SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 68–89 P.

Computing Relations in the Quantum Query Model [1]

Alina Vasilieva, Taisia Mischenko-Slatenkova
Faculty of Computing, University of Latvia
Raina blvd. 29, LV-1459, Riga, Latvia
Alina.Vasiljeva@gmail.com, Taisia.Miscenko@gmail.com

Query algorithms are used to compute mathematical functions. Classical version of this model is also known as decision trees. Quantum counterpart of decision trees – quantum query model – has been actively studied in recent years. Typically, query model is used to compute Boolean functions. In this paper, we consider computing mathematical relations instead of functions. A relation is a set of ordered pairs and the difference from a function is that each element from a domain set may be mapped to multiple elements from a range set. We demonstrate that quantum query model is well suited for computing relations. We present examples of quantum query algorithms that are more efficient than the best possible classical algorithms for computing specific relations.

Keywords: quantum computing, quantum query algorithm, algorithm complexity, mathematical relation.

1 Introduction

Query model is a popular, elegant and rather simple model of computation. The goal is to compute the value of a well-known function for an arbitrary hidden input. The complexity of a query algorithm is measured by the number of questions it asks about the input variables on the worst-case input. The classical version of this model is known as *decision trees* [1].

Quantum computing is an alternative way of computation based on the laws of quantum mechanics. Quantum algorithms can solve certain problems faster than classical algorithms. The most exciting examples are Shor's [2] and Grover's algorithms [3]. This branch of computer science is developing rapidly; various computational models exist and we consider one of them. Many impressive quantum query algorithms have been developed in a query model in recent years [4-8]. An important task in complexity theory is to find examples with a large gap between classical and quantum algorithm complexity of the same computational problem.

[1] This work has been supported by the European Social Fund within the project „Support for Doctoral Studies at University of Latvia".

Most often query model is used to compute Boolean functions. However, it is possible to apply query model to functions with larger domain and range as well. In this paper, we consider even more uncommon case – computing *mathematical relations* instead of functions. A binary relation is 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. Alternative way is to consider relations as multi-valued functions.

The study of query complexity of relations has been inspired by the book on communication complexity by Kushilevitz and Nisan [9]. 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 traditional query model to compute relations. In classical deterministic settings, however, it does not seem to be possible to employ the difference between a relation and a function to obtain new interesting results. A deterministic decision tree always follows one and the same fixed path for each certain input and outputs one and the same value each time. The situation is different in the quantum case. Quantum state before the measurement is in a superposition of the basis states, so it is not determined to which exactly basis state quantum system collapses after the measurement.

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

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

In this paper, we adapt the query model for computing relations. First, we give the definitions related to mathematical relations. We define several types of query algorithms that may compute relations in different manners. Then we demonstrate examples of computing relations in classical and quantum query models, where quantum algorithm achieves a speed-up comparing to classical algorithm. Finally, we discuss the prospects of achieving good results in enlarging the complexity gap between classical and quantum query complexity for relations.

2 Preliminaries

This section contains definitions and provides theoretical background on the subject. First, we define classical decision tree. Next, we provide a brief overview of the basics of quantum computing. Finally, we describe the quantum query model.

2.1 Classical Decision Trees

The classical version of the query model is known as *decision trees* [1]. Definition of Boolean function is known to everybody, but the input $X = (x_a, x_2, ..., x_n)$ is hidden in a black box, and can be accessed by querying x_i values. Algorithm must be able to determine value of the function correctly for arbitrary input. Complexity of the algorithm is measured by number of queries on the worst-case input. For more details, see the survey by Buhrman and de Wolf [1].

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. 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 $D(f)$.

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 Quantum Computing

We briefly outline basic notions of quantum computing here that are necessary to define the quantum query model. For more details, see [5, 6, 10].

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 $\mathbb{C}^n$. Then, any state can be expressed as $|\psi\rangle=\sum_{i=0}^{n-1} a_i |i\rangle$ for some $a_i \in \mathbb{C}$. The norm of $|\psi\rangle$ is 1, so 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 $\sum_{i=0}^{n-1} |a_i|^2 = 1$ is called a *superposition* of $|0\rangle,\ldots,|n\text{-}1\rangle$. The coefficient a_i is called the *amplitude* of $|i\rangle$.

