x_2,x_3,x_4,x_5,1)=1$, and therefore $x_1=x_2=x_3=x_4=x_5=1$. For $x_1x_2x_3x_4x_5$ not equal to 11111, the value of the function is $EQUALITY_6(x_1,x_2,x_3,x_4,x_5,1)=0$ exactly as function $AND_5(x_1,x_2,x_3,x_4,x_5)$. We get the quantum algorithm for $AND_5(x_1,x_2,x_3,x_4,x_5)$ by redefining the query operator in the $EQUALITY_6$ algorithm, leaving everything else unchanged:

- the algorithm uses 3-qubit quantum system and starts with the quantum state |0>, and stops in the state $|\varphi\rangle = U_1 \cdot Q \cdot U_0 |0\rangle$ after application of a transformation sequence on the initial quantum state
- finally, we get the function's value by measuring the quantum state |0>. The correct answer probability is *p=9/16*.
- U_0 is 8x8 Hadamard matrix and U_1 is the following:

$$U_1 = \frac{1}{\sqrt{8}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \end{pmatrix}$$

The new query matrix Q is:

$$Q = \begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & (-1)^{x_3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & (-1)^{x_4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (-1)^{x_5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

All the processing repeats one from the section 3.1.3 with the only difference – the last of six bits is constantly set to 1. □

All results have been checked using *Wolfram Mathematica* program (see *Appendix 3* for program code and output values)

Let us evaluate the classical algorithm's complexity. First, it is obvious $D(\mathrm{AND}_5) = 5$.

Theorem 3.11. *The Boolean function $AND_5(x_1,x_2,x_3,x_4,x_5)$ is computable by a classical randomized decision tree with one query and the maximum probability p= 5/9.*

Proof. The general form of the optimal randomized decision tree for 5-argument AND

Boolean function is shown in Fig. 3.5.

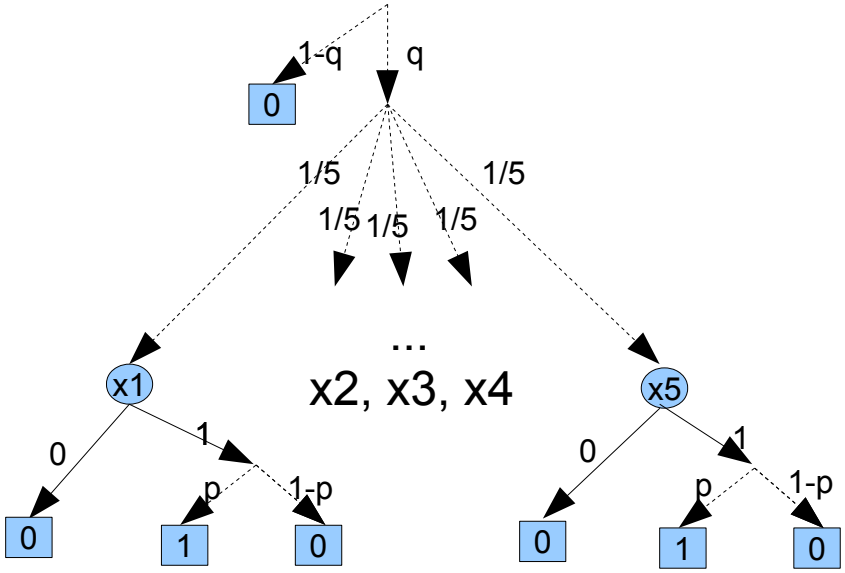


Fig. 3.5 The general form of the optimal classical randomized decision tree for computing $AND_5(x_1,\ldots,x_5)$.

The probability to get the correct answer on the input 11111 is:

$$Pr(1|X=11111)=5\cdot\frac{1}{5}q\,p=q\cdot p \quad .$$

The probability to get the correct answer on all other inputs:

$$Pr(0|X=00000)=1-q+5\cdot\frac{1}{5}q=1$$
$$Pr(0|X:|X|=1)=1-q+\frac{1}{5}q\cdot(5-p)=1-\frac{1}{5}qp$$
$$Pr(0|X:|X|=2)=1-q+\frac{1}{5}q\cdot(5-2\mathrm{p})=1-\frac{2}{5}qp \quad .$$
$$Pr(0|X:|X|=3)=1-q+\frac{1}{5}q\cdot(5-3\mathrm{p})=1-\frac{3}{5}qp$$
$$Pr(0|X:|X|=4)=1-q+\frac{1}{5}q\cdot(5-4\mathrm{p})=1-\frac{4}{5}qp$$

$$Pr(0)=min\left(1,1-\frac{1}{5}qp,1-\frac{2}{5}qp,1-\frac{3}{5}qp,1-\frac{4}{5}qp\right)=1-\frac{4}{5}qp$$

The probability of the correct answer is the minimum of *Pr(0)* and *Pr(1)*, *min(qp, 1-4/5 qp)*. The highest probability is obtained in case when both answers are equally probable, *qp = 1 – 4/5 qp*. Then *qp = 5/9,* which is the probability of the described algorithm. □

3.3 Quantum Query Algorithm for An Extension of EQUALITY Boolean function, EOX for 3n, 4n and 6n arguments

This section offers a set of Boolean functions and quantum algorithms, each based on the corresponding EQUALITY algorithm.

3.3.1 Quantum Algorithm for An Extension of EQUALITY$_3$ Boolean function, EOX$_{3n}$

This section is partly based on the author's Master Thesis [Mi07] . We define a 3n-argument Boolean function EOX$_{3n}$ (short for *Equality Of XORs*) as follows:

$$EOX_{3n}(x_1, x_2, x_3, \ldots, x_{3i+1}, x_{3i+2}, x_{3i+3}, \ldots, x_{3n-2}, x_{3n-1}, x_{3n}) = \ldots$$
$$\ldots = EQUALITY_3(\underbrace{XOR}_{i=0}^{n-1}(x_{3i+1}), \underbrace{XOR}_{i=0}^{n-1}(x_{3i+2}), \underbrace{XOR}_{i=0}^{n-1}(x_{3i+3}))$$

For example, for n=2, Boolean function EOX$_6$ is defined by it's truth table, see the Table 3.1 for details.

X	EOX$_6$	X	EOX$_6$
000000	1	011011	1
000111	1	011100	1
111000	1	100011	1
111111	1	100100	1
001001	1	101010	1
001110	1	101101	1
010010	1	110001	1
010101	1	110110	1
Otherwise	0		

Table 3.1. Boolean function EOX$_6$

Theorem 3.12. *There is a bounded-error quantum query algorithm computing the Boolean function* EOX$_{3n}$(X) *with n quantum queries and error probability p=1/10:* $Q_{9/10}(EOX_{3n}) = n$.

Proof. The algorithm for EOX$_{3n}$ Boolean function is very similar to one that

computes $EQUALITY_3$ function (see section 3.1.1 for details). The only difference is in the query. The current algorithm asks values of 3n arguments by executing n queries one after another,

$$\forall i = 0,\ldots,n-1:$$
$$Q_i = \begin{pmatrix} (-1)^{x_{3i+1}} & 0 & 0 & 0 \\ 0 & (-1)^{x_{3i+1}} & 0 & 0 \\ 0 & 0 & (-1)^{x_{3i+2}} & 0 \\ 0 & 0 & 0 & (-1)^{x_{3i+3}} \end{pmatrix}.$$

Everything except for the query remains the same:

- the initial quantum state is $|0\rangle$,
- the final state is reached by execution of a transformation sequence $|\varphi\rangle = U_1 \cdot Q_n \cdot \ldots \cdot Q_1 \cdot U_0 |0\rangle$ where U_0 and U_1 are exactly the same as in the section 3.1.1.

To prove that the described quantum algorithm computes the function, observe the product of all query matrices:

$$Q = \begin{pmatrix} (-1)^{\sum_{i=0}^{n-1} x_{3i+1}} & 0 & 0 & 0 \\ 0 & (-1)^{\sum_{i=0}^{n-1} x_{3i+1}} & 0 & 0 \\ 0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{3i+2}} & 0 \\ 0 & 0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{3i+3}} \end{pmatrix}$$
$$EOX_{3n}(x_{1,\ldots},x_{3n}) = EQUALITY_3\left(\sum_{i=0}^{n-1} x_{3i+1}, \sum_{i=0}^{n-1} x_{3i+2}, \sum_{i=0}^{n-1} x_{3i+3}\right), where \sum is\ modulo\ 2$$

Finally, we measure the quantum state $|0\rangle$, which squared amplitude is the probability of the value $EOX_{3n}(X) = 1$. The error probability is no more than $p=1/10$.

All results could be observed using *Wolfram Mathematica* program for $EQUALITY_3$ Boolean function (*Appendix 1*) by defining more queries in the program body and adding them to the transformation sequence. □

For example, in case of 6 arguments two queries are:

$$Q_1=\begin{pmatrix}(-1)^{x_1} & 0 & 0 & 0\\ 0 & (-1)^{x_1} & 0 & 0\\ 0 & 0 & (-1)^{x_2} & 0\\ 0 & 0 & 0 & (-1)^{x_3}\end{pmatrix}, Q_2=\begin{pmatrix}(-1)^{x_4} & 0 & 0 & 0\\ 0 & (-1)^{x_4} & 0 & 0\\ 0 & 0 & (-1)^{x_5} & 0\\ 0 & 0 & 0 & (-1)^{x_6}\end{pmatrix}.$$

Running the algorithm on 000000 and 15 other input strings from Table 3.1, the quantum state |0> is reached with probability $\frac{9}{10}$, which is equivalent to the algorithm's output 1 with probability 9/10.

By the definition of the function, $EOX_{3n}(000...0) = 1$. Changing only one bit value to 1 changes the function's value to 0. Consequently, the sensitivity of $EOX_{3n}(000...0)$ is equal to 3n, and $D(EOX_{3n})=3n$.

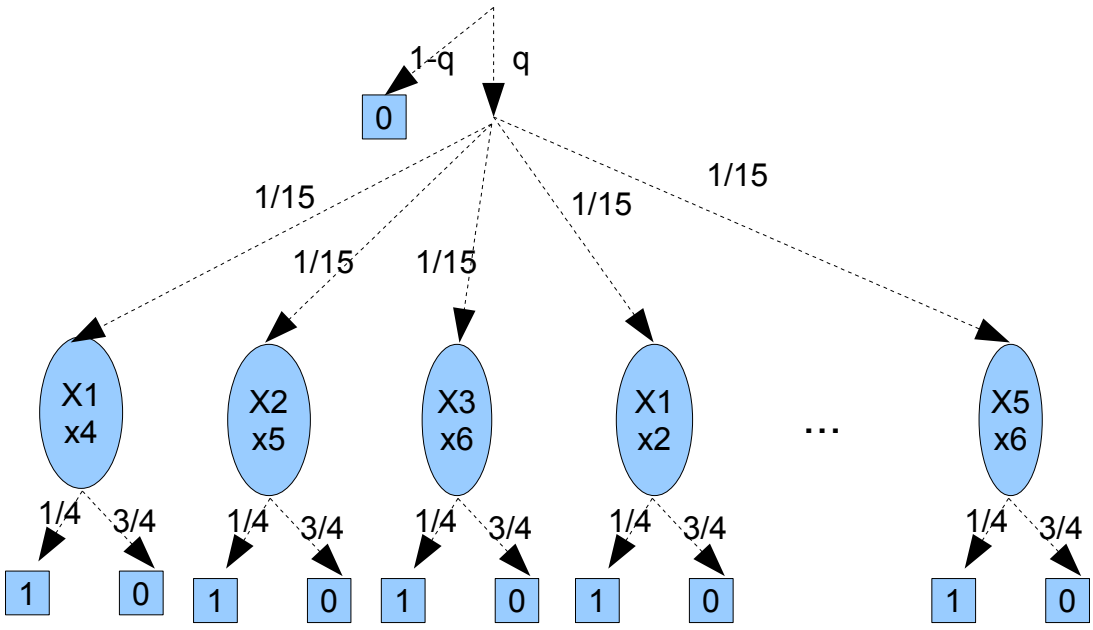


Fig. 3.6 The general form of the optimal classical randomized decision tree for computing $EOX_6(x_1,...,x_6)$.

Lemma 3.1. *Given the general optimal randomized decision tree of depth n computing $EOX_{3n}(X)$, the probability of getting the answer 1 is equal for each combination of randomly chosen n values.*

Proof. All 3n arguments of the function EOX_{3n} are divided into 3 non intersecting subsets according to the structure of the function:

$\{x_1,...,x_n\}$, $\{x_{n+1},...,x_{2n}\}$, $\{x_{2n+1},...,x_{3n}\}$.

Suppose we are given k_1, k_2, k_3 – numbers of queried arguments in the first, the second and the third subset, such that $n=k_1+k_2+k_3$.

Having n queried arguments, we have a probabilistic randomized decision tree of depth n. There are 2^{2n} different combinations of the remaining unqueried argument values. Let us count all possible strings of unquestioned arguments that give the value $EOX_{3n}(x_1,...,x_{3n})=1$.

Step 1. It is important to notice, that the number of all possible binary strings X of length K for which $XOR_K(X)=0$ is exactly the same as the number of all possible binary strings Y of length K for which $XOR_K(Y)=1$. This statement follows from the properties of the Binomial coefficients:

- to get $XOR_K(X)=0$, all strings with the even number of "1" should be count, $c_0=C_K^0+C_K^2+...+C_K^{2j}, 2j\le K$,
- to get $XOR_K(X)=1$, all strings with the odd number of "1" should be count, $c_1=C_K^1+C_K^3+...+C_K^{2j-1}, 2j-1\le K$.

We get $c_0 = c_1$. This property says that its not important which values do queried arguments k_i have. By the property of the Binomial coefficients, $\sum_{i=0}^{n} C_K^{\,i}=2^K$, consequently $c_0=c_1=2^{K-1}$.

