SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2010. Vol. 756
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 227–247 P.

Quantum Algorithms for Computing the Boolean Function *AND* and Verifying Repetition Code

Alina Vasilieva[1]
Faculty of Computing, University of Latvia
Raina bulv. 29, LV-1459, Riga, Latvia
Alina.Vasilieva@gmail.com

Quantum algorithms can be analyzed in a query model to compute Boolean functions. Function input is provided in a black box, and the aim is to compute the function value using as few queries to the black box as possible. In this paper we present two quantum algorithms. The first algorithm computes the Boolean function *AND* of two bits using one query with a probability p=4/5. It is also described how to extend this algorithm to compute $AND(f_1,f_2)$, where f_1 and f_2 are arbitrary Boolean functions. The second algorithm can be used for verification of the repetition code for error detection. A repetition code is an error detection scheme that repeats each bit of the original message r times. After a message with redundant bits is transmitted via a communication channel, it must be verified. The verification procedure can be interpreted as an application of a query algorithm, where input is a message to be checked. Classically, for an N-bit message, values of all N variables must be queried. We present an exact quantum algorithm that uses only N/2 queries in the case when r=2.

Keywords: quantum computing, quantum query algorithms, complexity theory, Boolean functions, algorithm design.

1 Introduction

Quantum computing is an exciting alternative way of computation based on the laws of quantum mechanics. This branch of computer science is developing rapidly; various computational models exist, and this is a study of one of them.

Let $f(x_1,x_2,...,x_N):\{0,1\}^N \to \{0,1\}$ be a Boolean function. We consider the black box model (also known as the query model), where a black box contains the input $X=(x_1,x_2,...,x_N)$ and can be accessed by querying x_i values. The goal is to compute the value of the function. The complexity of a query algorithm is measured by the number of questions it asks. The classical version of this model is known as *decision trees* [1]. This computational model is widely applicable in software engineering. For instance, a database can be considered a black box, and, to speed up application performance, the goal is to reduce the number of database queries.

Quantum query algorithms can solve certain problems faster than classical algorithms. The best known and at the same time the simplest exact quantum algorithm for a total Boolean function was designed for the *XOR* function with N/2

[1] This research is supported by the European Social Fund project No. 2009/0138/1DP/1.1.2.1.2/09/IPIA/VIAA/004, No. ESS2009/77.

questions versus N questions required by classical algorithm [2]. The quantum query model differs from the quantum circuit model [2, 3, 4], and algorithm construction techniques for this model are less developed. The problem of quantum query algorithm construction is very significant. Although there are many lower-bound and upper-bound estimations of quantum query algorithm complexity [2, 5, 6, 7], there are very few examples of original quantum query algorithms.

This paper consists of two parts, in which algorithm construction results for two different computational problems are presented.

In the first part of this paper, we consider computing the Boolean function *AND*. First, we demonstrate a bounded-error quantum query algorithm, which computes Boolean function $AND(x_1, x_2) = x_1 \wedge x_2$ with one query and probability $p = 4/5$. This is better than the best possible classical probabilistic algorithm, where the probability to obtain correct result is $p = 2/3$. Then we extend our approach and formulate a general method for computing a composite Boolean function $AND(f_1,f_2)$, where f_1 and f_2 are arbitrary Boolean functions. In particular, we explicitly show how an N-variable Boolean function $AND_N(x_1,...,x_N) = x_1 \wedge x_2 \wedge ... \wedge x_N$ can be computed by the quantum bounded-error algorithm with a probability p=4/5 using N/2 queries.

In the second part of this paper, we present an exact quantum query algorithm for resolving a specific problem. The task is to verify a codeword message that has been encoded using the *repetition code* for detecting errors [8] and has been transmitted across a communication channel. The considered repetition code simply duplicates each bit of the message. The verification procedure can be considered to be an application of a query algorithm, where the codeword to be checked is contained in a black box. To verify the message in the classical way, we would need to access all bits. That is, for a codeword of length N, all N queries to the black box would be required. We present an exact quantum query algorithm that requires only N/2 queries.

An exact algorithm always produces a correct answer with 100% probability. Another variation is to use a bounded-error model, where an error margin $\varepsilon < 1/2$ is allowed. It is well-known that in the bounded-error model, a large difference between classical and quantum computation is possible. The complexity gap between the best known classical algorithm and quantum algorithm can be exponential, as, for instance, in the case of the Shor's algorithm [9]. Another famous example is the Grover's search algorithm that achieves a quadratic speed-up [10]. However, in certain types of computer software, we cannot allow even a small probability of error, for example, in spacecraft, aircraft, or medical software. For this reason, the development of exact algorithms is extremely important.

Regarding exact quantum algorithms, the maximum speed-up achieved as of now is half the number of queries compared with a classical deterministic case[2] [11]. The major open question is: is it possible to reduce the number of queries by more than 50%? In this paper, we present an algorithm that achieves the borderline gap of N/2 versus N.

---

[2] Exact quantum algorithm with complexity $Q_E(f) < D(f)/2$ is not yet discovered for a total Boolean function. For partial Boolean functions this limitation can be exceeded. An excellent example is the Deutsch-Jozsa algorithm [12, 13].

2 Preliminaries

This section contains definitions and provides theoretical background on the subject. First, we describe classical decision trees and show how to compute a simple Boolean function in this model. Next, we provide a brief overview of the basics of quantum computing. Finally, we describe the quantum query model that is the subject of this paper.