The state of a system can be changed by applying *unitary transformation.* Unitary transformation U is a linear transformation on $\mathbb{C}^n$ that maps each vector of unit norm to a vector of unit norm. The *transpose* of an $m \times n$ matrix A is the $n \times m$ matrix $A_{ij}^T = A_{ji}$ for $1 \le i \le n$, $1 \le j \le m$. H denotes the Hadamard gate.

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

2.3 Quantum Query Model

The quantum query model is the quantum counterpart of the decision tree model and is intended for computing Boolean functions. For a detailed description, see [4-6].

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

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

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

We use the following definition of a query transformation: if the input is a state $|\psi\rangle = \sum_i a_i |i\rangle$, then the output is:

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

For each query, we may arbitrarily choose a variable assignment φ_i for each basis state. If the value of the assigned variable $\varphi_i \in \{x_1, ..., x_N\}$ is "1", then the sign of the i-th amplitude a_i changes to the opposite.

Each quantum basis state corresponds to an algorithm's output. We assign a value of the function to each output. The probability of obtaining the result j after executing the algorithm on input X equals to the sum of squared modulus of all amplitudes that correspond to outputs with value j.

Definition 1 [1]. *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* $Q_E(f)$.

3 Mathematical Relations

The main object which is studied in this paper is mathematical relations.

Definition 2 [11]. *A* relation R *from a set A to a set B is a subset of Cartesian product A × B – a collection of ordered pairs (a, b) with first components from the set A (domain) and second components from the set B (range).*

In other words, relation associates each value from the domain set with a subset of values from the range set. We call each value from the domain set – an input $X = (x_1, x_2, \ldots, x_N)$. We call each x_i – a variable. We call a set of associated values from the range set – a result set for input X and denote it by $R(X)$. We consider *left-total* relations only, when the result set is not empty for each domain set element. Relation can actually be considered a multi-valued function.

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. 1 graphically demonstrates this difference.

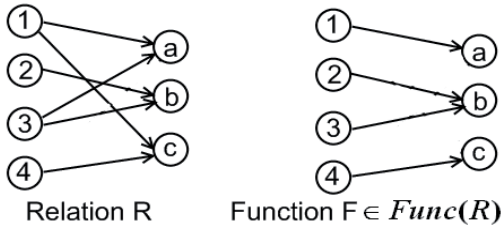

Fig. 1. Example of a relation and a function

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

Example. The graph on the left side of Fig. 1 defines the relation:

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

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

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

4 Computing Relations in a Query Model

It is well known how to compute functions in a query model. Algorithm simply has to output the function value with certain probability. But what does it mean to compute a relation in a query model? We propose three different options to describe that a query algorithm computes a relation and define three types of query algorithms based on these options.

Definition 3. *Query algorithm computes relation R 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* $C_D(R)$*. Quantum query complexity is denoted by* $Q_D(R)$*.*

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

Definition 4. *Query algorithm computes relation R in a* randomly distributed manner*, if for each X it outputs arbitrary values from a result set with arbitrary probabilities (for each value, such probability has to be positive) and never outputs an incorrect value. Classical query complexity is denoted by* $C_{RD}(R)$*. Quantum query complexity is denoted by* $Q_{RD}(R)$*.*

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

Definition 5. *Query algorithm computes relation R in uniformly* distributed manner*, if for each X it outputs each value from a result set with equal probability and never outputs an incorrect value. Classical query complexity is denoted by* $C_{UD}(R)$*. Quantum query complexity is denoted by* $Q_{UD}(R)$*.*

This definition adds a serious constraint to design of a query algorithm. However, in our opinion, this definition is the most reasonable in a sense of computing a relation.

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

5 Examples of Computing Relations

In this section, we present examples of computing relations in both classical and quantum query models. In all our examples, we achieve a speed-up in quantum algorithm complexity comparing to the best possible classical analogue.

5.1 First Example of Computing a Relation

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

In our example, we assume four agents: Alice (id = 1), Bob (id = 2), Carol (id = 3) and Daren (id = 4). There are three factors that determine a set of appropriate agents for each client – location, client status and loan history.