Step 2. Let us make an assignment $K=n-k_i$. Then calculate c for each of k_i, mark it as $c_{ki}= 2^{n-ki-1}$. Calculate the product $c_{k1}\cdot c_{k2}\cdot c_{k3}$, which is the number of strings, where XOR over each subset is equal to 0. The same stands for 1. By definition of the function, the number of all strings for $EOX_{3n}(x_1,...,x_{3n})=1$ is $2\cdot 2^{n-k_1-1}\cdot 2^{n-k_2-1}\cdot 2^{n-k_3-1}=2^{2n-2}=2^{2(n-1)}$,which does not depend on the choice of $k_1+k_2+k_3=n$.

Step 3. Overall, the number of all different strings of length $2n$ is 2^{2n}. Consequently, the probability to get the answer 1 at the end of the branch of the probabilistic randomized decision tree is $\frac{2^{2(n-1)}}{2^{2n}}=\frac{1}{4}$ □

Theorem 3.13. *The Boolean function $EOX_{3n}(x_1,...,x_{3n})$ is computable by a randomized classical decision tree with n queries with the maximum probability p= ½.*

Proof. The optimal randomized decision tree for EOX_6 is shown in Fig. 3.6. The general form of an algorithm has n queries. According to Lemma 3.1, all tree's leafs with value 1 have the same probability. There are $3n\cdot(3n-1)\cdot...\cdot(2n+1)$ possibilities to choose n arguments out of 3n, which coincides with the number of arrows exiting from the root of the randomized tree:

$$Pr(1|X:EOX_{3n}(X)=1)=3n\cdot(3n-1)\cdot...\cdot(2n+1)\cdot\frac{1}{3n\cdot(3n-1)\cdot...\cdot(2n+1)}q\cdot p=q\cdot p$$
$$Pr(0|X:EOX_{3n}(X)=0)$$
$$...=1-q+3n\cdot(3n-1)\cdot...\cdot(2n+1)\cdot\frac{1}{3n\cdot(3n-1)\cdot...\cdot(2n+1)}q\cdot(1-p)$$
$$...=1-q+q\cdot(1-p)=1-q\cdot p$$

The probability of the correct answer is the minimum of *Pr(0)* and *Pr(1)*, *min(qp, 1-qp).* It has to be as large as possible to make the algorithm effective. The highest probability is obtained in case when both answers are equally probable: *qp = 1 – qp*, *qp = ½,*

which is the probability of the described algorithm. □

3.3.2 Quantum Algorithm for An Extension of EQUALITY$_4$ Boolean Function, EOX$_{4n}$

We define a 4n-argument Boolean function EOX_{4n} (short for *Equality Of XORs*):

$$EOX_{4n}(x_1,x_2,x_3,x_4,\underbrace{\ldots,x_{4i+1}}_{n-1},\underbrace{x_{4i+2}}_{n-1},\underbrace{x_{4i+3}}_{n-1},\underbrace{x_{4i+4}}_{n-1},\ldots,x_{4n-3},x_{4n-2},x_{4n-1},x_{4n})=...$$
$$...=EQUALITY_4(\underbrace{XOR(x_{4i+1})}_{i=0},\underbrace{XOR(x_{4i+2})}_{i=0},\underbrace{XOR(x_{4i+3})}_{i=0},\underbrace{XOR(x_{4i+4})}_{i=0})$$

Theorem 3.14. *There is a bounded-error quantum query algorithm computing the Boolean function $EOX_{4n}(X)$ with n quantum queries, the algorithm outputs the value 1 exactly and the value 0 with error probability p = 1/4:* $Q_{3/4}(EOX_{4n})=n$

Proof. The algorithm for EOX_{4n} Boolean function is very similar to one that computes the EQUALITY$_4$ function (see the section 3.1.4 for details). Instead of one query matrix the new algorithm uses n matrices one by one, each query of the form:

$$\forall i = 0,\ldots,n-1:$$

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

All other parts of the algorithm remain the same:

- the initial quantum state is $|0\rangle$,
- the final state is acquired after applying a transformation sequence $|\varphi\rangle = U_1 \cdot Q_n \cdot \ldots \cdot Q_1 \cdot U_0 |0\rangle$, where U_0 and U_1 are exactly the same as in the section 3.1.4.

To prove that the described quantum algorithm computes the function, observe the product of all query matrices:

$$Q = \begin{pmatrix} (-1)^{\sum_{i=0}^{n-1} x_{4i+1}} & 0 & 0 & 0 \\ 0 & (-1)^{\sum_{i=0}^{n-1} x_{4i+2}} & 0 & 0 \\ 0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{4i+3}} & 0 \\ 0 & 0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{4i+4}} \end{pmatrix}$$

$$EOX_{4n}(x_{1,\ldots},x_{4n}) = EQUALITY_4\left(\sum_{i=0}^{n-1} x_{4i+1}, \sum_{i=0}^{n-1} x_{4i+2}, \sum_{i=0}^{n-1} x_{4i+3}, \sum_{i=0}^{n-1} x_{4i+4}\right),$$

$$where \sum is\ modulo\ 2$$

Finally, we measure the quantum state |0>. Error probability is no more than *p=1/4*.

All results could be observed using *Wolfram Mathematica* program for $EQUALITY_4$ Boolean function (*Appendix 2*) by defining more queries in the program body and adding them to the transformation sequence. □

For example, in case of 8 arguments two queries are:

$$Q_1 = \begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 \\ 0 & 0 & (-1)^{x_3} & 0 \\ 0 & 0 & 0 & (-1)^{x_4} \end{pmatrix}, Q_2 = \begin{pmatrix} (-1)^{x_5} & 0 & 0 & 0 \\ 0 & (-1)^{x_6} & 0 & 0 \\ 0 & 0 & (-1)^{x_7} & 0 \\ 0 & 0 & 0 & (-1)^{x_8} \end{pmatrix}$$

The algorithm outputs the correct result with the maximum error probability ¼ .

By definition of the function EOX_{4n}, $EOX_{4n}(000...0) = 1$. Changing only one bit value to 1 changes the function's value to 0. Consequently, the sensitivity of $EOX_{4n}(000...0)$ is equal to 4n, therefore $D(EOX_{4n}) = 4n$.

Lemma 3.2. *Given the general optimal randomized decision tree of depth n computing EOX_{4n} with n queries, the probability of getting the answer 1 is equal for each combination of randomly chosen n values.*

Proof. To prove this lemma it is necessary to divide 4n arguments of the function EOX_{4n} into 4 non intersecting subsets according to the structure of the function: $\{x_1,...,x_n\}$, $\{x_{n+1},...,x_{2n}\}$, $\{x_{2n+1},...,x_{3n}\}$, $\{x_{3n+1},...,x_{4n}\}$. All other steps of the proof repeat ones from the proof of Lemma 3.1. □

Theorem 3.15. *The Boolean function $EOX_{4n}(x_1,...,x_{4n})$ is computable by a randomized classical decision tree with n queries with the maximum probability p= ½.*

Proof. The proof repeats the proof of Theorem 3.13 very closely. The general form of an optimal randomized decision tree has n queries. According to Lemma 3.2, all tree's leafs with value 1 have the same probability, therefore:

$$Pr(1|X:EOX_{4n}(X)=1) = 4n\cdot(4n-1)\cdot...\cdot(3n+1)\cdot\frac{1}{4n\cdot(4n-1)\cdot...\cdot(3n+1)}q\cdot p = q\cdot p$$

$$Pr(0|X:EOX_{4n}(X)=0)$$

$$...=1-q+4n\cdot(4n-1)\cdot...\cdot(3n+1)\cdot\frac{1}{4n\cdot(4n-1)\cdot...\cdot(3n+1)}q\cdot(1-p)$$

$$...=1-q+q\cdot(1-p)=1-q\cdot p$$

The probability of the correct answer is the minimum of *Pr(0)* and *Pr(1)*, *min(qp, 1-qp)*. The highest probability is obtained, when both answers are equally probable. This means, *qp = 1 – qp*, then *qp = ½*. Consequently, the probability of the described algorithm is ½. □

3.3.3 Quantum Algorithm for An Extension of EQUALITY$_6$ Boolean function, EOX$_{6n}$

We define a 6n-argument Boolean function EOX_{6n} (short for *Equality Of XORs*):

$$EOX_{6n}(x_1,x_2,x_3,\underbrace{x_4,x_5,x_6,\ldots,x_{6n-5},x_{6n-4},x_{6n-3}}_{n-1},x_{6n-2},x_{6n-1},x_{6n}) = ...$$

$$...=EQUALITY_6(\underbrace{XOR(x_{6i+1})}_{i=0}^{n-1},\underbrace{XOR(x_{6i+2})}_{i=0}^{n-1},...,\underbrace{XOR(x_{6i+6})}_{i=0}^{n-1})$$

Theorem 3.16. *There is a bounded-error quantum query algorithm computing the Boolean function $EOX_{6n}(X)$ with probability p=9/16, that asks n quantum queries :*

$$Q_{9/16}(EOX_{6n})=n \quad .$$

Proof. The algorithm for the EOX_{6n} Boolean function is very similar to one that computes $EQUALITY_6$ function (see section 3.1.7 for details). Instead of one query matrix the algorirthm uses n matrices one by one. Each query matrix is of the form:

$$\forall i=0,\ldots,n-1:$$

$$Q_i=\begin{pmatrix} (-1)^{x_{6i+1}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & (-1)^{x_{6i+2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & (-1)^{x_{6i+3}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & (-1)^{x_{6i+4}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (-1)^{x_{6i+5}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & (-1)^{x_{6i+6}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} .$$

All other parts of the algorithm remain the same:

- the initial quantum state is $|0\rangle$,
- the final state is acquired by applying a transformation sequence $|\varphi\rangle=U_1\cdot Q_n\cdot\ldots\cdot Q_1\cdot U_0|0\rangle$ where U_0 and U_1 are exactly the same as in the section 3.1.7.

To prove that the described quantum algorithm computes the function, observe the product of all query matrices:

$$Q=\begin{pmatrix}
(-1)^{\sum_{i=0}^{n-1} x_{6i+1}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (-1)^{\sum_{i=0}^{n-1} x_{6i+2}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{6i+3}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{6i+4}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{6i+5}} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (-1)^{\sum_{i=0}^{n-1} x_{6i+6}} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

$$EOX_{6n}(x_1,\dots,x_{6n}) = EQUALITY_6\left(\sum_{i=0}^{n-1} x_{6i+1}, \sum_{i=0}^{n-1} x_{6i+2}, \dots, \sum_{i=0}^{n-1} x_{6i+6}\right),$$

where $\sum$ *is modulo* 2

Finally, we measure the quantum state |0>. The correct answer probability is *p=9/16*.

All results could be observed using *Wolfram Mathematica* program for $EQUALITY_6$ Boolean function (*Appendix 3*) by defining more queries in the program body and adding them to the transformation sequence. □

By definition of the function EOX_{6n}, $EOX_{6n}(000...0) = 1$. Changing only one bit value to 1 changes function's value to 0, consequently, sensitivity of $EOX_{6n}(000...0)$ is equal to 6n, therefore $D(EOX_{6n}) = 6n$.

Lemma 3.3. *Given the general optimal randomized decision tree of depth n computing the function EOX_{6n}, the probability of getting the answer 1 is the same for each combination of randomly chosen n values.*

Proof. To prove this lemma it is necessary to divide 6n arguments of the function EOX_{6n} into 6 non intersecting subsets according to the structure of the function: $\{x_1,...,x_n\}$, $\{x_{n+1},...,x_{2n}\}$, $\{x_{2n+1},...,x_{3n}\}$, $\{x_{3n+1},...,x_{4n}\}$, $\{x_{4n+1},...,x_{5n}\}$, $\{x_{5n+1},...,x_{6n}\}$. All other steps of the proof repeat ones from the proof of Lemma 3.1. □

Theorem 3.17. *The Boolean function $EOX_{6n}(x_1,...,x_{6n})$ is computable by a randomized classical decision tree with n queries with the maximum probability p= ½.*

Proof. The proof repeats the proof of the Theorem 3.13 very closely.

The general form of an optimal randomized decision tree has n queries. According to Lemma 3.3, all tree's leafs with the value 1 have the same probability, then

$$Pr(1|X:EOX_{6n}(X)=1)=6n\cdot(6n-1)\cdot\ldots\cdot(5n+1)\cdot\frac{1}{6n\cdot(6n-1)\cdot\ldots\cdot(5n+1)}q\cdot p=q\cdot p$$

$$Pr(0|X:EOX_{6n}(X)=0)$$
$$\ldots=1-q+6n\cdot(6n-1)\cdot\ldots\cdot(5n+1)\cdot\frac{1}{6n\cdot(6n-1)\cdot\ldots\cdot(5n+1)}q\cdot(1-p)$$
$$\ldots=1-q+q\cdot(1-p)=1-q\cdot p$$

The probability of the correct answer is the minimum of Pr(0) and Pr(1), *min(qp, 1-qp)*. The highest probability is obtained when both answers are equally probable, qp = 1 – qp, qp = ½, thus the probability of the described algorithm is ½.

□

3.3.4 Application of EOX Boolean Functions

All algorithms for EOX Boolean functions described in sections 3.3.1, 3.3.2, 3.3.3 are examples of very effective quantum algorithms. This section contains a list of useful Boolean functions formed on the basis of the EOX function of 3n, 4n or 6n arguments, which algorithms are constructed on the basis of the algorithm for EOX_{3n}, EOX_{4n} or EOX_{6n} respectively.

Let us define a set of Boolean functions and call it AOX (short for *And Of XORs*). Similarly to EOX set of functions, AOX_{2n} algorithm is built of AND_2 query algorithm by adding more queries in the body of the algorithm. The same stands for AOX_{3n} and AOX_{5n} algorithms based on algorithms for AND_3 and AND_5 correspondingly. All AOX-type algorithms have n queries, the structure of each query has to be the same as in its basic AND_k algorithm. The AOX_{kn} algorithm is in fact a shortened EOX_{kn} algorithm – one argument in each of n queries is fixed as a constant value, so all complexity estimations could be proved using the same approaches.