2.1 Classical Decision Trees

The classical version of the query model is known as *decision trees* [1]. A black box contains the input $X = (x_1, x_2, ..., x_N)$ and can be accessed by querying x_i values. The algorithm must allow to determine the value of a function correctly for arbitrary input. The complexity of the algorithm is measured by the number of queries on the worst-case input. For more details, see the survey by Buhrman and de Wolf [1].

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

It has been proven that $D(f) \geq s(f)$ [1].

Figure 1 demonstrates a classical deterministic decision tree, which computes $MAJORITY_3(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$. In this figure, circles represent queries, and rectangles represent output. It is easy to see that the third query is necessary if values of first two queried variables are different: $D(MAJORITY_3(x_1, x_2, x_3)) = 3$.

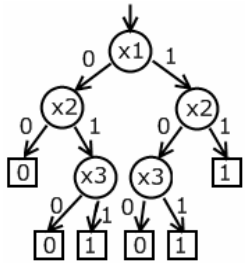

Fig. 1. Classical deterministic decision tree for computing $MAJORITY_3(x_1, x_2, x_3)$

As in many other models of computation, the power of randomization can be added to decision trees [1]. A *probabilistic decision tree* may contain internal nodes with a probabilistic branching, i.e., multiple arrows exiting from this node, each one labeled with a probability for algorithm to follow that way. The total sum of all probabilities assigned to arrows in a probabilistic branching is supposed not to exceed 1. The result

of a probabilistic decision tree is not determined by the input X with certainty anymore. Instead, there is a probability distribution over the set of leaves. The total probability to obtain a result $b \in \{0,1\}$ after the execution of an algorithm on certain input X equals the sum of probabilities for each leaf labeled with b to be reached. The total probability of an algorithm to produce the correct result is the probability on the worst-case input.

2.2 Quantum Computing

This section briefly outlines the basic notions of quantum computing that are necessary to define the computational model used in this paper. For more details, see the textbooks by Nielsen and Chuang [3] and Kaye et al [4].

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

The state of a system can be changed by applying *unitary transformation.* The unitary transformation U is a linear transformation on C^n that maps vectors of unit norm to vectors of unit norm. The *transpose* of a matrix A is denoted with $A_{ij}^T = A_{ji}$. We denote the *tensor product* of two matrices with $A \otimes B$.

The simplest case of quantum measurement is used in our model – the full measurement in the computation basis. Performing this measurement on a state $|\psi\rangle=a_0|0\rangle+\ldots a_{n\text{-}1}|n\text{-}1\rangle$ produces the outcome i with probability $|a_i|^2$. The measurement changes the state of the system to $|i\rangle$ and destroys the original state $|\psi\rangle$.

2.3 The Quantum Query Model

The quantum query model is also known as the quantum black box model. This model is the quantum counterpart of decision trees and is intended for computing Boolean functions. For a detailed description, see the survey by Ambainis [5] and textbooks by Kaye, Laflamme, Mosca [4], and de Wolf [2].

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

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

U_i's can be arbitrary unitary transformations that do not depend on input bits. Q_i's are query transformations. Computation starts in the initial state $\left|\vec{0}\right\rangle$. Then we apply U_0, $Q_0,\ldots, Q_{T\text{-}1}$, U_T and measure the final state.

We use the *ket* notation [3] to describe state vectors and algorithm structure:

$$\left| final \right\rangle = U_T \cdot Q_{T-1} \cdot \ldots \cdot Q_0 \cdot U_0 \cdot \left|\vec{0}\right\rangle .$$

We use the following definition of a query transformation: if input is a state $|\psi\rangle = \sum_i a_i |i\rangle$, then the output is $|\phi\rangle = \sum_i (-1)^{x_{k_i}} a_i |i\rangle$, where we can arbitrarily choose a variable assignment of x_{k_i} for each basis state $|i\rangle$. It is also allowed to skip variable assignment for any particular basis state, i.e. to set $x_{k_i} = 0$ for a particular $|i\rangle$.

Formally, any transformation must be defined as a unitary matrix. The following is a matrix representation of a quantum black box query.

$$Q = \begin{pmatrix} (-1)^{X_{k_1}} & 0 & \dots & 0 \\ 0 & (-1)^{X_{k_2}} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & (-1)^{X_{k_m}} \end{pmatrix}$$

Each quantum basis state corresponds to the algorithm's output. We assign a value of a function to each output. The probability of obtaining the result $j \in \{0,1\}$ after executing an algorithm on input *X* equals the sum of squared moduli of all amplitudes, which correspond to outputs with value *j*.

Definition 3 [1]. *A quantum query algorithm* ***computes f exactly*** *if the output equals f(x) with a probability* $p = 1$*, for all* $x \in \{0,1\}^N$*. Complexity is equal to the number of queries and is denoted with* $Q_E(f)$*.*

Definition 4 [1]. *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 with* $Q_p(f)$*.*

Quantum query algorithms can be conveniently represented in diagrams, and we will use this approach in this paper.

3 Quantum Query Algorithms for the Boolean Function *AND*

In this section, we present our results in constructing quantum query algorithms for a set of Boolean functions based on the *AND* Boolean operation. We consider bounded-error algorithms, which output a correct answer with some probability. Regarding computing a two-variable function $AND(x_1, x_2)$, the results were obtained as follows: using a method described in Section 2.2.1 of [14], it is possible to construct a bounded-error quantum algorithm for $AND(x_1, x_2)$ with one query and a probability $p = 2/3$. A better probability of correct answer for a one-query algorithm was obtained in [15] and it is $p = 3/4$. In [16] in a proof for Lemma 1, an algorithm for computing an arbitrary two-variable Boolean function is presented, whose probability

is p=11/14. The authors also claim to be able to prove that probability p=9/10 is optimal.