Table 1 describes these parameters. Second column contains a reference to the system function that has to be invoked to calculate parameter value. Invocation of each function can be interpreted as querying a black box and internal calculations may involve various database requests and other costly operations. Third column contains possible parameter values; fourth column contains corresponding numeric value returned by each function.

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

Table 1

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

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

Rows of Table 2 have to be interpreted as the following statements:

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

Table 2

Definition of the relation $\Re 1$

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

Now, let us discuss the computational complexity of relation $\Re 1$.

5.2 Definite Query Complexity of Relation $\Re 1$

When computing a relation in definite manner, algorithm has to output one certain correct value from a result set with probability $p = 1$. It means that we are just aware of that client's request is not forwarded to incompetent agent, but we do not care about the work distribution among the competent agents.

Theorem 1. $C_D(\Re 1) = 2$.

Proof. It is easy to see that one query is not enough to compute this relation classically in a definite manner. However, two queries are sufficient to reach the goal – we only need to know the values of the first two variables. Deterministic decision tree is shown in Fig. 2.

Actually, what we need is to compute *XOR* of the first two bits. If $XOR(x_1, x_2) = 0$, algorithm outputs "1". Otherwise, algorithm outputs "2". □

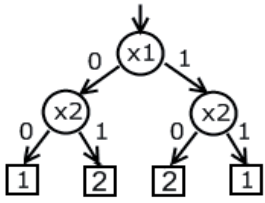

Fig. 2. Deterministic decision tree that computes $\Re 1$ in a definite manner

□

Theorem 2. $Q_D(\Re 1) = 1$.

Proof. It is well known that *XOR* of two bits can be computed exactly in the quantum query model by asking one query. It immediately implies that relation $\Re 1$ can

be computed with one query in a quantum query model in a definite manner. Quantum algorithm is shown in Fig. 3 and described below.

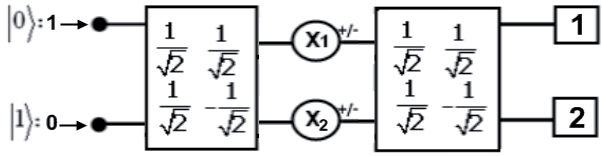

Fig. 3. Quantum query algorithm that computes $\Re 1$ in a definite manner

Algorithm uses one-qubit quantum system. Each horizontal line corresponds to the amplitudes of the basis states $|0\rangle$ and $|1\rangle$. Large rectangles correspond to the 2×2 Hadamard matrices. Single query Q_0 is defined by the unitary matrix:

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

Query matrix specifies how the signs of amplitudes of basis states change depending on variable values. Measurement is performed after the last unitary transformation. Finally, two small squares at the end of each horizontal line define the output value for each basis state. □

The problem with such implementation of work distribution algorithm is that all requests will be forwarded to Alice and Bob only, but Carol and Daren will be bored without work.

5.3 Uniformly Distributed Query Complexity of Relation $\Re 1$

Now, let us consider computing $\Re 1$ in uniformly distributed manner, which seems to be much more practical. This time algorithm has to output each value from the result set with equal probability and should never output incorrect value.

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

Theorem 3. $C_{UD}(\Re 1) = 2$.

Proof. Classical probabilistic decision tree that computes $\Re 1$ in uniformly distributed manner is shown in Fig. 4. □

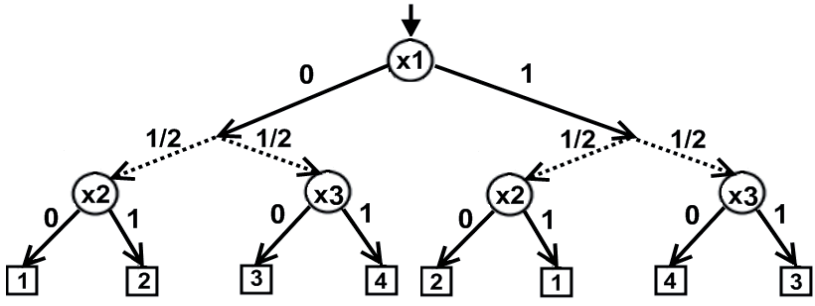

Fig. 4. Probabilistic decision tree that computes $\Re 1$ in uniformly distributed manner

Theorem 4. $Q_{UD}(\Re 1) = 1$.