For example, having some particular requirements to the function, it might be very useful to restrict $XOR(x_1,\ldots,x_n)=XOR(x_{n+1},\ldots,x_{2n})=XOR(x_{2n+1},\ldots,x_{3n})=0$ in case of 3n arguments. This could be done using the definition of the AOX_{3n} function as a basis. As a result we are not supposed to think out a specific quantum algorithm for the problem, but already have a quantum query algorithm which has a prooved

number of queries.

Let us note that any of our AND function algorithms are easily transformed to All-Zeroes function (function is equal to 1 if all input bits are zeroes, otherwise 0) by setting a constant of each query to zero, not one.

The general algorithm for EOX or AOX gives an opportunity to define a similar, but very specific Boolean function for particular n arguments with guaranteed query complexity of the algorithm and its error probability. We can get a new function by reducing the number of arguments of the basic function (EOX_{kn} or $AOX_{(k-1)n}$, k from {3, 4, 6}) by setting some of them by constants, or by asking the value of the same argument in different queries. This function might have no scientific value, but it is required at the specific moment for the specific purpose, so we have to be quick enough in producing the algorithm.

Example 1. Code verification, a short example: each bit of the input string is repeated twice, the output string is checked for pairs of bits to know if all data is received without losses or changes. It helps to discover a low quality transfer channel, which can be applied in cryptography.

AOX_6 and AOX_{10} are exactly such functions based on AND_3 and AND_5, both algorithm quantum complexity is 2:

- $AOX_6 = AND_3(XOR(x_1,x_2), XOR(x_3,x_4), XOR(x_5,x_6))$ is equal to 1, if and only if $x_1=x_2$ & $x_3=x_4$ & $x_5=x_6$. The query algorithm for the function AOX_6 is the EOX_8 query algorithm with the 4th and the 8th arguments set to *0*
- $AOX_{10}=AND_5(XOR(x_1,x_2), XOR(x_3,x_4), XOR(x_5,x_6), XOR(x_7,x_8), XOR(x_9,x_{10}))$ is equal to 1, if and only if $x_1=x_2$ & $x_3=x_4$ & $x_5=x_6$ & $x_7=x_8$ & $x_9=x_{10}$. The query algorithm for the function AOX_{10} is the EOX_{12} query algorithm with the 6th and the 12th arguments set to *0*

Example 2. Code verification, a more general approach: by AOX function definition, for example AOX_{3n}, all 3n arguments are divided into 3 non intersecting subsets of size n. The function has the value 1 if XOR_n of every subset are equal to each other and equal to 1. An expression $XOR(x_1,...,x_n) = 1$ is equivalent to an expression

$x_i = XOR(x_1,..., x_{i-1},x_{i+1},...,x_n,1)$ for an arbirary i. This could be used in verification of

data transmission: AOX_{3n} returns 1 if the string is sent correctly, asking only n queries with error probability ¼ .

Example 3. Define a function

$F(x_1,\ldots,x_7) = XOR(x_1,x_2,x_3)\ AND\ XOR(x_1,x_4,x_5)\ AND\ XOR(x_1,x_6,x_7)$.

The algorithm for this function is AOX_{3n} for n = 3 with query matrices

$$Q_1=\begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 \\ 0 & (-1)^{x_1} & 0 & 0 \\ 0 & 0 & (-1)^{x_1} & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, Q_2=\begin{pmatrix} (-1)^{x_2} & 0 & 0 & 0 \\ 0 & (-1)^{x_4} & 0 & 0 \\ 0 & 0 & (-1)^{x_6} & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

$$Q_3=\begin{pmatrix} (-1)^{x_3} & 0 & 0 & 0 \\ 0 & (-1)^{x_5} & 0 & 0 \\ 0 & 0 & (-1)^{x_7} & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Example 4. Define a function

$F(x_1,\ldots,x_{10}) = XOR(x_1 x_2,x_3)\ AND\ \ XOR(x_1,x_4,x_5)\ AND$

$XOR(x_6,x_7,x_8)\ AND\ \ XOR(x_6,x_9,x_{10})\ OR$

$NOT\ XOR(x_1 x_2,x_3)\ AND\ \ NOT\ XOR(x_1,x_4,x_5)\ AND$

$NOT\ XOR(x_6,x_7,x_8)\ AND\ \ NOT\ XOR(x_6,x_9,x_{10})$.

The algorithm for this function is EOX_{4n} algorithm for n = 3 with query matrices

$$Q_1=\begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 \\ 0 & (-1)^{x_1} & 0 & 0 \\ 0 & 0 & (-1)^{x_6} & 0 \\ 0 & 0 & 0 & (-1)^{x_6} \end{pmatrix}, Q_2=\begin{pmatrix} (-1)^{x_2} & 0 & 0 & 0 \\ 0 & (-1)^{x_4} & 0 & 0 \\ 0 & 0 & (-1)^{x_7} & 0 \\ 0 & 0 & 0 & (-1)^{x_9} \end{pmatrix},$$

$$Q_3=\begin{pmatrix} (-1)^{x_3} & 0 & 0 & 0 \\ 0 & (-1)^{x_5} & 0 & 0 \\ 0 & 0 & (-1)^{x_8} & 0 \\ 0 & 0 & 0 & (-1)^{x_{10}} \end{pmatrix}$$

3.4 Quantum Query Algorithm for 2n-bit AND Boolean Function: $Q_{3/4}(AND_{2n})=n$

Theorem 3.18. *There is a quantum query algorithm computing the Boolean function $AND_{2n}(x_1,...,x_{2n})$ with n quantum queries and maximum error probability p=1/4 :*

$$Q_{3/4}(\mathrm{AND}_{2n})=n$$

Proof. In fact, the algorithm is supposed to recognize the all-ones input string from any other string of the length 2n for the given n.

Quantum query description. The behavior of the quantum black box of this section differs from other algorithms' queries. Let us introduce two internal reference tables placed inside the black box , that are used in transforming the quantum state before the query. The size of both tables depends on n.

The first reference table (Table 3.2) consists of 2^n rows - all possible n-bit strings. Index k is assigned to each table row, being in fact the same number in decimal notation as the binary string contained in the current row.

Row index	Value
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table 3.2. The first reference table for AND_{2n} algorithm, an example for n = 3

The second internal reference table is a 2^n x 2^n Hadamard matrix.

Inner mechanism of the black box.

1. The black box inspects both n-bit input strings and discovers two indeces from the first reference table in parallel: k_1 – the index of the first n-bit string, k_2 – the index of

the last n-bit string.

2. The black box changes signs of the first 2^n amplitudes of the quantum state according to signs of the k_1-th column of 2^n x 2^n Hadamard matrix, and changes signs of the last 2^n amplitudes of the quantum state according to signs of the k_2-th column of 2^n x 2^n Hadamard matrix.

Let us note, that every Hadamard matrix's row/column has an even number of positive/negative elements. This property has a great impact on the result.

Algorithm description. The algorithm uses $2^{(n+1)}$ -qubit quantum system, the initial state is $\frac{1}{\sqrt{2^{n+1}}}\sum_{i=0}^{2^{n+1}-1}|i\rangle$. After the query we apply a unitary transformation, which is $2^{(n+1)}$ x $2^{(n+1)}$ Hadamard matrix, and then we measure the quantum state $|2^n-1\rangle$.

- In case k_1 and k_2 both are equal to some k, the quantum state before the measurement is $|k\rangle$

- In case k_1 and k_2 are not equal, quantum state before the measurement is $\frac{1}{2}(|k_1\rangle+|k_2\rangle+|2^n+k_1\rangle-|2^n+k_2\rangle)$, that follows from the properties of the Hadamard matrix and the way the current black box works. Suppose, $k_1 = 2^n-1$ and some other k_2, for example $k_2=0$, then the quantum state is $\frac{1}{2}(|2^n-1\rangle+|0\rangle+|2^{n+1}-1\rangle-|2^n\rangle)$. After the measurement we have the state $|2^n-1\rangle$ with probability ¼. The function has value 0 on such an input, therefore we recognize it as an output *0* with probability $p=1/4$. The same error probability holds for $k_2 = 2^n-1$ and some other k_1. In cases when both k_1 and k_2 are not equal to 2^n-1, after the measurement the system is in the state $|2^n-1\rangle$ with probability $p = 0$.

Thus, for the all-ones input string we have $k_1=k_2=2^n-1$. We measure the state $|2^n-1\rangle$ and obtain the value 1 with probability $p = 1$, otherwise algorithm outputs 0 with error probability $p = 1/4$. □

3.5 Other Bounded-Error Quantum Algorithms

3.5.1 Quantum Algorithm for "All Zeroes or Single One" 6 Argument Boolean Function

We define a 6-argument Boolean function AZSO (*All Zeroes or Single One*) equal to 1 on vectors 000000 and 000001, 000010, 000100, 001000, 010000, 100000, as follows from the name, otherwise the function is equal to 0.

Theorem 3.19. *There is a bounded-error quantum query algorithm computing the Boolean function $AZSO_6$ $(x_1,...,x_6)$ with one quantum query and the correct answer probabilty p = 9/16:* $Q_{9/16}(AZSO_6)=1$.

Proof. The algorithm for $AZSO_6$ Boolean function uses 3-qubit quantum system with basis states $\{|0\rangle,|1\rangle,|2\rangle,|3\rangle,|4\rangle,|5\rangle,|6\rangle,|7\rangle\}$.

Define a unitary matrix U_0 as a 8x8 Hadamard matrix, and U_1 by

$$U_0=\frac{1}{\sqrt{8}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix},$$

$$U_1=\frac{1}{\sqrt{8}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \end{pmatrix}$$

respectively. Define a query matrix Q by

$$Q=\begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & (-1)^{x_3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & (-1)^{x_4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (-1)^{x_5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & (-1)^{x_6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

We begin in the state $|\varphi_0\rangle=|0\rangle$, apply U_0, then Q, and then U_1. Finally, we perform the measurement consisting of a projection onto the state $|0\rangle$ and its orthogonal complement. If the output is the state $|0\rangle$, we output 1, otherwise 0. Consequently, the final quantum state, a state after applying a transformation sequence on the initial quantum state, is $|\varphi\rangle=U_1\cdot Q\cdot U_0|0\rangle$.

We claim that the sequence $V_x=U_1\cdot Q\cdot U_0$ leaves $|\varphi_0\rangle$ unchanged up to the phase factor, when the input string has all bits equal, otherwise maps $|\varphi_0\rangle$ into a subspace orthogonal to $|\varphi_0\rangle$.

First, $U_0|0\rangle=\frac{1}{\sqrt{8}}\sum_{i=0}^{7}|i\rangle$, since U_0 is Hadamard matrix. For the input string *x=000000 Q* is the identity matrix, thus

$$V_{000000}|0\rangle=U_1\cdot I\cdot U_0|0\rangle=\begin{pmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix}^T.$$

For other input strings, for example, for *x = 100000*

$$Q\cdot U_0|0\rangle=\frac{1}{\sqrt{8}}(-|0\rangle+|1\rangle+|2\rangle+|3\rangle+|4\rangle+|5\rangle+|6\rangle+|7\rangle)$$

$$V_{100000}|0\rangle=\begin{pmatrix}\frac{3}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{2} & 0 & 0 & 0\end{pmatrix}^T,$$

which is the quantum state $|0\rangle$ after the measurement, with probability 9/16.

Running the algorithm on any other input vector, except for those vectors with all zeroes but one one, results in quantum state orthogonal to $|0\rangle$ with error probability no more than ¼, proving our claim.

See *Appendix 4* for other input and output vector correspondence. □

Theorem 3.20. *The Boolean function $AZSO_6(x_1,...,x_6)$ is computable by a randomized classical decision tree with one query with the maximum probability $p = ½$.*

Proof. The general form of the optimal randomized decision tree is shown in Fig. 3.7. Let us denote the probability of an answer $r \in \{0,1\}$ after execution of the algorithm on the input string X by Pr(r|X).

Then: $Pr(1|X = 000000 \cup X = 100000 ... \cup X = 000001) = qp$

$Pr(0|X \neq 000000 \cup ... \cup X \neq 000001) = 1 - qp$

The probability of the correct answer is *min(qp, 1-qp)*. The highest probability is obtained in case when both answers are equally probable, $qp = 1 - qp$, $qp = ½$, consequently, the probability of the described algorithm is ½. □

Unfortunately, no interesting extensions of this algorithm have been found.

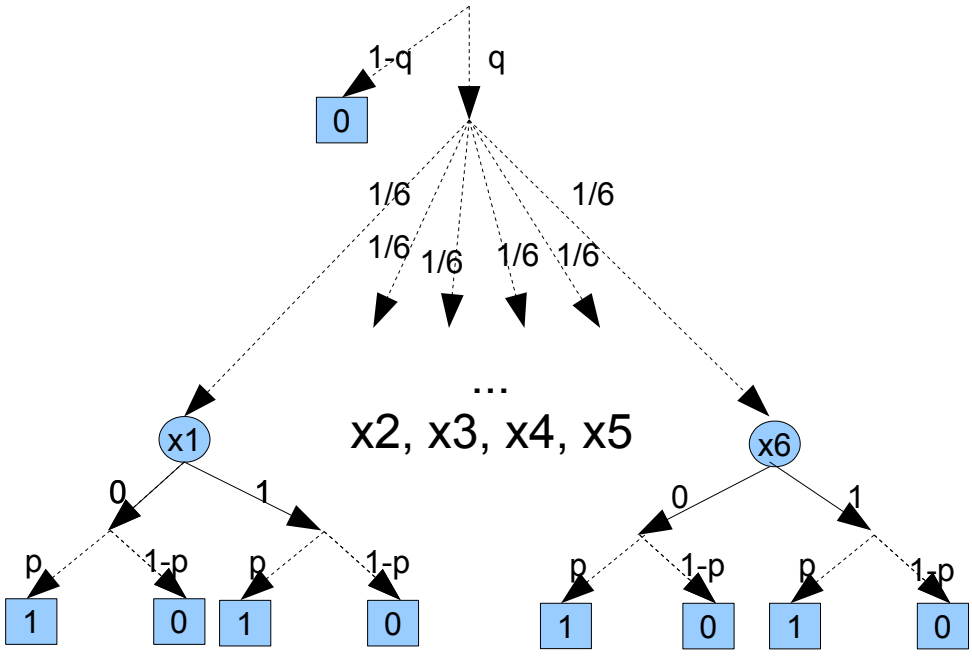


Fig. 3.7 The general form of the optimal classical randomized decision tree for computing $AZSO_6(x_1,...,x_6)$

3.6 Conclusion and Open Problems

This section contains a wide range of bounded-error quantum query algorithms. The most interesting are algorithms for conjunctions – for particular n = 2, 3, 5 using one query with correct answer probability $\frac{9}{10}$, $\frac{3}{4}$ and $\frac{9}{16}$ correspondingly, and a common algorithm for conjunction of 2n bits with n queries and error probability $\frac{1}{4}$.