In this paper, we improve these results and show an algorithm which computes $AND(x_1, x_2)$ with one query and a probability $p = 4/5$. Moreover, we extend an algorithm to compute the *AND* of two functions.

This section is organized as follows: first, we discuss the classical complexity of the two-argument Boolean function $AND(x_1, x_2)$. Then we demonstrate a bounded-error quantum query algorithm that computes $AND(x_1, x_2)$ with a probability $p = 4/5$. Finally, we generalize our approach and present a method for constructing efficient quantum algorithms for computing a *composite function* $\overline{AND}_2[f_1, f_2]$, where f_1 and f_2 are Boolean functions.

Definition 5. *We define* $\overline{\boldsymbol{AND}}_{\boldsymbol{n}}$ ***construction*** *($n \in N$) as a composite Boolean function where arguments are arbitrary Boolean functions f_i and which is defined as*

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

$X = X_1X_2...X_n$; X_i is input for i^{th} function[3]; f_i's are called base functions.

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

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

Next we will show that the best probability for a classical randomized decision tree to compute this function with one query is p = 2/3. The general form of the optimal randomized decision tree is shown in Figure 2.

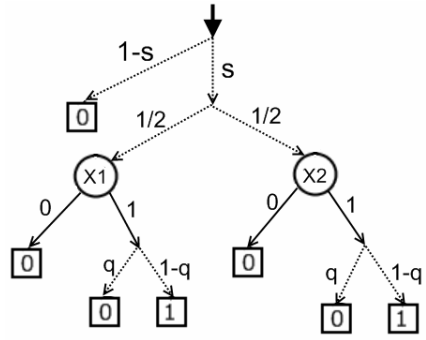

Fig. 2. The general form of the optimal randomized decision tree for computing $AND_2(x_1,x_2)$

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

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

1) $\Pr("0" \mid X = 00) = (1-s) + \frac{1}{2}s + \frac{1}{2}s = 1$,

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

3) $\Pr("1" \mid X = 11) = \frac{1}{2}s(1-q) + \frac{1}{2}s(1-q) = s - sq$.

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

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

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

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

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

Corollary 1. The *Boolean function* $AND_2(x_1, x_2)$ *can be computed by a randomized classical decision tree with one query with the maximum probability* p=2/3.

3.2 Quantum Query Algorithms for $AND(x_1, x_2)$

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

Theorem 1. *There exists a quantum query algorithm Q1 that computes the Boolean function* $AND(x_1, x_2)$ *with one quantum query and correct answer probability is* p=4/5: $Q_{4/5}(AND_2) = 1$.

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

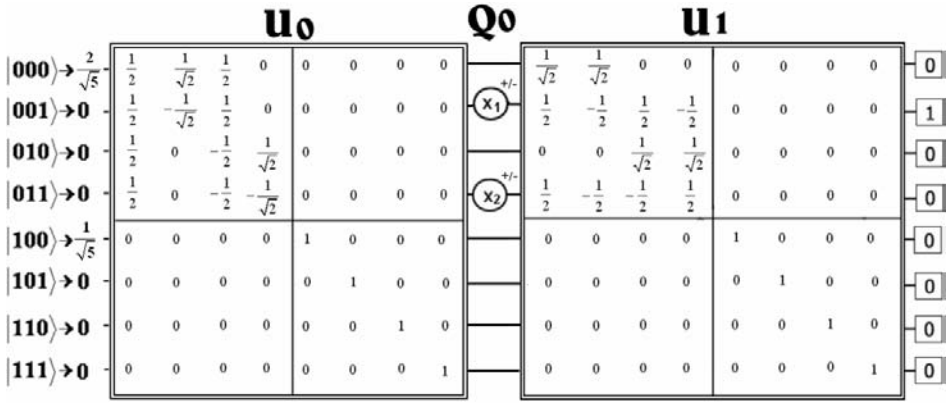

Fig. 3. Bounded-error quantum query algorithm Q1 for computing AND(x_1,x_2)

Quantum state after the first transformation U_0 becomes equal to

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

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

Further evolution of the quantum system for each input X is shown in Table 1.

Table 1.

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

X	$\lvert\varphi_3\rangle = Q_0U_0\lvert\varphi\rangle$	$\lvert\varphi_{FINAL}\rangle = U_1Q_0U_0\lvert\varphi\rangle$	p("1")
00	$\left(\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	$\left(\sqrt{\frac{2}{5}},\ 0,\ \sqrt{\frac{2}{5}},\ 0,\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	0
01	$\left(\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ -\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	$\left(\sqrt{\frac{2}{5}},\ \frac{1}{\sqrt{5}},\ 0,\ -\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	$\frac{1}{5}$
10	$\left(\frac{1}{\sqrt{5}},\ -\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	$\left(0,\ \frac{1}{\sqrt{5}},\ \sqrt{\frac{2}{5}},\ \frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	$\frac{1}{5}$
11	$\left(\frac{1}{\sqrt{5}},\ -\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ -\frac{1}{\sqrt{5}},\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	$\left(0,\ \frac{2}{\sqrt{5}},\ 0,\ 0,\ \frac{1}{\sqrt{5}},\ 0,\ 0,\ 0\right)^T$	$\frac{4}{5}$

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

This section is a transitional point to the generalized method for computing the construction $\overline{AND_2}$. Now we will reveal the internal details of the algorithm *Q1* that allow us to adapt its structure to compute a much wider set of Boolean functions.
The quite chaotic and asymmetric matrix U_0 actually is a product of two other matrices.