Proof. Quantum query algorithm $Q1$, which computes $\Re 1$ in the same uniformly distributed manner with one query, is presented in Fig. 5 and described below.

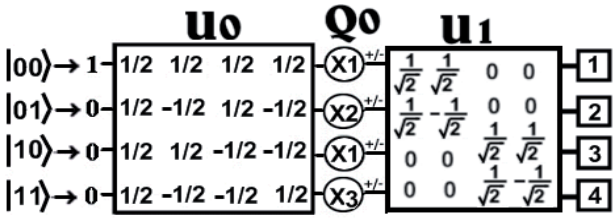

Fig. 5. Quantum query algorithm $Q1$ for computing $\Re 1$ in uniformly distributed manner

Algorithm $Q1$ uses two-qubit quantum system. Each horizontal line corresponds to the amplitude of the basis state. Large rectangles correspond to the 4 × 4 unitary matrices. Four small squares at the end of each horizontal line define the output value for each basis state.

Single query Q_0 is defined by the unitary matrix:

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

Computational process for each input X is shown in Table 3. □

Table 3

Computation process of the quantum query algorithm $Q1$

X	State after the query	State before the measurement	Output
000	$\left(\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("1")=1/2 Pr("3")=1/2
001	$\left(\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("1")=1/2 Pr("4")=1/2
010	$\left(\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("2")=1/2 Pr("3")=1/2
011	$\left(\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("2")=1/2 Pr("4")=1/2
100	$\left(-\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("2")=1/2 Pr("4")=1/2
101	$\left(-\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(0\ \frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("2")=1/2 Pr("3")=1/2
110	$\left(-\frac{1}{2}\ -\frac{1}{2}\ \frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ 0\ \frac{1}{\sqrt{2}}\right)^T$	Pr("1")=1/2 Pr("4")=1/2
111	$\left(-\frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\ -\frac{1}{2}\right)^T$	$\left(\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\ 0\right)^T$	Pr("1")=1/2 Pr("3")=1/2

This time all work items are equally distributed among agents.

With this basic example we have demonstrated query algorithms for computing relations in action. We have shown that even in such a simple case of relation with three Boolean variables it is possible to obtain a gap between classical and quantum query complexity. In the next subsection, we demonstrate how to enlarge the complexity gap in uniformly distributed case.

5.4 Generalizations of the Relation $\Re 1$

In this subsection, we demonstrate two extensions of the relation $\Re 1$ with a bigger number of variables and more impressive complexity separation between classical and quantum algorithms.

Definition 6. *Relation* $\Re 2 : \{0,1\}^N \rightarrow \{1, 2, ..., 2(N–1)\}$ *associates each input element from the domain set with (N-1) output elements from the range set according to the following rule:*

$$\forall\ 1 < i \leq N : \text{if } (x_1 \oplus x_i = 0), \quad \text{then } (X, 2(i–1)–1) \in \Re 2$$
$$\text{otherwise } (X, 2(i–1)) \in \Re 2$$

It turns out that it is possible to compute relation $\Re 2$ classically in uniformly distributed manner using two queries.

Theorem 5. $C_{UD}(\Re 2) = 2$.

Proof. Classical probabilistic decision tree is demonstrated in Fig. 6. □

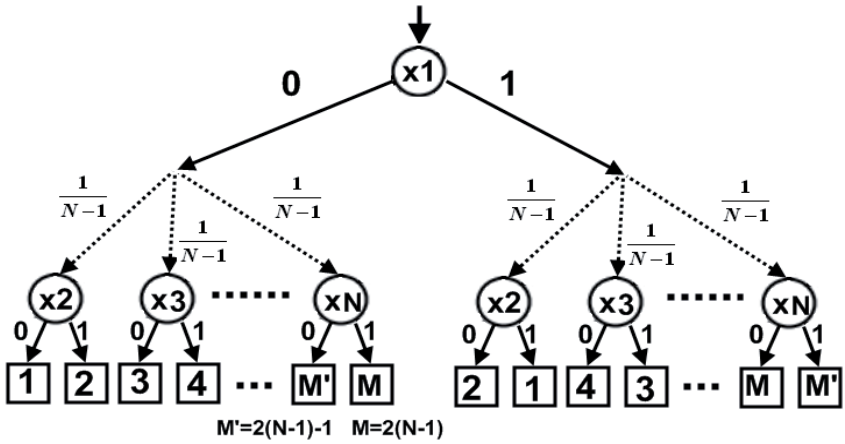

Fig. 6. Classical query algorithm for computing $\Re 2$ in uniformly distributed manner

Theorem 6. $Q_{UD}(\Re 2) = 1$.