Unfortunately, bounded-error algorithms are hardly used as subroutines of more complex ones because of pushing down the correct answer probability of the whole algorithm. Nonetheless, bounded-error algorithms offer more freedom in designing approaches, and the upper bound of possible speedup in comparison with classical algorithm is never known.

The major further goal of this research is to work out approaches for step-by-step construction of an efficient query algorithm for an arbitrary Boolean function.

4 Quantum Query Algorithms for Multivalued Functions

This chapter is based on the following paper:

T. Miscenko-Slatenkova, A. Vasilieva. Computing Relations in the Quantum Query Model. *Scientific Papers, University of Latvia*, 2011 [VM11]

A binary relation is a more general type of problem than a function. A relation is a set of ordered pairs that associates values from a domain set with values from a range set. Difference from a function is in element mapping: each element from a domain set may be mapped to multiple elements from a range set. So, a function is simply a special case of a relation, where each value from a domain set is mapped to no more than one value from a range set. An alternative way is to consider relations as multivalued functions.

The study of query complexity of multifunctions has been inspired by the book on communication complexity by Kushilevitz and Nisan [KN97]. The main part of this book discusses communication complexity of functions, but Chapter 5 is devoted exactly to the communication complexity of relations.

We apply the traditional query model to compute multifunctions. In classical deterministic settings, however, it does not seem to be possible to employ the difference between multivalued and single-valued functions 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. A 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.

Significant difficulty in designing quantum query algorithm is making it exact (i.e. to output a correct result with probability $p = 1$ on all input strings). The largest complexity separation between the classical deterministic and the quantum exact query algorithm complexity for the same total function known for today is N versus $N/2$. However, in the case of a multifunction we are allowed to output values from a fixed set instead of one fixed value for the certain input. We assert 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 a quantum algorithm is more than two times less than it is required by a classical algorithm for the same computational problem.

We adapt the query model for computing multifunctions. First, we give the definitions related to multivalued functions. We define several types of query algorithms that may compute multifunctions in different manners. Then we demonstrate examples of computing multifunctions in classical and quantum query models, where a quantum algorithm achieves a speed-up in comparison with a classical algorithm. Finally, we discuss the chances of achieving good results in enlarging the complexity gap between the classical and the quantum query complexity for multifunctions.

4.1 Multivalued Functions

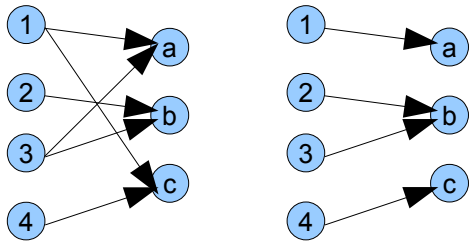
Definition 4.1. [We12] ***A multivalued function*** (also known as a multiple-valued function [Kn96], multifunction, many-valued function, set-valued function, set-valued map, multi-valued map, multimap, correspondence, carrier) *is a "function" that assumes two or more distinct values in its range for at least one point in its domain.*

Although these "functions" are not functions in the normal sense of being one-to-one or many-to-one, the usage is so common that there is no way to dislodge it. When considering multivalued functions, it is therefore necessary to refer to usual "functions" as single-valued functions.

While the trigonometric, hyperbolic, exponential, and integer power functions are all single-valued functions, their inverses are multivalued. For example, the function z^2 maps each complex number z to a well-defined number z^2, while its inverse function $\sqrt{z}$ maps, for example, the value $z = 1$ to $\sqrt{1}=\pm 1$. A unique principal value can be chosen for such functions (in this case, the principal square root is the positive one), the choices cannot be made continuous over the whole complex plane. Instead, lines of discontinuity must occur. The discontinuities of multivalued functions in the complex plane are commonly handled through the adoption of branch cuts, but use of Riemann surfaces is another possibility.

Multivalued function is the subject of this research. Using function's notation, multifunction's definition looks like $M(X):\{0,1\}^n \to \mathbb{N}$: the domain set consists of all possible binary strings of length n, $X=(x_1,x_2,...,x_n)$, each bit x_i of the input string is an argument of the multifunction, while the output – a result set $M(X)$ - is a subset of the range set $\mathbb{N}$.

We consider *left-total* multifunctions only, when the result set is not empty for each element from the domain set. A function is a special case of relation and it uniquely associates each value from the domain set with one value from the range set. Fig. 4.1 graphically demonstrates this difference.



Multifunction M *Function* $F \in \mathrm{Func}(M)$

Fig. 4.1 Example of a multivalued and a single-valued functions

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

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

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

The set *Func*(*M*) consists of four total functions:

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

4.2 Computing Multifunctions in a Query Model

Computation of functions in a query model is studied well enough, however, it is not obvious how to extend a query model to compute multifunctions. For the first

time this question has been discussed in [Va10].

We propose three different options to describe the way a query algorithm computes a relation and define three types of query algorithms based on these options.

Definition 4.2. [Va10] *Query algorithm computes multifunction M 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* $Q_D(M)$*.*

A type of a classical decision tree which computes a multifunction in a definite manner is a deterministic decision tree. In settings of the quantum model, the corresponding algorithm type is an exact quantum query algorithm.

Definition 4.3. [Va10] *Query algorithm computes multifunction M 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 incorrect value. Classical query complexity is denoted by* $C_{RD}(M)$*. Quantum query complexity is denoted by* $Q_{RD}(M)$*.*

This definition is a more natural and takes into account the essence of a 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 achieve the goal we need to bring a quantum system in such a superposition, where only basis states associated with values from result set have non-zero amplitude values. After the measurement quantum system collapses to one of these basis states with probability determined by its amplitude value.

Definition 4.4. [Va10] *Query algorithm computes multifunction M in a* ***uniformly distributed manner****, if for each X it outputs each value from the result set with equal probability and never outputs the 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.

However, in our opinion this definition is the most reasonable in a terms of computing a multifunction.

Each definition may be applied to solving specific real-world computational problems. Most of all we are interested in comparing complexity of computing multifunctions in the same manners in classical and quantum query models. Our goal is to analyze algorithm implementation special features and differences to produce examples with a large gap between the classical and the quantum query complexity.

4.3 Multifunction Example and Its Computation

In this section we present some results in designing efficient quantum query algorithms for computing multifunctions. Our approach is searching for interesting algorithms that would compute multifunctions with specific properties. In each example we demonstrate a quantum query algorithm for computing a specific multifunction and then prove the classical complexity lower bound.

4.3.1 Multifunction M: $Q_{UD}(M) = 1$ vs. $C_{UD}(M) = 3$

Table 4.5 defines a four-argument multifunction with Boolean domain set and four-valued range set: $M:\{0,1\}^4 \to \{1,2,3,4\}$.

X	M(X)	X	M(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}

Table 4.5 Definition of the multifunction M

Theorem 4.1. *There exists a quantum query algorithm, which computes the multifunction M with one query: $Q_{UD}(M) = 1$.*

Proof. The algorithm for computation of the multifunction M in the uniformly distributed manner with one query uses 2-qubit quantum system with basis states $\{|0\rangle, |1\rangle, |2\rangle, |3\rangle\}$. Define a unitary matrix U_0 and U_1 as 4x4 Hadamard matrices:

$$U_0 = U_1 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Define a query matrix Q as follows:

$$Q = \begin{pmatrix} (-1)^{x_1} & 0 & 0 & 0 \\ 0 & (-1)^{x_2} & 0 & 0 \\ 0 & 0 & (-1)^{x_3} & 0 \\ 0 & 0 & 0 & (-1)^{x_4} \end{pmatrix}$$

We begin in the state $|\varphi_0\rangle = |0\rangle$ and then apply U_0, then Q and then U_1.

Finally, we measure all basis states at once:

- if the output is $|i\rangle$, we output the answer $\{i+1\}$
- if the output is a superposition of all four basis states, we output $\{0, 1, 2, 3\}$

The final state $|\varphi\rangle = U_1 \cdot Q \cdot U_0 |0\rangle$ is a state after applying a transformation sequence $V_x = U_1 \cdot Q \cdot U_0$ on the initial quantum state. We claim that the sequence V_x changes the initial state according to the definition of the multifunction M.

To see that the claim is correct, note first that $U_0|0\rangle = \frac{1}{2}\sum_{i=0}^{3}|i\rangle$, since U_0 is a Hadamard matrix. For the input string $x=0000$ Q is the identity matrix, but for $x=1111$ Q is minus identity:

$$V_{0000}|0\rangle = U_1 \cdot I \cdot U_0 |0\rangle = |0\rangle, \; V_{1111}|0\rangle = U_1 \cdot (-I) \cdot U_0 |0\rangle = -|0\rangle .$$

For the input string $x=0011$ and $x=1100$ $Q \cdot U_0|0\rangle$ is $\frac{1}{2}(|0\rangle + |1\rangle - |2\rangle - |3\rangle)$ or $-\frac{1}{2}(|0\rangle + |1\rangle - |2\rangle - |3\rangle)$, respectively. Then, $V_{0011}|0\rangle = |3\rangle, V_{1100}|0\rangle = -|3\rangle$.

For the input string $x=0101$ and $x=1010$ $Q \cdot U_0|0\rangle$ is

$\frac{1}{2}(|0\rangle-|1\rangle+|2\rangle-|3\rangle)$ or $-\frac{1}{2}(|0\rangle-|1\rangle+|2\rangle-|3\rangle)$, respectively. Then, $V_{0101}|0\rangle=|2\rangle, V_{1010}|0\rangle=-|2\rangle$.

For the input string *x=0110* and *x=1001* $Q\cdot U_0|0\rangle$ is $\frac{1}{2}(|0\rangle-|1\rangle-|2\rangle+|3\rangle)$ or $-\frac{1}{2}(|0\rangle-|1\rangle-|2\rangle+|3\rangle)$, respectively. Then, $V_{0110}|0\rangle=|4\rangle, V_{1001}|0\rangle=-|4\rangle$.

For all input strings with Hamming weight 1 (1000, 0100, 0010, 0001) or their inverse, i.e. strings with Hamming weight 3, $V_X|0\rangle=\frac{1}{2}(\pm|0\rangle\pm|1\rangle\pm|2\rangle\pm|3\rangle)$. □

Theorem 4.2. $C_{UD}(M) = 3$.

Proof. The proof of this theorem consists of two steps. First, we show that it is impossible to build a classical randomized decision tree of depth $d = 2$, which computes M in a uniformly distributed manner. Second, we present a tree, which computes M using three queries.

Lemma 4.1. *It is impossible to build a classical randomized decision tree of depth d= 2, which computes M in a uniformly distributed manner,* $C_{UD}(M)\geq 3$.

Proof. Let us assume there exists a tree where all paths from root to leaves contain no more than two variables. When executing algorithm on the input $X = 0000$ result "1" has to be output with probability $p = 1$. It means that there exists a path from the root to a leaf with output value "1", which goes through some two variables: $x_A = 0$ and $x_B = 0$. This path is depicted in Fig. 4.2. The fact is that it is not possible to select A and B to avoid contradictions with other inputs.

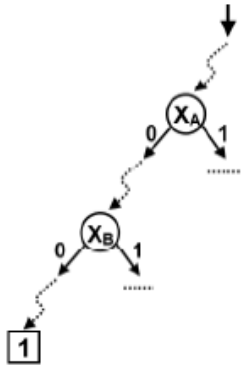


Fig. 4.2. Path for the input *X=0000* in a potential classical randomized decision tree of depth *d = 2* for computing M

For any choice of indices A and B there are four inputs that pass through the path depicted in Fig. 4.2 and finish in a leaf with output value "1". We denote $X_i = x_A x_B x_C x_D,\ A, B, C, D \in \{1,2,3,4\},\ A \neq B \neq C \neq D$, 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 4.5 it is easy to see that for any input with exactly two "1" a value set *M(X)* consists of exactly one output value, which is always different from "1". We obtained a contradiction: input X_4 passes through the path depicted in Fig. 4.2 and the algorithm outputs incorrect value "1". □

Lemma 4.2. *There exists a classical randomized decision tree, which computes M in a uniformly distributed manner using three queries.*

Proof. A classical probabilistic decision tree that computes M in a uniformly distributed manner is shown in Fig. 4.3. □

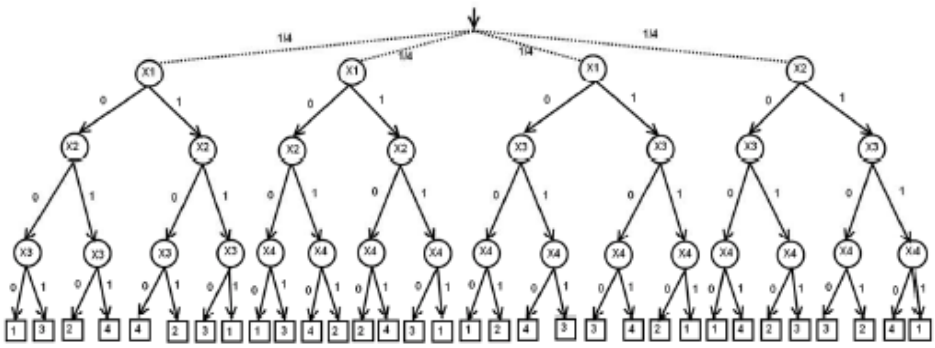


Fig. 4.3. The probabilistic decision tree that computes *M* in a uniformly distributed manner

4.3.2 The First Generalization of The Multifunction M

In this subsection, we generalize the multifunction *M* to the case of *4N* arguments. The generalized multifunction is

$$M' : \{0,1\}^{4N} \to \{1,2,3,4\} \quad .$$

Imagine that *4N* arguments are put on four vertical lines, *v-lines*, in such a way that: $\forall i \in \{0,\ldots,N-1\}, \forall k \in \{1,2,3,4\} : x_{4i+k}$ belongs to v-line number *k*. For example, $x_1, x_5, x_9, x_{13}, \ldots$ are placed on the 1st v-line, $x_2, x_6, x_{10}, x_{14}, \ldots$ - on the 2nd, and so on.