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

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

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

Detailed algorithm structure now looks as follows.

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

The final vector is calculated as $|\varphi_{FINAL}\rangle = U_1^B \cdot U_1^A \cdot Q_0 \cdot U_0^B \cdot U_0^A \cdot |\varphi\rangle$.

Now the most important point – the algorithm part represented by transformations U_0^B , Q_0 , U_1^A actually executes two instances of an exact quantum query algorithm for $f(x) = x$ in parallel. Fig. 4 and 5 graphically demonstrate this significant detail.

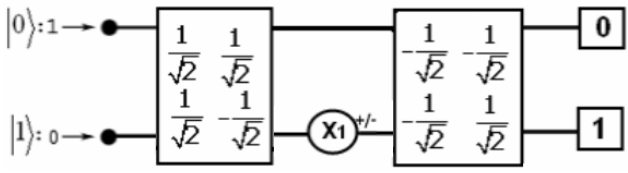

Fig. 4. An exact quantum query algorithm for computing f(x) = x

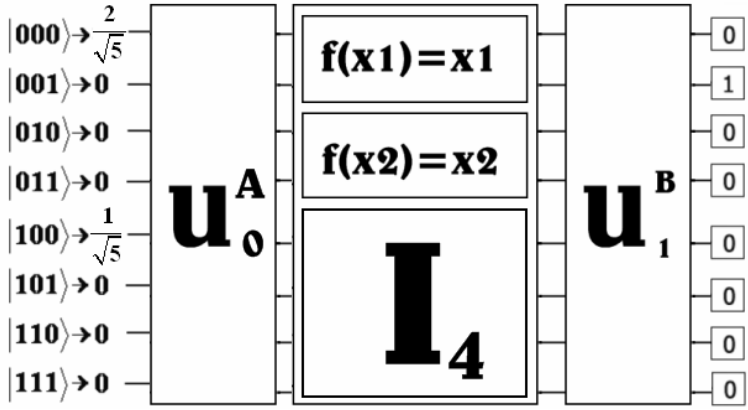

Fig. 5. A quantum algorithm for AND(x_1,x_2), revised

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

In the next section we will generalize this approach to allow to use other Boolean functions as sub-routines.

3.4 A Method for Computing $\overline{AND_2}[f_1, f_2]$

It is possible to replace a sub-algorithm for $f(x) = x$ (in an algorithm construction demonstrated in the previous section) with any other quantum algorithm which satisfies specific properties. We define a class *EQQA+*, and our method is applicable to base algorithms that belong to this class.

Definition 6. *An exact quantum query algorithm belongs to the* ***class EQQA+*** *(positive exact quantum query algorithms) iff there is exactly one accepting basis state, and on any input for its amplitude* $\alpha \in C$ *only two values are possible before the final measurement: either* $\alpha = 0$ *or* $\alpha = 1$.

Theorem 2. *If there exist exact quantum query algorithms A1 and A2 for computing Boolean functions* $f_1(X_1)$ *and* $f_2(X_2)$ *that belong to the class EQQA+, then a composite Boolean function* $\overline{AND_2}[f_1, f_2]$ *can be computed with a probability p = 4/5 using* $\max(Q_E(A1), Q_E(A2))$ *queries to the black box.*

Proof. A general algorithm construction method for computing the Boolean function $\overline{AND_2}[f_1, f_2]$ is presented below. The main idea is to assign the amplitude value $\alpha = 1/\sqrt{5}$ to some fixed basis state and leave it invariable until the end of the execution.

A method for computing $\overline{AND}_2[f_1, f_2]$

Input. Two exact quantum query algorithms A1, A2 $\in EQQA+$ compute Boolean functions $f_1(X_1), f_2(X_2)$. We denote the dimension of Hilbert space utilized by the first algorithm with m_1 (the number of amplitudes), and by the second algorithm with m_2. We denote the positions of accepting outputs of A1 and A2 with acc_1 and acc_2.

Constructing steps

1. If $m_1 = m_2$, then utilize a quantum system with $4m_1$ amplitudes for a new algorithm. First $2m_1$ amplitudes will be used for the parallel execution of A1 and A2. Additional qubit is required to provide separate amplitude for storing the value of $1/\sqrt{5}$.
2. If $m_1 \neq m_2$ (without loss of generality assume that $m_1 > m_2$), then utilize a quantum system with $2m_1$ amplitudes for a new algorithm. First $(m_1 + m_2)$ amplitudes will be used for the parallel execution of A1 and A2. Use the first remaining free amplitude for storing the value of $1/\sqrt{5}$.
3. Combine unitary transformations and queries of A1 and A2 in the following way: $U_i = \begin{pmatrix} U_1 & O_{m_1 \times m_2} & O_{m_1 \times m_1} \\ O_{m_2 \times m_1} & U_2 & O_{m_2 \times m_1} \\ O_{m_1 \times m_1} & O_{m_1 \times m_2} & I_{m_1 - m_2} \end{pmatrix}$, here $O_{m_i \times m_j}$ are $m_i \times m_j$ zero-matrices, $I_{m_1 - m_2}$ is $(m_1 - m_2) \times (m_1 - m_2)$ identity matrix, U_i^1 and U_i^2 are either unitary transformations or query transformations of A1 and A2.
4. Start computation from the state
$$|\varphi\rangle = \left(\underbrace{\sqrt{2/5},\ 0,...,0}_{m_1}, \underbrace{\sqrt{2/5},\ 0,...,0,}_{m_2} \quad \underbrace{1/\sqrt{5},\ 0,..,0}_{remaining\ amplitudes} \right)^T .$$
5. Apply gates U_i. Before the final measurement apply an additional unitary gate.
$$U = (u_{ij}) = \begin{cases} 1, & \text{if } (i = j) \,\&\, (i \neq acc_1) \,\&\, (i \neq (m_1 + acc_2)) \\ 1/\sqrt{2}, & \text{if } (i = j = acc_1) \\ 1/\sqrt{2}, & \text{if } (i = acc_1) \,\&\, (j = (m_1 + acc_2)) \text{ OR } (i = (m_1 + acc_2)) \,\&\, (j = acc_1) \\ -1/\sqrt{2}, & \text{if } (i = j = (m_1 + acc_2)) \\ 0, & \text{otherwise} \end{cases}$$
6. Define as accepting output exactly one basis state $|acc_1\rangle$.