Proof. To compute relation $\Re 2$ in a quantum query settings, we extend algorithm $Q1$ to query all N relation variables in a single query. To be able to handle all variables,

we extend quantum system to have $2(N–1)$ basis states. Fig. 7 shows quantum algorithm $Q2$, which is an extended version of the algorithm $Q1$. H is the 2×2 Hadamard transformation, $\oplus$ denotes matrix tensor product operation. Quantum system consists of A qubits, where $A = \lceil \log_2(2(N-1)) \rceil$.

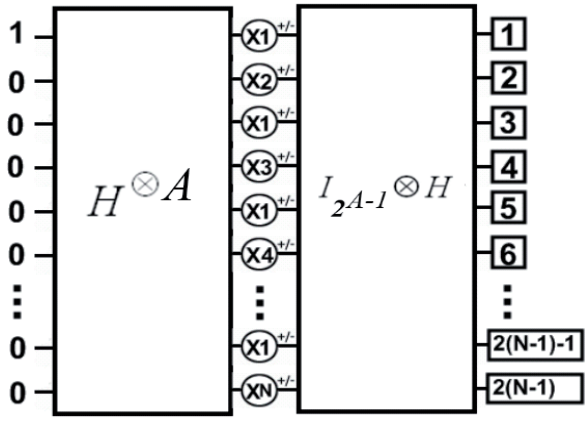

Fig. 7. Quantum query algorithm $Q2$ for computing $\Re 2$ in uniformly distributed manner

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

In this example, we enlarged the number of relation variables, but did not succeeded yet in enlarging the gap between classical and quantum query complexity.

Next, we demonstrate another generalization of the relation $\Re 1$. This time we achieve a gap $2N$ versus N between classical and quantum query complexity.

Definition and behavior of relation $\Re 3$ is similar to relation $\Re 2$ – it associates each input element with $(N\text{-}1)$ output elements from the range set. But this time more variables are involved in the condition of the rule, which defines the relation.

Definition 7. *Relation* $\Re 3 : \{0,1\}^{N^2} \rightarrow \{1, 2, ..., 2(N-1)\}$ *associates each input element from the domain set with (N-1) output elements from the range set according to the following rule:*

$$\forall\ 1 < i \le N : \text{if } ((x_1 \oplus x_2 \oplus ... \oplus x_N) \oplus (x_{(i-1)N+1} \oplus x_{(i-1)N+2} \oplus ... \oplus x_{(i-1)N+N}) = 0)$$
$$\text{then } (X, 2(i-1)-1) \in \Re 3$$
$$\text{otherwise } (X, 2(i-1)) \in \Re 3$$

To compute relation $\Re 3$ in a classical query model, $2N$ queries are required.

Theorem 7. $C_{UD}(\Re 3) = 2N$.

Proof. In order to determine which range set element to include into the result set, it is necessary to know values of all $2N$ variables involved into condition of the rule. A part of classical decision tree is depicted in Fig. 8. All sequentially queried variables are joined into one common query represented in the diagram by ellipses. Multiple arrows corresponding to common query outcomes are exiting these ellipses. □