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

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

4. *M'(X)={4}*, if the 1st and the 4th v-lines of *X* have odd numbers of "1", and the 2nd and the 3rd have even numbers of "1", or vice versa: the 1st and the 4th – even, and the 2nd and the 3rd - odd. For example, input strings 00000110, 00001001, 00111010, 10011111 have the result set {4}.
5. In all other cases *M'(X)={1,2,3,4}*.

Theorem 4.3. $Q_{UD}(M')\le N$.

Proof. Quantum algorithm for the multifunction M' is very similar to the algorithm computing M in a uniformly distributed manner: we use a 2-qubit quantum system, apply a Hadamard operation to the initial quantum state |0> and apply one more Hadamard transformation right before the measurement. The difference is in the query between two Hadamard transformations: there are n query matrices applied to quantum state $U_0|0>$ one by one, each query Q_i is defined by the matrix

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

The product of all query matrices is equal to

$$\begin{pmatrix}(-1)^{r_1} & 0 & 0 & 0\\ 0 & (-1)^{r_2} & 0 & 0\\ 0 & 0 & (-1)^{r_3} & 0\\ 0 & 0 & 0 & (-1)^{r_4}\end{pmatrix},$$

$$r_1=\underset{i=0}{\overset{N-1}{XOR}}x_{4i+1},\ r_2=\underset{i=0}{\overset{N-1}{XOR}}x_{4i+2},$$

$$r_3=\underset{i=0}{\overset{N-1}{XOR}}x_{4i+3},\ r_4=\underset{i=0}{\overset{N-1}{XOR}}x_{4i+4}$$

Let us note, that r_i is equal to XOR over the i-th v-line's arguments. As discussed in the proof of Theorem 4.1,

- the answer {1} is obtained when $r_1r_2r_3r_4$ equals to 0000 or 1111. This is equivalent to the demand to have all v-lines with either odd or even number of 1's
- the answer {2}, when $r_1r_2r_3r_4$ is equal to 0101 or 1010. This is equivalent to the demand to have the 1st and the 3rd v-line's parity different from the the 2nd and the 4th v-line's parity.

- the answer {3}, when $r_1r_2r_3r_4$ is equal to 0011 or 1100. This is equivalent to the demand to have the 1st and the 2nd v-line's parity different from the the 3rd and the 4th v-line's parity.
- the answer {4}, when $r_1r_2r_3r_4$ is equal to 0110 or 1001. This is equivalent to the demand to have the 1st and the 4th v-line's parity different from the 2nd and the 3rd v-line's parity.
- the answer {1,2,3,4} is obtained, when $r_1r_2r_3r_4$ has exactly one or three 1's in the input string.

These statements fully repeats the definition of the multifunction M' . □

Theorem 4.4. $3N \le C_{UD}(M') \le 4N$.

Proof. Let us assume there exists a classical decision tree that computes the multifunction *M* by asking *3N-1* questions. We use the all-zeroes input *X = 0...0* to demonstrate a contradiction. Suppose we have queried arbitrary *3N-1* variables, *N+1* variables remain unquestioned.

On the *4N*-zeroes input *X = 0...0* the tree has to output the value "1" because all v-lines contain zero number of "1". Then, we consider only such inputs 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 and finishes in leaves with the 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. As we know, the result set in such a case is {2} or {3} or {4}. Thus, the algorithm outputs an incorrect value for this input. This fact contradicts with the initial assumption and implies $C_{UD}(M') \ge 3N$.

□

4.3.3 The Second Generalization of The Multifunction M

In this subsection we demonstrate the second way to generalize the multifunction M. In the previous generalization we added more queries. This time we extend the quantum system and put more variables in single query.

Suppose we are given a multifunction of *N* variables

$M'':\{0,1\}^N \to \{1,2,\ldots,N\}$, where N is a power of 2. We do not provide the full definition of the multifunction; it follows from properties of quantum algorithm described below. We just would like to demonstrate that such generalization is technically possible.

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

Theorem 4.5. *There exists a quantum query algorithm, which computes a specific multifunction M'' in a randomly distributed manner asking one question only.* $Q_{RD}(M'') = 1$.

Proof. The algorithm for M'' is analogous to one for M, we just add more qubits and arguments: given $N=2^k, k\in\mathbb{N}$, the quantum algorithm starts with k-qubit zero state $|0\rangle$ in a k-qubit quantum system, then applies a N x N Hadamard matrix, then runs a N-variable query, and finally applies a N x N Hadamard matrix once again.

The query matrix Q is as follows:

$$Q=\begin{pmatrix} (-1)^{x_1} & 0 & \ldots & 0 & 0 \\ 0 & (-1)^{x_2} & \ldots & 0 & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ 0 & 0 & \ldots & (-1)^{x_{N-1}} & 0 \\ 0 & 0 & \ldots & 0 & (-1)^{x_N} \end{pmatrix} .$$

Finally, we measure all basis states at once:

- if the output is $|i\rangle$, we output the answer $\{i+1\}$
- if the output is a superposition of all four basis states, we output $\{1, \ldots, N\}$

The final state $|\varphi\rangle = U_1 \cdot Q \cdot U_0 |0\rangle$ is a state after applying a transformation sequence $V_x = U_1 \cdot Q \cdot U_0$ on the initial quantum state. The definition of the multifunction M'' follows directly from the way the sequence V_x changes the initial state.

First, $U_0|0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$, since U_0 is a Hadamard matrix. For the input string $x=00\ldots0_N$ Q is the identity matrix, but for $x=11\ldots1_N$ Q is minus identity:

$V_X|0\rangle = \pm|0\rangle$, respectively. According to properties of Hadamard matrix, input strings with an odd Hamming weight end up in some state $|i\rangle, i \in \{0,...,N-1\}$, while input strings with an even Hamming weight end up in a state like $V_X|0\rangle = \frac{1}{\sqrt{N}}(\pm|0\rangle \pm |1\rangle ... \pm |N-1\rangle)$. □

Theorem 4.6. $\frac{N}{2}+1 \le C_{RD}(M'') \le N$.

Proof. Let us analyze a multifunction that is computable by the 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:

$$\alpha_1 = \frac{(-1)^{x_1} + (-1)^{x_2} + ... + (-1)^{x_N}}{N}$$

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

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

4.4 Conclusion and Open Problems

In this paper we continued the research initiated in [Va10] on computing multivalued instead of single-valued functions in a query model and presented new results. We presented three examples of computing multifunctions in different manners – in a uniformly distributed manner and in a randomly distributed manner. When computing multifunctions in either uniformly or randomly distributed manner, the error probability is also not allowed. In this regard the quantum query model appears to be more suitable for computing multifunctions.

Main results of this research sub-direction are the following:

- an algorithm for M_N with the quantum query complexity three times fewer than the classical query complexity for the same multifunction. This is an example of computing a multifunction in a uniformly distributed manner
- algorithm for M_N with one quantum query and the classical query complexity $\frac{N}{2}+1 \le C(M_N) \le N$. This is an example of computing a multifunction in a randomly distributed manner.

Discussing specifics of computing multifunctions in a definite manner led us to conclusion that the task of computing multifunctions in a distributed manner is more promising for enlarging classical and quantum query complexity gap.

Further work is to continue the research and to produce more interesting results regarding computing multifunctions in a query model. We are planning to analyze and generalize current examples to develop a general approach for designing efficient quantum query algorithms for computing multifunctions. Important work direction is to develop efficient techniques for proving complexity lower bounds for computing multifunctions in a classical query model.

5 Low Degree Boolean Functions

This chapter is based on the following paper:

T. Miscenko-Slatenkova, A. Dubrovska. 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, [DM06]

In this part of the thesis we study the complexity of quantum query algorithms computing the value of Boolean function and its relation to the degree of algebraic polynomial representing this function. We pay special attention to Boolean functions with quantum query algorithm complexity lower than the deterministic one.

Definition 5.1. *The degree of Boolean function is the degree of the representing polynomial and is denoted as* ***deg(f)***.

We have the following lower and upper bound estimations for the degree of the representing polynomial, which is unique for particular Boolean function:

$$deg(f) \le D(f) \quad \text{[NS02] and} \quad deg(f) \ge 2Q_E(f) \quad \text{[BW02].}$$

A wish to improve the best ever achieved complexity of a quantum algorithm means to leave the sign "less" instead of "less or equal" in the expression $Q_E(f) \le \frac{D(f)}{2}$ for a Boolean function *f*. This is true for those functions *f* with *deg*(*f*)<*D*(*f*). This shows the importance of functions with low polynomial degree in terms of quantum computing, unfortunately, there are few examples of quantum algorithms to illustrate theoretical evaluation of the complexity. Here we describe polynomials with relation between the number of variables and the degree of the polynomial greater than 2 times in the best case.

The main purpose of this work was to design new approaches to construct Boolean functions with a large gap between the deterministic complexity and the degree of the representing polynomial.

5.1 Definition of A Low Degree Polynomial

To define a polynomial we have to pass the following two steps:

1. we present a polynomial of degree 2 with non-Boolean range of values
2. choose an appropriate polynomial and transfer the non-Boolean range of values to {0,1}.

It is therefore obvious, that we solve the problem from the other side - we find out a polynomial with Boolean values, and then define a Boolean function by matching each possible input with the corresponding value of the previously defined polynomial. Unfortunately, the problem is not trivial, so the design of a polynomial for a given Boolean function remains one of the directions for our future work.

5.1.1 A Polynomial of Degree 2

For the first step we define some set *S* and then define a polynomial of degree 2. To define a set *S* we split *N* variables in three non-empty sets and choose pairs of 1-valued variables.

Restrictions for *S* are the following:

- variables in each pair should come from different sets
- if variables x and y are from sets *I* and *J,* and a pair *(x,y)* is in *S*, then there is no other x_0 from *I* or y_0 from *J*, such that (x_0,y) or (x,y_0) are in *S*

The polynomial is $p(x_{1,}\ldots,x_N)=\sum_{i\in[N]} x_i - \sum_{i,j:i\neq j,(i,j)\in S} x_i x_j$,

where

- *S* is a specially defined set of pairs of numbers from a given range [1..*N*]
- $\sum_{i\in[N]} x_i = |x|$ is the Hamming weight of the input x
- $\sum_{i,j:i\neq j,(i,j)\in S} x_i x_j$ is the number of pairs where $x_i=x_j=1$and (i,j) belongs to *S*

Assume that N variables are divided into three sets with n_1, n_2 and n_3 variables

correspondingly: $N=n_1+n_2+n_3$. Let $n_{max}=max(n_1, n_2, n_3)$. Then the polynomial obeys the restriction:

Lemma 5.1. $0\le p(x_1,\dots,x_N)\le n_{max}$

Proof. Let us take an arbitrary input *x*, such that $|x|=k=k_1+k_2+k_3$,

where k_1 is the number of „1" in the first set, correspondingly, k_2 and k_3 are the numbers of "1" in other two, $k_1\ge k_2\ge k_3$. Take the smallest of the three, k_3, it is the number of pairs corresponding set has with each of two other, $2k_3$ in all. Then, a set with k_2 "1" has k_2 pairs with the largest group. Thus, number of pairs is $\sum_{i,j:i\ne j,(i,j)\in S} x_i x_j = 2k_3+k_2$ for an arbitrary x. As far as $\sum_{i\in[n]} x_i = k_1+k_2+k_3=|x|$, the polynomial p(x) value is restricted from both left and right:

$p(x)=(k_1+k_2+k_3)-(2k_3+k_2)=k_1-k_3$
$0\le k_1-k_3\le k_1\le n_{max}$. □

5.1.2 A Polynomial With The Boolean Value Range

For the second step we define a polynomial with the Boolean value range: for any odd *k* there is a polynomial of degree *(k-1)* that transforms a set {0,…,*k*} to {0,1}, which is known from algebra course.

5.1.3 Examples and The General Form Of The Approach

Example 1.

An advantageous example is a polynomial $p(x_1,\dots,x_9)=\sum_{i\in[9]} x_i - \sum_{i,j:i\ne j,(i,j)\in S} x_i x_j$,

where variables are divided into 3 sets, so $0\le p(x)\le 3$, and *S* contains 9 pairs, for example,

$S=(x_1,x_4),(x_1,x_9),(x_2,x_5),(x_2,x_8),(x_3,x_6),(x_3,x_7),(x_4,x_9),(x_5,x_8),(x_6,x_7)$.

The polynomial p then is equal to

$$p(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) = ...$$
$$... = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 - ...$$
$$... - (x_1 \cdot x_4 + x_1 \cdot x_9 + x_2 \cdot x_5 + x_2 \cdot x_8 + x_3 \cdot x_6 + x_3 \cdot x_7 + x_4 \cdot x_9 + x_5 \cdot x_8 + x_6 \cdot x_7)$$

Next we find an appropriate polynomial p_b of degree 2 that would transform the set {0,1,2,3} to {0,1}, for example , $p_b(z) = \frac{1}{2}z^2 - \frac{3}{2}z + 1$, $p_b(0) = p_b(3) = 1$, $p_b(1) = p_b(2) = 0$.