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

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

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

3.5 Class *EQQA*+

In this section, we show that *EQQA*+ class (see Definition 6) is wide enough to be taken into consideration. At the same time, approaches for constructing efficient instances of *EQQA*+ are worth to be examined in a separate paper.

3.5.1 Conversion of Classical Decision Trees into Quantum Query Algorithms

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

A classical query to the black box can be simulated with a quantum query algorithm construction presented in Fig. 6.

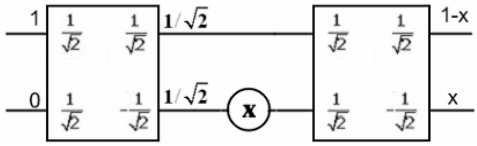

Fig. 6. Quantum query algorithm construction for simulating a classical query

After the second Hadamard gate we obtain $(1,0)^T$ if $x = 0$, or $(0,1)^T$ if $x = 1$.

Then we can continue to query other variables by logically splitting the algorithm flow into two separate threads and so on.

We demonstrate a complete example of converting a classical decision tree for computing $AND(x_1, x_2)$ into an exact quantum query algorithm. Fig. 7 shows a classical decision tree. Figure 8 shows the corresponding exact quantum query algorithm.

We would like to note that, although such conversion is possible, it is not optimal. For instance, it is well known that *XOR* can be computed in a quantum model using two times less queries than required in a classical model.

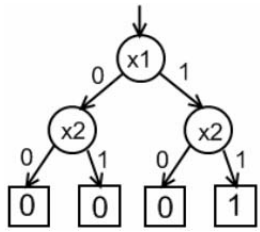

Fig. 7. A classical deterministic decision tree for $AND(x_1, x_2)$

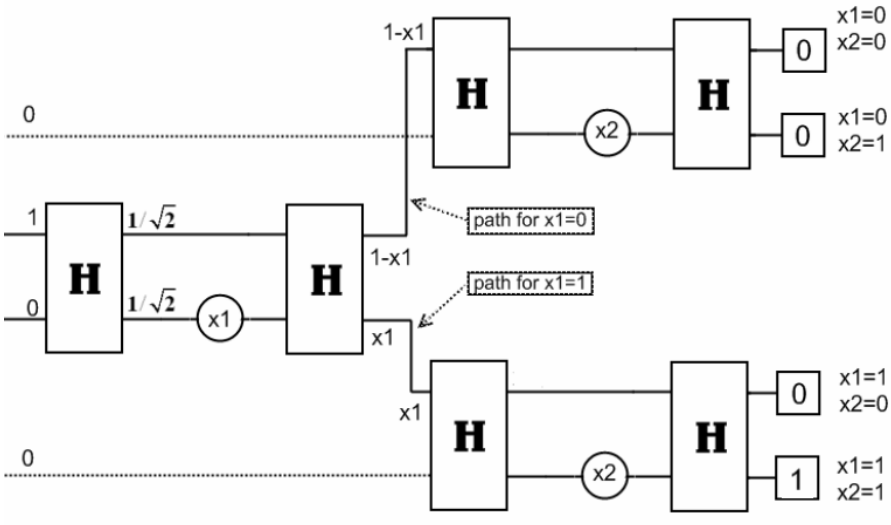

Fig. 8. An exact quantum query algorithm for $AND(x_1, x_2)$

If a deterministic decision tree has exactly one leaf with output value "1", then it obviously will be converted into an algorithm of class *EQQA+*.

It means that we can place into $\overline{AND_2}$ construction any Boolean function that has exactly one accepting input vector. Thus, our method is applicable to an infinite set of base functions.

Theorem 3. *For an infinite set of Boolean functions, quantum query algorithms can be constructed using a method described in Section 3.4. As a result, the following complexity gap can be achieved when computing the same function in quantum and classical deterministic models:* $Q_{4/5}(\overline{AND}_2(f_1, f_2)) = \frac{1}{2} \cdot D(\overline{AND}_2(f_1, f_2))$.

Proof. For any Boolean function f that has exactly one accepting vector, the sensitivity $s(f)$ and, consequently, the deterministic complexity $D(f)$ are equal to the number of variables. Suppose we have two such Boolean functions f_1 and f_2, with the same number of variables N, and wish to compute $\overline{AND_2}(f_1, f_2)$ [4]. Obviously, the

[4] We assume that variables do not overlap this time.

classical deterministic complexity of this function is $D(\overline{AND}_2(f_1, f_2)) = 2N$. For each function we can convert a deterministic algorithm into an exact quantum query algorithm of the class *EQQA+*, which will use the same *N* queries. Finally, we apply the method for constructing an algorithm for $\overline{AND}_2(f_1, f_2)$ which does not require additional queries: $Q_{4/5}(\overline{AND}_2(f_1, f_2)) = N = \frac{1}{2} \cdot D(\overline{AND}_2(f_1, f_2))$.