Hence, the polynomial $p_b(p(x))$ has Boolean values and the degree $deg(p_b(p(x))) = 4$.

Finally, we define a Boolean function

$$f_9(x_1, ..., x_9) = p_b(p(x_1, ..., x_9)) \quad .$$

The degree of the polynomial is a product of the degrees of p and p_b, $deg(f_9)=4$.

For the input |x|=0 the value of the polynomial is $p_b(p(x))=1$. Flipping any zero variable to 1 will change the polynomial's value to 0. Hence, $D(f_9)=9$.

See *Appendix 5* for the truth table for the Boolean function $f_9(x_1,...,x_9)$.

Example 2.

To show another example of low-degree function, let us take a polynomial $p(x_1,...,x_{21})$ with the range of values {0,1,2,3,4,5,6,7}:

polynomial $p(x_1, ..., x_{21}) = \sum_{i \in [21]} x_i - \sum_{i,j: i \neq j, (i,j) \in S} x_i x_j$,

where variables are divided into 3 sets, so $0 \le p(x) \le 7$,

and S contains of 21 pairs, for example,

$$(x_1, x_8), (x_2, x_9), (x_3, x_{10}), (x_4, x_{11}), (x_5, x_{12}), (x_6, x_{13}), (x_7, x_{14}),$$
$$(x_8, x_{15}), (x_9, x_{16}), (x_{10}, x_{17}), (x_{11}, x_{18}), (x_{12}, x_{19}), (x_{13}, x_{20}), (x_{14}, x_{21}), \quad .$$
$$(x_1, x_{15}), (x_2, x_{16}), (x_3, x_{17}), (x_4, x_{18}), (x_5, x_{19}), (x_6, x_{20}), (x_7, x_{21})$$

There is a polynomial of degree 6 that we will use to transfer a set {0,1,2,3,4,5,6,7} to {0,1},

$$p_b(x) = -\frac{1}{144}x^6 + \frac{7}{48}x^5 - 1\frac{19}{144}x^4 + 3\frac{15}{16}x^3 - 5\frac{31}{36}x^2 + 2\frac{11}{12}x \quad .$$

$p_b(p(x))$ values are from the range {0,1}. For the corresponding Boolean function

$f_{21}(x) = p_b(p(x))$: $deg(f_{21})$=12. $D(f_{21})$=21 by sensitivity on the input x: |x|=0.

Let us generalize the idea described earlier. We will enlarge the number of variables, but still divide them in 3 groups. It appears that the best results come from the case of $N = n+n+n$ variable functions (n variables in each set), but it is not forbidden to define S assimmetrically.

General form of this method is formulated as

Lemma 5.2. *For each odd* $k>1$ *there exists a 3k-variable Boolean function f with D(f)=3k and deg(f)=2(k-1).*

Proof. Follows directly from the definition of the Boolean function f: the representing polynomial is obtained as described in sections 5.1.1 and 5.1.2, the the function is defined by its representing polynomial.

5.2 Tripple Function Method

Example 1. A 12-variable Boolean function $f_{12}(x_1,...,x_{12})$ with $D(f_{12})$=12 and $deg(f_{12})$=6.

First, define a 4-variable polynomial of degree 3:

$$p_4(x_1, x_2, x_3, x_4)=(x_1x_2+x_2x_3+x_3x_4+x_1x_4)-(x_1x_2x_3+x_1x_2x_4+x_1x_3x_4+x_2x_3x_4)$$

$p_4(x)$ range of values is {0,1}. Next, define a polynomial

$p_{12}(x_1,\dots,x_{12})=p_4(x_1,x_2,x_3,x_4)+p_4(x_5,x_6,x_7,x_8)+p_4(x_9,x_{10},x_{11},x_{12})$ with values from the range {0,1,2,3}. The last step is to choose an appropriate polynomial of degree 2 to transform elements from the set {0,1,2,3} to {0,1}. For example,

$S(z)=\frac{1}{2}z^2-\frac{3}{2}z+1$. Then $deg(S(p_{12}(x)))$=6. Finally, define the corresponding Boolean function $f_{12}(x)=S(p_{12}(x))$:

$D(f_{12})$=12 by its block sensitivity on the input x, |x|=12, s(f(x))=12.

A function of the similar form has been described in [OFI05]. Here we will gneralize the approach described in the previous example to the case of N variables.

Take *N=3r* variables polynomial, where we subdivide all variables into 3 sets each containing *r* variables. Then we define a polynomial

$P_{3r}(x_1,...,x_r,x_{r+1},...,x_{2r},x_{2r+1},...,x_{3r})=P_r(x_1,...,x_r)+P_r(x_{r+1},...,x_{2r})+P_r(x_{2r+1},...,x_{3r})$.

Lemma 5.3. *For each odd k>1 and for each t>1 there exists* $3^{t+1}\cdot k$ *-variable Boolean function f:* $D(f)=3^{t+1}\cdot k$ *and* $deg(f)=2^{t+1}\cdot(k-1)$.

Proof.

Step 1. Choose an odd *k* and define a polynomial according to Lemma 5.2. in the form $\begin{array}{l} p_2\,[deg(p_2)=2]:\{0,1\}^{3k}\to\{0,\dots,k\} \\ p_{k-1}\,[deg(p_{k-1})=(k-1)]:\{0,\dots,k\}\to\{0,1\} \end{array}$.

Its degree equals to *2(k-1)* while the number of variables is *3k*.

Step 2. Define *3r*-variable polynomial (variables are divided in 3 sets of *r* variables):

$P(x_1,...,x_r,x_{r+1},...,x_{2r},x_{2r+1},...,x_{3r})=P_r(x_1,...,x_r)+P_r(x_{r+1},...,x_{2r})+P_r(x_{2r+1},...,x_{3r})$, where P_r has Boolean values and the degree *d*. *P(x)* has a range of values {0,1,2,3} that can be transformed to {0,1} by an appropriate polynomial of the degree 2, thus *deg(P) =2d*. This type of a polynomial can be iterated, as a result we get $deg(P)=2^t\cdot d$ and the number of variables $3^t\cdot r$ for any *t>0.*

Step 3.Take *r=3k* and *d=2(k-1)*. The Boolean function *f(x)* represented by *P(x)* has $D(f)=3^t\cdot 3k=3^{t+1}k$ and $deg(f)=2^t\cdot 2(k-1)=2^{t+1}(k-1)$. □

Lemma 5.3. leads to the definition of a *9n*-variable Boolean function *f* with *D(f) = 9n* and *deg(f) = 4n-4* for *n>1*; a *27n*-variable Boolean function *f* with *D(f) = 27n* and *deg(f) = 8n-8* for *n>1* and so forth.

5.3 Conclusion and Open Problems

The aim of our future work in this area is to create quantum query algorithms for described functions with advantages over classical counterparts; and to be able to design a low-degree polynomial for a given Boolean function.

6 Conclusion

In the conclusion we will summarize all the items that were discussed in the thesis. The work was held in three directions:

- quantum query algorithms for Boolean functions
- quantum query algorithms for multivallued functions
- low degree Boolean functions

In the chapter devoted to algoritghms for Boolean functions we presented a wide range of bounded-error quantum query algorithms. It was of interest to find a "good" algorithm for conjunctions of several bits and the following results were achieved:

- a one-query quantum algorithm for conjunction of two bits with correct answer probability $p = \frac{9}{10}$
- a one-query quantum algorithm for conjunction of three bits with correct answer probability $p = \frac{3}{4}$
- a one-query quantum algorithm for conjunction of five bits with correct answer probability $p = \frac{9}{16}$

Moreover, algorithms mentioned above are transformed from algorithms for bit equality Boolean function: each of them preserves the same correct answer probability and asks correspondingly 3, 4 and 6 arguments in a single quantum query.

$EQUALITY_k$ Boolean function (k is the number of arguments) described previously is generalized to EOX_{kn} (Equality Of k n-argument XOR's) function. As for the algorithm, we just added n-1 more queries to the initial $EQUALIY_k$ algorithm saving the correct answer probability. Using the same contributions, we defined AOX_{kn} (AND Of k n-argument XOR's) Boolean function and computing the function quantum query algorithm.

One more specific quantum algorithm for the conjunction of 2n bits using n quantum queries with error probability $\frac{1}{4}$ is presented in this chapter.

Although we did not come to our major goal, which is working out approaches for step-by-step algorithm design for an arbitrary function, we hope that the steps we have made are to the right direction.

In the chapter devoted to multivalued functions we considered computing a multivalued function in a query model. The main achievements are the following:

- an algorithm for a multifunction M_N with quantum query complexity three times fewer than classical query complexity for the same multifunction
- algorithm for a multifunction M_N with one quantum query and classical query complexity $\frac{N}{2}+1 \le C(M_N) \le N$

Obviously, there is an additional piece of work to prove complexity lower bounds for computing multifunctions in a classical query model.

In the chapter devoted to low degree Boolean functions we discussed the degree of algebraic polynomial representing a Boolean function and its relation to the complexity of a quantum query algorithm computing the function. We presented some approaches for definition of Boolean functions with a large gap between deterministic complexity and degree of representing polynomial. Polynomials described have an attractive relation between the number of variables and the degree of the polynomial, which is greater than 2 times in the best case.

We hope that algorithm examples presented in the thesis demonstrate some useful approaches of algorithm design, and will inspire newcomers of this area. However, we believe that serious improvements of results achieved during the research are possible.

First of all, an idea to use a larger quantum system for algorithms computing either Boolean functions or multifunctions seems to be promising.

Secondly, to analyze real-life problems, especially those with inborn symmetry: quantum query algorithms have a certain advantage for symmetric functions, that comes from the properties of quantum query algorithms.

Thirdly, to develop some useful tools for easy programmatic analysis of a function and further construction of an efficient quantum query algorithm.

And finally, imagine a situation when using a bounded-error algorithm as a subroutine in a complex algorithm is required. The aim is to work out an approach to reduce losses of the correct answer probability of the whole complex algorithm. This would make bounded-error algorithms to be even more popular subject of researches.

Bibliography

[BW02] H. Buhrman, R. de Wolf. Complexity Measures and Decision Tree Complexity: A Survey. *Theoretical Computer Science,* v. 288(1), pp.21–43, 2002.

[Sh97] P. Shor. Polynomial time algorithms for prime factorization and discrete logarithmson a quantum computer. *SIAM Journal on Computing,* 26(5), pp.1484-1509, 1997.

[Gr01] L. Grover . From Schrödinger's equation to quantum search algorithm. *American Journal of Physics,* 69(7): 769-777, 2001.

[Am04] A. Ambainis. Quantum query algorithms and lower bounds (survey article). *In Proceedings of FOTFS III, Trends on Logic,* vol. 23, pp. 15-32, 2004.

[KLM07] R. Kaye, R. Laflamme, M. Mosca. An Introduction to Quantum Computing. *Oxford*, 2007.

[VM10] A. Vasilieva, T. Mischenko-Slatenkova. Quantum Query Algorithms for Conjunctions. *Proc. of the 9th International Conference UC 2010, Lecture Notesin Computer Science, Springer Berlin / Heidelberg,* vol. 6079/2010, ISBN: 978-3-642-13522-4, pp. 140-151, 2010.

[ACRSZ10] A. Ambainis, A.M. Childs, B.W. Reichardt, R. Spalek, S. Zhang. Any AND-OR Formula of Size N Can Be Evaluated in Time $N^{1/2+o(1)}$ on a Quantum Computer. *SIAM J. Comput. Volume 39, Issue 6,* pp. 2513-2530, 2010.

[KN97] E. Kushilevitz, N. Nisan. Communication complexity. *Cambridge UniversityPress*, 1997.

[Am02] A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences,* vol. 64(4), p.750-767, 2002.

[Am98] A. Ambainis. Polynomial degree vs. quantum query complexity. *Proceedings of FOCS 98,* p.230-240, 1998.

[Am11] A. Ambainis. Personal communication, 2011.

[CEMM98] R. Cleve, A. Ekert, C. Macchiavello, M. Mosca. Quantum algorithms revisited. *Proc. of the Royal Society of London,* vol. A 454, pp.339–354, 1998.

[Pa94] C. Papadimitriou. Computational Complexity. *Addison-Wesley, Rading,* 500pp., 1994.

[De85] D. Deutch. Quantum theory, the Church-Turing principle and theuniversal quantum computer. *In proc. of the Royal Society, London, A400:97{117*, 1985.

[Gr96] L. Grover. A fast quantum mechanical algorithm for database search. *In Proceedings of28th STOC'96,* pp. 212. –219, 1996.

[DJ92] D. Deutsch, R. Jozsa. Rapid solutions of problems by quantum computation. *Proc. ofthe Royal Society of London,* vol. A 439, pp.553-558, 1992.

[Si94] I. Simon. String matching algorithms and automata. *Lecture Notes in ComputerScience,* vol. 814, pp.386-395, 1994.

[FI09] R. Freivalds, K. Iwama. Quantum Queries on Permutations with a Promise. *Implementation and Application of Automata, Lecture Notes in Computer Science,* vol. 5642/2009, pp.208-216, 2009.

[Mi07] T. Miscenko-Slatenkova. Quantum Query Algorithms for Computation of Boolean Functions. *Master Thesis*, 2007.

[Wo01] R. de Wolf. Quantum Computing and Communication Complexity. *University of Amsterdam*, 2001.

[KW04] R. Kerenidis, R. de Wolf. Exponential Lower Bound for 2-Query Locally DecodableCodes via a Quantum Argument. *Journal of Computer and System Sciences,* pp.395-420, 2004.

[VM11] A.Vasilieva, T. Mischenko-Slatenkova. Computing Relations in the Quantum Query Model. *Scientific Papers, University of Latvia,* Volume 770, Computer Science andInformation Technologies, ISBN 978-9984-45-377-4, pp. 68-89, 2011.

[We12] Eric W. Weisstein. Multivalued Function. *MathWorld - A Wolfram Web Resource*, 2012.

[Kn96] K. Knopp. Multiple-Valued Functions. *Section II in Theory of Functions Parts I and II, Two Volumes Bound as One. New York: Dover,* Part I p. 103 and Part II pp. 93-146, 1996.

[Va10] A. Vasilieva. Quantum Query Algorithms for Relations. Randomized and Quantum Computation. *MFCS and CSL 2011 satellite workshop,* ISBN 978-80-87342-08-4, pp. 78-89, 2010.

[DM06] 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.

[NS02] N. Nisan, M. Szegedi. On the degree of Boolean functions as real polynomials. *Computational complexity,* v. 288(1): 21-43, 2002.

[OFI05] R.Ozols, R. Freivalds, J.Ivanovs et al. Boolean functions with a low polynomial degree and quantum query algorithms. *Proc. of SOFSEM 2005*, 2005.

Appendix 1. Computation in Wolfram Mathematica. Programs for $EQUALITY_3(x_1,x_2,x_3)$ and $AND_2(x_1,x_2)$ computation

```
(*Definition of unitary transformations*)
H4=1/2{{1,1,1,1},
{1,-1,1,-1},
{1,1,-1,-1},
{1,-1,-1,1}};
U={
{1/Sqrt[10],1/Sqrt[10],Sqrt[2/5],Sqrt[2/5]},
{1/2,-1/2,1/2,-1/2},
{Sqrt[2/5],Sqrt[2/5],-1/Sqrt[10],-Sqrt[2/5]},
{0,1/Sqrt[2],0,-1/Sqrt[2]}
};

(*The algorithm for EQUALITY(x1,x2,x3)*)
START={1,0,0,0};
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)^X1,0,0},
{0,0,(-1)^X2,0},
{0,0,0,(-1)^X3}};
RESULT=U.QUERY1.H4.START;
Print[X1,X2,X3," | ",RESULT];
];];];
(*output*)
000 | {3/Sqrt[10],0,1/(2 Sqrt[10]),0}
001 | {1/Sqrt[10],1/2,Sqrt[5/2]/2,1/Sqrt[2]}
010 | {1/Sqrt[10],-(1/2),3/(2 Sqrt[10]),0}
011 | {-(1/Sqrt[10]),0,7/(2 Sqrt[10]),1/Sqrt[2]}
100 | {1/Sqrt[10],0,-(7/(2 Sqrt[10])),-(1/Sqrt[2])}
101 | {-(1/Sqrt[10]),1/2,-(3/(2 Sqrt[10])),0}
110 | {-(1/Sqrt[10]),-(1/2),-(Sqrt[(5/2)]/2),-(1/Sqrt[2])}
111 | {-(3/Sqrt[10]),0,-(1/(2 Sqrt[10])),0}
```

```
(*The algorithm for AND(x1,x2)*)
START={1,0,0,0};
For[X1=0,X1<=1,X1++,
For[X2=0,X2<=1,X2++,
QUERY1={{(-1)^X1,0,0,0},{0,(-1)^X1,0,0},{0,0,(-1)^X2,0},{0,0,0,-
    1}};
RESULT=U.QUERY1.H4.START;
Print[X1,X2," | ",RESULT];
];];
(*output*)
00 | {1/Sqrt[10],1/2,Sqrt[5/2]/2,1/Sqrt[2]}
01 | {-(1/Sqrt[10]),0,7/(2 Sqrt[10]),1/Sqrt[2]}
10 | {-(1/Sqrt[10]),1/2,-(3/(2 Sqrt[10])),0}
11 | {-(3/Sqrt[10]),0,-(1/(2 Sqrt[10])),0}
```

Appendix 2. Computation in Wolfram Mathematica. Programs for $EQUALITY_4(x_1,x_2,x_3,x_4)$ and $AND_3(x_1,x_2,x_3)$ computation

```
(*Definition of unitary transformations*)
H4=1/2{{1,1,1,1},
{1,-1,1,-1},
{1,1,-1,-1},
{1,-1,-1,1}};
 U={
   {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}};

(*The algorithm for EQUALITY(x1,x2,x3,x4)*)
START={1,0,0,0};
For[X1=0,X1<=1,X1++,
  For[X2=0,X2<=1,X2++,
    For[X3=0,X3<=1,X3++,
      For[X4=0,X4<=1,X4++,
        QUERY1={
          {(-1)^X1,0,0,0},
          {0,(-1)^X2,0,0},
          {0,0,(-1)^X3,0},
          {0,0,0,(-1)^X4}};
        RESULT=U.QUERY1.H4.START;
        Print[X1,X2,X3,X4," | ",RESULT];
        ];];];];

(*output*)
0000 | {1,0,0,0}
0001 | {1/2,-(1/2),1/2,-(1/2)}
0010 | {1/2,1/2,1/2,1/2}
0011 | {0,0,1,0}
0100 | {1/2,1/2,-(1/2),-(1/2)}
0101 | {0,0,0,-1}
0110 | {0,1,0,0}
0111 | {-(1/2),1/2,1/2,-(1/2)}
```

```
1000 | {1/2,-(1/2),-(1/2),1/2}
1001 | {0,-1,0,0}
1010 | {0,0,0,1}
1011 | {-(1/2),-(1/2),1/2,1/2}
1100 | {0,0,-1,0}
1101 | {-(1/2),-(1/2),-(1/2),-(1/2)}
1110 | {-(1/2),1/2,-(1/2),1/2}
1111 | {-1,0,0,0}

(*The algorithm for AND(x1,x2,x3)*)
START={1,0,0,0};
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)^X3,0},
        {0,0,0,-1}};
      RESULT=U.QUERY1.H4.START;
      Print[X1,X2,X3," | ",RESULT];
      ];];];
(*output*)
000 | {1/2,-(1/2),1/2,-(1/2)}
001 | {0,0,1,0}
010 | {0,0,0,-1}
011 | {-(1/2),1/2,1/2,-(1/2)}
100 | {0,-1,0,0}
101 | {-(1/2),-(1/2),1/2,1/2}
110 | {-(1/2),-(1/2),-(1/2),-(1/2)}
111 | {-1,0,0,0}
```

Appendix 3. Computation in Wolfram Mathematica. Programs for $EQUALITY_6(x_1,...,x_6)$ and $AND_5(x_1,...,x_5)$ computation

```
(*Definition of unitary transformations*)
(* 8 x 8 Hadamard matrix*)
H8=1/Sqrt[8]{
    {1,1,1,1,1,1,1,1},
    {1,-1,1,-1,1,-1,1,-1},
    {1,1,-1,-1,1,1,-1,-1},
    {1,-1,-1,1, 1,-1,-1,1},
    {1,1,1,1,-1,-1,-1,-1},
    {1,-1,1,-1,-1,1,-1,1},
    {1,1,-1,-1,-1,-1,1,1},
    {1,-1,-1,1,-1,1,1,-1}
    };
U=1/(2 Sqrt[2]){
    {1,1,1,1,1,1,1,-1},
    {1,1,-1,-1,-1,-1,1,-1},
    {1,1,1,1,-1,-1,-1,1},
    {-1,-1,1,1,-1,-1,1,-1},
    {2,-2,0,0,0,0,0,0},
    {0,0,0,0,2,-2,0,0},
    {0,0,2,-2,0,0,0,0},
    {0,0,0,0,0,0,2,2}};
(*Query matrix defined as a function*)
F[x1_,x2_,x3_,x4_,x5_,x6_]:={
{(-1)^x1,0,0,0,0,0,0,0},
{0,(-1)^x2,0,0,0,0,0,0},
{0,0,(-1)^x3,0,0,0,0,0},
{0,0,0,(-1)^x4,0,0,0,0},
{0,0,0,0,(-1)^x5,0,0,0},
{0,0,0,0,0,(-1)^x6,0,0},
{0,0,0,0,0,0,1,0},
{0,0,0,0,0,0,0,1}
};
(*The algorithm for EQUALITY(x1,...,x6)*)
START={1,0,0,0,0,0,0,0};
For[X1=0,X1<=1,X1++,
```

```
  For[X2=0,X2<=1,X2++,
    For[X3=0,X3<=1,X3++,
      For[X4=0,X4<=1,X4++,
        For[X5=0,X5<=1,X5++,
          For[X6=0,X6<=1,X6++,
            FQ=F[X1,X2,X3,X4,X5,X6];
            RESULT=U.FQ.H8.START;
            Print[X1,X2,X3,X4,X5,X6," | ",RESULT];
            ];];];];];];
(*output*)
000000 | {3/4,-(1/4),1/4,-(1/4),0,0,0,1/2}
000001 | {1/2,0,1/2,0,0,1/2,0,1/2}
000010 | {1/2,0,1/2,0,0,-(1/2),0,1/2}
000011 | {1/4,1/4,3/4,1/4,0,0,0,1/2}
000100 | {1/2,0,0,-(1/2),0,0,1/2,1/2}
000101 | {1/4,1/4,1/4,-(1/4),0,1/2,1/2,1/2}
000110 | {1/4,1/4,1/4,-(1/4),0,-(1/2),1/2,1/2}
000111 | {0,1/2,1/2,0,0,0,1/2,1/2}
001000 | {1/2,0,0,-(1/2),0,0,-(1/2),1/2}
001001 | {1/4,1/4,1/4,-(1/4),0,1/2,-(1/2),1/2}
001010 | {1/4,1/4,1/4,-(1/4),0,-(1/2),-(1/2),1/2}
001011 | {0,1/2,1/2,0,0,0,-(1/2),1/2}
001100 | {1/4,1/4,-(1/4),-(3/4),0,0,0,1/2}
001101 | {0,1/2,0,-(1/2),0,1/2,0,1/2}
001110 | {0,1/2,0,-(1/2),0,-(1/2),0,1/2}
001111 | {-(1/4),3/4,1/4,-(1/4),0,0,0,1/2}
010000 | {1/2,-(1/2),0,0,1/2,0,0,1/2}
010001 | {1/4,-(1/4),1/4,1/4,1/2,1/2,0,1/2}
010010 | {1/4,-(1/4),1/4,1/4,1/2,-(1/2),0,1/2}
010011 | {0,0,1/2,1/2,1/2,0,0,1/2}
010100 | {1/4,-(1/4),-(1/4),-(1/4),1/2,0,1/2,1/2}
010101 | {0,0,0,0,1/2,1/2,1/2,1/2}
010110 | {0,0,0,0,1/2,-(1/2),1/2,1/2}
010111 | {-(1/4),1/4,1/4,1/4,1/2,0,1/2,1/2}
011000 | {1/4,-(1/4),-(1/4),-(1/4),1/2,0,-(1/2),1/2}
011001 | {0,0,0,0,1/2,1/2,-(1/2),1/2}
011010 | {0,0,0,0,1/2,-(1/2),-(1/2),1/2}
011011 | {-(1/4),1/4,1/4,1/4,1/2,0,-(1/2),1/2}
011100 | {0,0,-(1/2),-(1/2),1/2,0,0,1/2}
```

```
011101 | {-(1/4),1/4,-(1/4),-(1/4),1/2,1/2,0,1/2}
011110 | {-(1/4),1/4,-(1/4),-(1/4),1/2,-(1/2),0,1/2}
011111 | {-(1/2),1/2,0,0,1/2,0,0,1/2}
100000 | {1/2,-(1/2),0,0,-(1/2),0,0,1/2}
100001 | {1/4,-(1/4),1/4,1/4,-(1/2),1/2,0,1/2}
100010 | {1/4,-(1/4),1/4,1/4,-(1/2),-(1/2),0,1/2}
100011 | {0,0,1/2,1/2,-(1/2),0,0,1/2}
100100 | {1/4,-(1/4),-(1/4),-(1/4),-(1/2),0,1/2,1/2}
100101 | {0,0,0,0,-(1/2),1/2,1/2,1/2}
100110 | {0,0,0,0,-(1/2),-(1/2),1/2,1/2}
100111 | {-(1/4),1/4,1/4,1/4,-(1/2),0,1/2,1/2}
101000 | {1/4,-(1/4),-(1/4),-(1/4),-(1/2),0,-(1/2),1/2}
101001 | {0,0,0,0,-(1/2),1/2,-(1/2),1/2}
101010 | {0,0,0,0,-(1/2),-(1/2),-(1/2),1/2}
101011 | {-(1/4),1/4,1/4,1/4,-(1/2),0,-(1/2),1/2}
101100 | {0,0,-(1/2),-(1/2),-(1/2),0,0,1/2}
101101 | {-(1/4),1/4,-(1/4),-(1/4),-(1/2),1/2,0,1/2}
101110 | {-(1/4),1/4,-(1/4),-(1/4),-(1/2),-(1/2),0,1/2}
101111 | {-(1/2),1/2,0,0,-(1/2),0,0,1/2}
110000 | {1/4,-(3/4),-(1/4),1/4,0,0,0,1/2}
110001 | {0,-(1/2),0,1/2,0,1/2,0,1/2}
110010 | {0,-(1/2),0,1/2,0,-(1/2),0,1/2}
110011 | {-(1/4),-(1/4),1/4,3/4,0,0,0,1/2}
110100 | {0,-(1/2),-(1/2),0,0,0,1/2,1/2}
110101 | {-(1/4),-(1/4),-(1/4),1/4,0,1/2,1/2,1/2}
110110 | {-(1/4),-(1/4),-(1/4),1/4,0,-(1/2),1/2,1/2}
110111 | {-(1/2),0,0,1/2,0,0,1/2,1/2}
111000 | {0,-(1/2),-(1/2),0,0,0,-(1/2),1/2}
111001 | {-(1/4),-(1/4),-(1/4),1/4,0,1/2,-(1/2),1/2}
111010 | {-(1/4),-(1/4),-(1/4),1/4,0,-(1/2),-(1/2),1/2}
111011 | {-(1/2),0,0,1/2,0,0,-(1/2),1/2}
111100 | {-(1/4),-(1/4),-(3/4),-(1/4),0,0,0,1/2}
111101 | {-(1/2),0,-(1/2),0,0,1/2,0,1/2}
111110 | {-(1/2),0,-(1/2),0,0,-(1/2),0,1/2}
111111 | {-(3/4),1/4,-(1/4),1/4,0,0,0,1/2}

(*The algorithm for EQUALITY(x1,...,x5)*)
START={1,0,0,0,0,0,0,0};
For[X1=0,X1<=1,X1++,
```

```
  For[X2=0,X2<=1,X2++,
    For[X3=0,X3<=1,X3++,
      For[X4=0,X4<=1,X4++,
        For[X5=0,X5<=1,X5++,
          FQ=F[X1,X2,X3,X4,X5,1];
          RESULT=U.FQ.H8.START;
          Print[X1,X2,X3,X4,X5," | ",RESULT];
          ];];];];];
(*output*)
00000 | {1/2,0,1/2,0,0,1/2,0,1/2}
00001 | {1/4,1/4,3/4,1/4,0,0,0,1/2}
00010 | {1/4,1/4,1/4,-(1/4),0,1/2,1/2,1/2}
00011 | {0,1/2,1/2,0,0,0,1/2,1/2}
00100 | {1/4,1/4,1/4,-(1/4),0,1/2,-(1/2),1/2}
00101 | {0,1/2,1/2,0,0,0,-(1/2),1/2}
00110 | {0,1/2,0,-(1/2),0,1/2,0,1/2}
00111 | {-(1/4),3/4,1/4,-(1/4),0,0,0,1/2}
01000 | {1/4,-(1/4),1/4,1/4,1/2,1/2,0,1/2}
01001 | {0,0,1/2,1/2,1/2,0,0,1/2}
01010 | {0,0,0,0,1/2,1/2,1/2,1/2}
01011 | {-(1/4),1/4,1/4,1/4,1/2,0,1/2,1/2}
01100 | {0,0,0,0,1/2,1/2,-(1/2),1/2}
01101 | {-(1/4),1/4,1/4,1/4,1/2,0,-(1/2),1/2}
01110 | {-(1/4),1/4,-(1/4),-(1/4),1/2,1/2,0,1/2}
01111 | {-(1/2),1/2,0,0,1/2,0,0,1/2}
10000 | {1/4,-(1/4),1/4,1/4,-(1/2),1/2,0,1/2}
10001 | {0,0,1/2,1/2,-(1/2),0,0,1/2}
10010 | {0,0,0,0,-(1/2),1/2,1/2,1/2}
10011 | {-(1/4),1/4,1/4,1/4,-(1/2),0,1/2,1/2}
10100 | {0,0,0,0,-(1/2),1/2,-(1/2),1/2}
10101 | {-(1/4),1/4,1/4,1/4,-(1/2),0,-(1/2),1/2}
10110 | {-(1/4),1/4,-(1/4),-(1/4),-(1/2),1/2,0,1/2}
10111 | {-(1/2),1/2,0,0,-(1/2),0,0,1/2}
11000 | {0,-(1/2),0,1/2,0,1/2,0,1/2}
11001 | {-(1/4),-(1/4),1/4,3/4,0,0,0,1/2}
11010 | {-(1/4),-(1/4),-(1/4),1/4,0,1/2,1/2,1/2}
11011 | {-(1/2),0,0,1/2,0,0,1/2,1/2}
11100 | {-(1/4),-(1/4),-(1/4),1/4,0,1/2,-(1/2),1/2}
11101 | {-(1/2),0,0,1/2,0,0,-(1/2),1/2}
```

```
11110 | {-(1/2),0,-(1/2),0,0,1/2,0,1/2}
11111 | {-(3/4),1/4,-(1/4),1/4,0,0,0,1/2}
```