In the theorem above, classical deterministic and quantum bounded-error query complexity is compared. It would be interesting to compare classical probabilistic and quantum bounded-error complexity correlation for $\overline{AND}_2(f_1, f_2)$. As of today, we do not have such estimation yet.

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

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

$$AND_N = \overline{AND}_2(AND_{N/2}, AND_{N/2}).$$

It means that by applying our construction method it is possible to obtain an algorithm with complexity

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

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

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

3.6 An Example of a Larger Separation: $D(f) = 6$ vs. $Q_{4/5}(f)=2$

We would like to demonstrate an example when quantum algorithm complexity can be over two times less than classical deterministic algorithm complexity. It is possible in cases when an exact quantum algorithm for a sub-function is better than the best possible deterministic algorithm for the same function.

An exact quantum query algorithm for $EQUALITY_3(X) = \neg(x_1 \oplus x_2) \wedge \neg(x_2 \oplus x_3)$ has been first presented in [17]. The algorithm is depicted in Fig. 12 and it uses only two quantum queries while classically all three queries are required. The algorithm belongs to the class *EQQA+* and can be used as a sub-algorithm for $\overline{AND}_2$ construction.

To evaluate deterministic complexity of $f = \overline{AND}_2[EQUALITY_3, EQUALITY_3]$, we use function sensitivity on any accepting input: $s(f) = 6 \Rightarrow D(f) = 6$.

A quantum bounded-error algorithm for $f = \overline{AND}_2[EQUALITY_3]$ constructed using our method will require only two queries: $Q_{4/5}(f) = 2$.

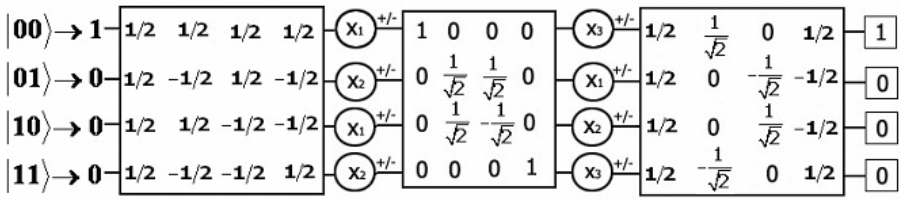

Fig. 9. An exact quantum query algorithm for EQUALITY$_3$

The same approach can be applied to any algorithm of class *EQQA+* that computes an *N*-variable Boolean function.

3.7 Repeated Application of a Method for Computing $\overline{AND}_2[f_1, f_2]$

The useful properties of the algorithm construction method described in Section 3.4 allow to apply this method repeatedly.

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

Proof. We straightforwardly apply the method for computing $AND_2[f_1,f_2]$ to algorithms $Q1$ and $Q2$ instead of instances of QQA^{+1} class. As a result, the obtained complex algorithm computes $F = AND_2[F_1, F_2]$ with a probability $p = \frac{4}{5} \cdot \frac{4}{5} = \frac{16}{25}$.

As a consequence, we are able to compute a four-variable function $AND(x_1,\ldots,x_4)$ with a single quantum query with a probability p=16/25.

Next iteration produces quantum algorithms that compute functions like

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

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

4 An Exact Quantum Query Algorithm for Verifying Repetition Code

In this section, we consider the second problem: verification of the codeword encoded by the repetition code for error detection. In the first sub-section, we introduce repetition codes and define a Boolean function for their verification. Secondly, we

show that classically, for an N-bit message, values of all N variables must be queried in order to detect an error. Finally, we present an exact quantum algorithm for N-bit codeword verification that uses only $N/2$ queries to the black box.

4.1 Error Detection and Repetition Codes

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

We consider a repetition error detection scheme known as repetition codes. A repetition code is a (r, N) coding scheme that repeats each N-bit block r times [8].

An example

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

Verification procedure for the repetition code is the following – we need to check if in each group of r consecutive blocks of size N all blocks are equal.

We start with verification of the (2,1) repetition code. The verification process can be expressed naturally as computing a Boolean function in a query model. We assume that the codeword to be checked is located in a black box. We define the Boolean function to be computed with the query algorithm as follows.

Definition 7. *The Boolean function* $VERIFY_N(X)$, *where* $N = 2k$, $X = (x_1, x_2, ..., x_{2k})$ *is defined to have a value of "1" iff variables are equal by pairs.*

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

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

$$\{0000, 0011, 1100, 1111\}.$$

4.2 Deterministic Complexity of $VERIFY_N$

Fig. 10 demonstrates a classical deterministic decision tree which computes $VERIFY_4(x_1, x_2, x_3, x_4)$. In this figure, circles represent queries, and rectangles represent output.

Theorem 6. $D(VERIFY_N) = N$ **.**

Proof. Check function sensitivity on any accepting input, for instance, on X = 1111..11. Inversion of any bit will invert the function value, because a pair of bits with different values will appear. $s(VERIFY_N) = N \quad \Rightarrow \quad D(VERIFY_N) = N$.