Appendix 4. Computation in Wolfram Mathematica. Program for computing $AZSO_6(x_1,...,x_6)$

```
(*Definition of unitary transformations*)
(* 8 x 8 Hadamard matrix*)
H8=1/Sqrt[8]{
{1,1,1,1,1,1,1,1},
{1,-1,1,-1,1,-1,1,-1},
{1,1,-1,-1,1,1,-1,-1},
{1,-1,-1,1, 1,-1,-1,1},
{1,1,1,1,-1,-1,-1,-1},
{1,-1,1,-1,-1,1,-1,1},
{1,1,-1,-1,-1,-1,1,1},
{1,-1,-1,1,-1,1,1,-1}
};
U=1/(2 Sqrt[2]){
    {1,1,1,1,1,1,1,1},
    {1,1,-1,-1,-1,-1,1,1},
    {1,1,1,1,-1,-1,-1,-1},
    {-1,-1,1,1,-1,-1,1,1},
    {2,-2,0,0,0,0,0,0},
    {0,0,0,0,2,-2,0,0},
    {0,0,0,0,0,0,2,-2},
    {0,0,2,-2,0,0,0,0}};

(*Query matrix defined as a function*)
F[x1_,x2_,x3_,x4_,x5_,x6_,x7_,x8_]:={
{(-1)^x1,0,0,0,0,0,0,0},
{0,(-1)^x2,0,0,0,0,0,0},
{0,0,(-1)^x3,0,0,0,0,0},
{0,0,0,(-1)^x4,0,0,0,0},
{0,0,0,0,(-1)^x5,0,0,0},
{0,0,0,0,0,(-1)^x6,0,0},
{0,0,0,0,0,0,(-1)^x7,0},
{0,0,0,0,0,0,0,(-1)^x8}
};

(*The algorithm for AZSO(x1,...,x6)*)
START={1,0,0,0,0,0,0,0};
```

```
For[X1=0,X1<=1,X1++,
For[X2=0,X2<=1,X2++,
For[X3=0,X3<=1,X3++,
For[X4=0,X4<=1,X4++,
For[X5=0,X5<=1,X5++,
For[X6=0,X6<=1,X6++,
FQ=F[X1,X2,X3,X4,X5,X6,0,0];
RESULT=U.FQ.H8.START;
Print[X1,X2,X3,X4,X5,X6," | ",RESULT];
];];];];];];
```

```
000000 | {1,0,0,0,0,0,0,0}
000001 | {3/4,1/4,1/4,1/4,0,1/2,0,0}
000010 | {3/4,1/4,1/4,1/4,0,-(1/2),0,0}
000011 | {1/2,1/2,1/2,1/2,0,0,0,0}
000100 | {3/4,1/4,-(1/4),-(1/4),0,0,0,1/2}
000101 | {1/2,1/2,0,0,0,1/2,0,1/2}
000110 | {1/2,1/2,0,0,0,-(1/2),0,1/2}
000111 | {1/4,3/4,1/4,1/4,0,0,0,1/2}
001000 | {3/4,1/4,-(1/4),-(1/4),0,0,0,-(1/2)}
001001 | {1/2,1/2,0,0,0,1/2,0,-(1/2)}
001010 | {1/2,1/2,0,0,0,-(1/2),0,-(1/2)}
001011 | {1/4,3/4,1/4,1/4,0,0,0,-(1/2)}
001100 | {1/2,1/2,-(1/2),-(1/2),0,0,0,0}
001101 | {1/4,3/4,-(1/4),-(1/4),0,1/2,0,0}
001110 | {1/4,3/4,-(1/4),-(1/4),0,-(1/2),0,0}
001111 | {0,1,0,0,0,0,0,0}
010000 | {3/4,-(1/4),-(1/4),1/4,1/2,0,0,0}
010001 | {1/2,0,0,1/2,1/2,1/2,0,0}
010010 | {1/2,0,0,1/2,1/2,-(1/2),0,0}
010011 | {1/4,1/4,1/4,3/4,1/2,0,0,0}
010100 | {1/2,0,-(1/2),0,1/2,0,0,1/2}
010101 | {1/4,1/4,-(1/4),1/4,1/2,1/2,0,1/2}
010110 | {1/4,1/4,-(1/4),1/4,1/2,-(1/2),0,1/2}
010111 | {0,1/2,0,1/2,1/2,0,0,1/2}
011000 | {1/2,0,-(1/2),0,1/2,0,0,-(1/2)}
011001 | {1/4,1/4,-(1/4),1/4,1/2,1/2,0,-(1/2)}
011010 | {1/4,1/4,-(1/4),1/4,1/2,-(1/2),0,-(1/2)}
011011 | {0,1/2,0,1/2,1/2,0,0,-(1/2)}
```

011100 | {1/4,1/4,-(3/4),-(1/4),1/2,0,0,0}
011101 | {0,1/2,-(1/2),0,1/2,1/2,0,0}
011110 | {0,1/2,-(1/2),0,1/2,-(1/2),0,0}
011111 | {-(1/4),3/4,-(1/4),1/4,1/2,0,0,0}
100000 | {3/4,-(1/4),-(1/4),1/4,-(1/2),0,0,0}
100001 | {1/2,0,0,1/2,-(1/2),1/2,0,0}
100010 | {1/2,0,0,1/2,-(1/2),-(1/2),0,0}
100011 | {1/4,1/4,1/4,3/4,-(1/2),0,0,0}
100100 | {1/2,0,-(1/2),0,-(1/2),0,0,1/2}
100101 | {1/4,1/4,-(1/4),1/4,-(1/2),1/2,0,1/2}
100110 | {1/4,1/4,-(1/4),1/4,-(1/2),-(1/2),0,1/2}
100111 | {0,1/2,0,1/2,-(1/2),0,0,1/2}
101000 | {1/2,0,-(1/2),0,-(1/2),0,0,-(1/2)}
101001 | {1/4,1/4,-(1/4),1/4,-(1/2),1/2,0,-(1/2)}
101010 | {1/4,1/4,-(1/4),1/4,-(1/2),-(1/2),0,-(1/2)}
101011 | {0,1/2,0,1/2,-(1/2),0,0,-(1/2)}
101100 | {1/4,1/4,-(3/4),-(1/4),-(1/2),0,0,0}
101101 | {0,1/2,-(1/2),0,-(1/2),1/2,0,0}
101110 | {0,1/2,-(1/2),0,-(1/2),-(1/2),0,0}
101111 | {-(1/4),3/4,-(1/4),1/4,-(1/2),0,0,0}
110000 | {1/2,-(1/2),-(1/2),1/2,0,0,0,0}
110001 | {1/4,-(1/4),-(1/4),3/4,0,1/2,0,0}
110010 | {1/4,-(1/4),-(1/4),3/4,0,-(1/2),0,0}
110011 | {0,0,0,1,0,0,0,0}
110100 | {1/4,-(1/4),-(3/4),1/4,0,0,0,1/2}
110101 | {0,0,-(1/2),1/2,0,1/2,0,1/2}
110110 | {0,0,-(1/2),1/2,0,-(1/2),0,1/2}
110111 | {-(1/4),1/4,-(1/4),3/4,0,0,0,1/2}
111000 | {1/4,-(1/4),-(3/4),1/4,0,0,0,-(1/2)}
111001 | {0,0,-(1/2),1/2,0,1/2,0,-(1/2)}
111010 | {0,0,-(1/2),1/2,0,-(1/2),0,-(1/2)}
111011 | {-(1/4),1/4,-(1/4),3/4,0,0,0,-(1/2)}
111100 | {0,0,-1,0,0,0,0,0}
111101 | {-(1/4),1/4,-(3/4),1/4,0,1/2,0,0}
111110 | {-(1/4),1/4,-(3/4),1/4,0,-(1/2),0,0}
111111 | {-(1/2),1/2,-(1/2),1/2,0,0,0,0}

Appendix 5. A Truth Table for the Boolean function $f_9(x_1,\ldots,x_9)$

The function $f_9(x_1,\ldots,x_9)$ is defined in section 5.1.3. Here we list those inputs on which the function's value is 1, on all other inputs the value is zero.

000000000	000000111	000001011	000001111	000010101	000010111
000011001	000011011	000011101	000011101	000011111	000100110
000100111	000101010	000101011	000101110	000101111	000110100
000110101	000110110	000110111	000111000	000111001	000111010
000111011	000111100	000111101	000111110	000111111	001000011
001000111	001001011	001001100	001010001	001010011	001010101
001010111	001011001	001011011	001100010	001100011	001100110
001100111	001101010	001101011	001110000	001110001	001110010
001110011	001110100	001110101	001110110	001110111	001111000
001111001	001111010	001111011	010000101	010000111	010001001
010001011	010001101	010001101	010010010	010010101	010011001
010011101	010100100	010100101	010100110	010100111	010101000
010101001	010101010	010101011	010101100	010101101	010101110
010101111	010110100	010110101	010111000	010111001	010111100
010111101	011000001	011000011	011000101	011000111	011001001
011001011	011010001	011010101	011011001	011011110	011100000
011100001	011100010	011100011	011100100	011100101	011100110
011100111	011101000	011101001	011101010	011101011	011110000
011110001	011110100	011110101	011111000	011111001	100000110
100000111	100001010	100001011	100001110	100001111	100010100
100010101	100010110	100010111	100011000	100011001	100011010
100011011	100011100	100011101	100011110	100011111	100100001
100100110	100101010	100101110	100110100	100110110	100111000
100111010	100111100	100111110	101000010	101000011	101000110

101000111	101001010	101001011	101010000	101010001	101010010
101010011	101010100	101010101	101010110	101010111	101011000
101011001	101011010	101011011	101100010	101100110	101101010
101101101	101110000	101110010	101110100	101110110	101111000
101111010	110000100	110000101	110000110	110000111	110001000
110001001	110001010	110001011	110001100	110001101	110001110
110001111	110010100	110010101	110011000	110011001	110011100
110011100	110100100	110100110	110101000	110101010	110101100
110101110	110110011	110110100	110111000	110111100	111000000
111000001	111000010	111000011	111000100	111000101	111000110
111000111	111001000	111001001	111001010	111001011	111010000
111010001	111010100	111010101	111011000	111011001	111100000
111100010	111100100	111100110	111101000	111101010	111110000
111110100	111111000	111111111			

Table app5.1. Boolean function $f_9(x_1,\ldots,x_9)$: inputs for the function's value 1