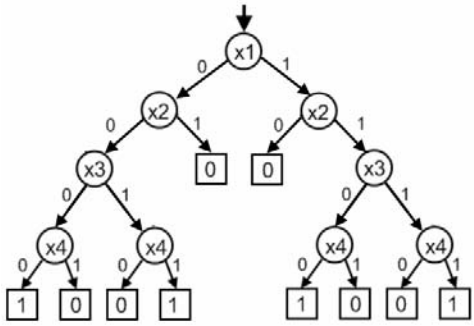

Fig. 10. A classical deterministic decision tree for computing $VERIFY_4(x_1, x_2, x_3, x_4)$

4.3 Computing the Function $VERIFY_N$ in a Quantum Query Model

Our approach to computing the Boolean function $VERIFY_N$ in a quantum query model is based on an exact quantum query algorithm for the *XOR* function.

Theorem 7. *There exists an exact quantum query algorithm that computes the Boolean function* $VERIFY_N(X)$ *using N/2 queries:* $Q_E(VERIFY_N) = N/2$.

Proof. Definition of the $VERIFY_N$ function can be re-formulated as follows.

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

An exact quantum algorithm for computing the Boolean function $f(x_1, x_2) = \neg(x_1 \oplus x_2)$ with one query is presented in Fig. 11. We compose an algorithm for $VERIFY_N$ using an algorithm for $f(x_1, x_2) = \neg(x_1 \oplus x_2)$ as building blocks.

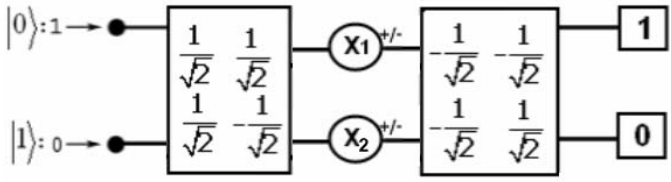

Fig. 11. An exact quantum query algorithm for computing $f(X) = \neg(x_1 \oplus x_2)$

First, we execute an algorithm for $f(x_1, x_2) = \neg(x_1 \oplus x_2)$ for variables x_1 and x_2. To the first output (which has "1" assigned, see Fig. 11), we concatenate the second instance of an algorithm for computing $f(x_1, x_2) = \neg(x_1 \oplus x_2)$. This time we execute it for variables x_3 and x_4. We continue this way until all variables of $VERIFY_N$ are queried. The algorithm has only one accepting output, which is the first output of the last sub-algorithm.

A schematic view of the described approach is depicted in Fig. 12. It is easy to see that the total number of queries is $N/2$. □

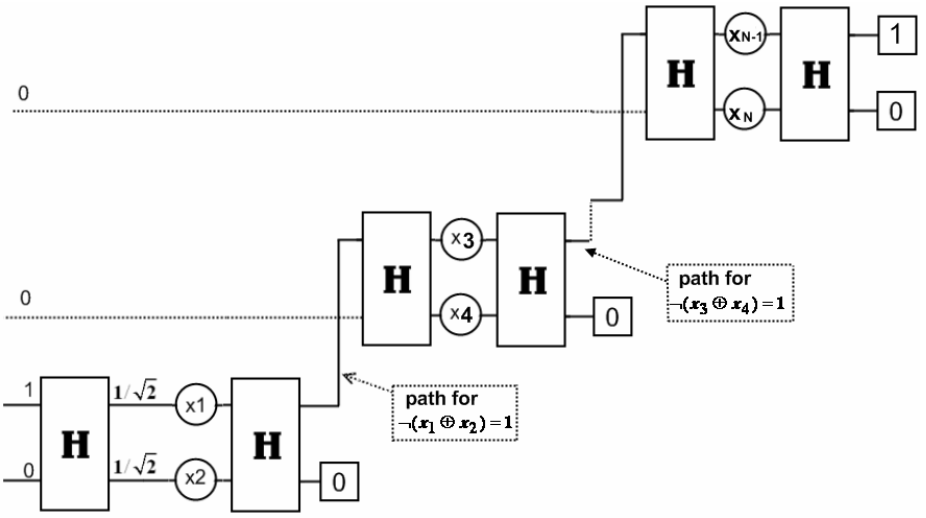

Fig. 12. An algorithm for computing the Boolean function $VERIFY_N$

4.4 Application to a String Equality Problem

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

A quantum algorithm for the Boolean function $VERIFY_N$ checks whether variables are equal by pairs, i.e. $(x_1 = x_2) \wedge (x_3 = x_4) \wedge \ldots \wedge (x_{N-1} = x_N)$. On the other hand, we can consider that this algorithm checks whether two binary strings,

$Y = x_1x_3x_5...x_{N-1}$ and $Z = x_2x_4x_6...x_N$, are equal. Therefore, the algorithm can be easily used not only to verify repetition codes, but also for checking equality of binary strings.

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

Now, let us consider the (*r*,1) repetition code, where each bit is repeated *r* times during encoding. Verification procedure for a codeword encoded using such code consists of checking whether in each sequence of *r* bits all bits are equal.

The Boolean function $EQUALITY_r$ is defined as

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

We define the Boolean function that corresponds to verification procedure as

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

Theorem 8. *Deterministic complexity of the Boolean function* $VERIFY^r_{r\cdot N}(X)$ *is equal to the number of variables:* $D(VERIFY^r_{r\cdot N}) = Nr$.

Proof. Again, we use function sensitivity on any accepting input. Inversion of any bit will invert the function value because a pair of bits with different values will appear. $s(VERIFY^r_{r\cdot N}) = Nr \quad \Rightarrow \quad D(VERIFY^r_{r\cdot N}) = Nr$.

Theorem 9. *There exists an exact quantum query algorithm that computes the Boolean function* $VERIFY^r_{r\cdot N}(X)$ *using* $N \cdot r - N$ *queries:*

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

Proof. Again, to speed up the verification procedure, we take into account the fact that *XOR* of two bits can be computed with one quantum query. The Boolean function $EQUALITY_r$ can be expressed using operations $\oplus$, $\wedge$ and $\neg$:

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

This logical formula contains (*r*-1) clauses each consisting of *XOR* of two bits. Using the approach described in the proof of Theorem 7, we can compose an exact quantum query algorithm which computes $EQUALITY_r$ using (*r*-1) quantum queries. This algorithm has one accepting output. Next we use an algorithm for $EQUALITY_r$ as a building block for composing an algorithm to compute $VERIFY^r_{r\cdot N}$. The resulting algorithm uses $N(r-1)$ queries to determine the value of the Boolean function $VERIFY^r_{r\cdot N}$ exactly.

5 Conclusion

In this paper, we presented quantum query algorithms for resolving two specific computational problems.

First, we considered computing the Boolean function *AND* in a bounded-error setting. We presented a quantum query algorithm that computes $AND(x_1, x_2)$ with one query and probability p = 4/5 while the optimal classical randomized algorithm can compute this function with a probability p=2/3 only. Then we extended our approach and formulated a general method for computing $\overline{AND}_2[f_1, f_2]$ composite construction with the same probability p=4/5 and number of queries equal to $\max(Q_E(f_1), Q_E(f_2))$. The suggested approach allows to build quantum algorithms for complex functions based on already known algorithms. A significant behavior is that the overall algorithm complexity does not increase; additional queries are not required to compute a composite function. However, error probability is the cost for efficient computing. We demonstrated that our method is applicable to a large set of Boolean functions. As a result, a complexity gap of $Q_{4/5}(f) = 1/2 \cdot D(f)$ can be achieved for an infinite set of Boolean functions. We also showed that this is not the lower bound for quantum algorithm complexity and examples where $Q_{4/5}(f) < 1/2 \cdot D(f)$ can be constructed as well.

In the second part of this paper, we considered verification of error detection codes. We have represented the verification procedure as an application of a query algorithm to an input codeword contained in a black box. We have represented an exact quantum query algorithm which allows to verify a codeword of length *N* using only *N*/2 queries to the black box. Our algorithm saves exactly half the number of queries comparing to the classical case. This result repeats the largest difference between classical and quantum algorithm complexity for a total Boolean function known today in this model.

We see many possibilities for future research in the area of quantum query algorithm design. The most significant open question still remains: is it possible to increase exact algorithm performance more than two times using quantum tools? Furthermore, there are many computational tasks waiting for an efficient solution in a quantum setting. Regarding the *AND* Boolean function, we would like to improve correct answer probability when computing a two-variable *AND* with one query. Regarding verification of repetition codes, we would like to be able to verify efficiently not only the (2,1) code, but also an arbitrary (*r*,*N*) code. Another fundamental goal is to develop a framework for building efficient *ad-hoc* quantum query algorithms for arbitrary Boolean functions.

Acknowledgments. I would like to thank my supervisor Rusins Freivalds for introducing me to quantum computation and for his constant support and advice.

This research is supported by the European Social Fund project No. 2009/0138/1DP/1.1.2.1.2/09/IPIA/VIAA/004, Nr. ESS2009/77.

References

1. H. Buhrman and R. de Wolf. Complexity Measures and Decision Tree Complexity: A Survey. *Theoretical Computer Science*, v. 288(1), 2002, pp. 21–43.
2. R. de Wolf. *Quantum Computing and Communication Complexity*. University of Amsterdam, 2001.
3. M. Nielsen, I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
4. P. Kaye, R. Laflamme, M. Mosca. *An Introduction to Quantum Computing*. Oxford, 2007.
5. A. Ambainis. Quantum query algorithms and lower bounds. (Survey article.) In: *Proceedings of FOTFS III, Trends on Logic*, vol. 23, 2004, pp. 15–32.
6. A. Ambainis and R. de Wolf. Average-case quantum query complexity. *Journal of Physics* A 34, 2001, pp. 6741–6754.
7. A. Ambainis. Polynomial degree vs. quantum query complexity. *Journal of Computer and System Sciences*, 72, 2006, pp. 220–238.
8. T. M. Cover, J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, 1991, pp. 209–212.
9. P. W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5), 1997, pp. 1484–1509.
10. L. Grover. A fast quantum mechanical algorithm for database search. In: *Proceedings of 28th STOC '96*, 1996, pp. 212–219.
11. A. Ambainis, personal communication, April 2009.
12. D. Deutsch, R. Jozsa. Rapid solutions of problems by quantum computation. *Proceedings of the Royal Society of London*, Vol. A 439, 1992, pp. 553–558.
13. R. Cleve, A. Ekert, C. Macchiavello, M. Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London*, Vol. A 454, 1998, pp. 339–354.
14. L. Lāce. Quantum Query Algorithms. Doctoral Thesis. University of Latvia, 2008, pp. 42–43.
15. A. Vasilieva. Quantum Query Algorithms for AND and OR Boolean Functions, Logic and Theory of Algorithms. *Proceedings of the 4th Conference on Computability in Europe*, 2008, pp. 453–462.
16. I. Kerenidis, R. de Wolf. Exponential Lower Bound for 2-Query Locally Decodable Codes via a Quantum Argument. *Journal of Computer and System Sciences*, 2004, pp. 395–420.
17. A. Dubrovska. Quantum Query Algorithms for Certain Functions and General Algorithm Construction Techniques. *Quantum Information and Computation V, Proc. of SPIE*, vol. 6573. SPIE, Bellingham, WA, article 65730F, 2007.