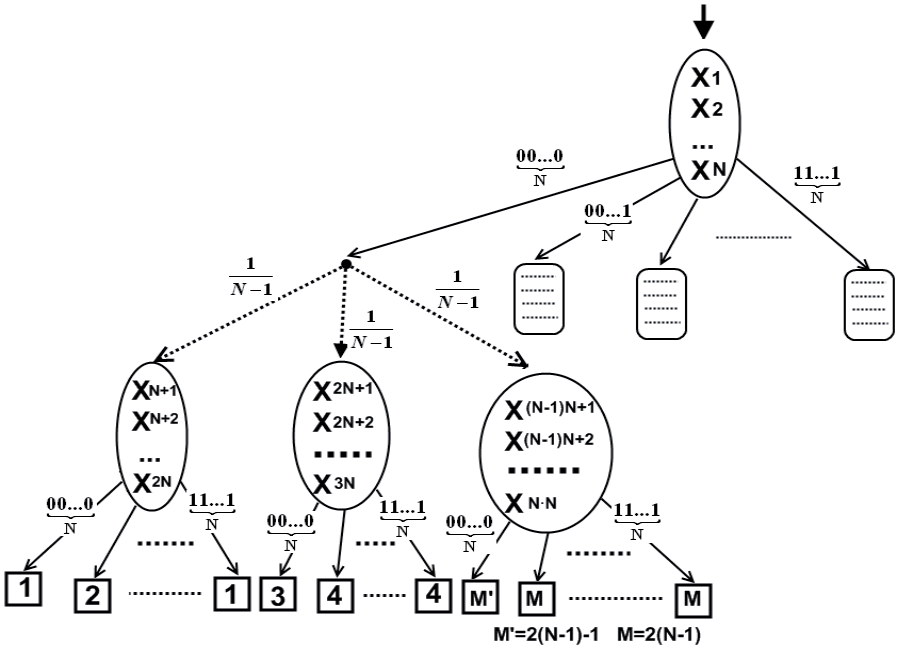

Fig. 8. Classical query algorithm for computing $\Re 3$ in uniformly distributed manner

Theorem 8. $Q_{UD}(\Re 3) = N$.

Proof. General structure of the algorithm remains the same, but we add more queries. Algorithm $Q3$ is presented in Fig. 9. Again, odd amplitudes all have the same set $(x_1, ..., x_N)$ of queried variables assigned. Remaining variables are sequentially assigned to even amplitudes. □

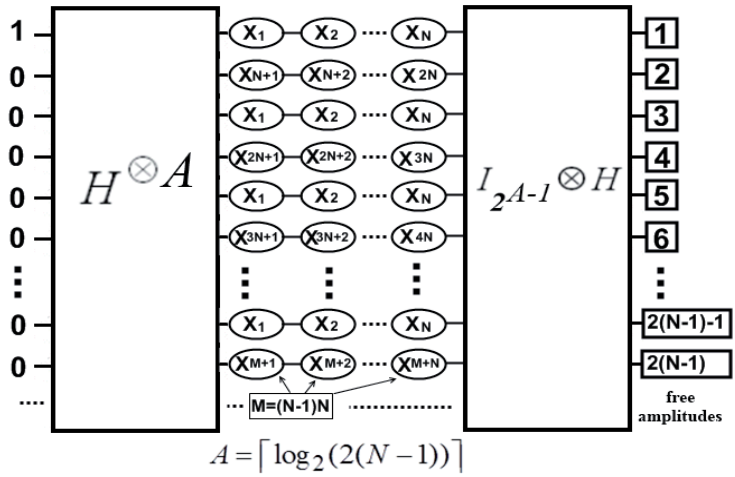

Fig. 9. Quantum query algorithm $Q3$ for computing $\Re 3$ in uniformly distributed manner

In this subsection, we demonstrated approach for extending relations to a larger number of variables. As a result we obtained a complexity separation N versus $2N$, which is the same as the largest separation between quantum exact and classical deterministic query algorithm for total functions known for today. During computing relations correct result is obtained with probability $p = 1$ as well (algorithm always outputs some correct value from the result set). However, the structure of considered relations is based on *XOR* operation. All examples of N versus $2N$ separations for functions, that we are aware of, are directly based on *XOR* as well. We are interested to find different cases, where *XOR* is not involved in obtaining a speed-up.

5.5 Second Example of Computing a Relation

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

Rule 1. If a client has one or three packages besides minimal package, company has to choose one from "1", "2", "3", "4" (probability to choose any gift from the scope has to be equally distributed between options, each having $p = ¼$ to be selected).

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

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

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

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

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

Table 4

Definition of the relation $\Re 4$

X	$\Re 4$ $\{X\}$	X	$\Re 4$ $\{X\}$
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}

5.6 Uniformly Distributed Query Complexity of Relation $\Re 4$

Now, let us discuss the complexity of relation $\Re 4$. We consider computing $\Re 4$ again in the same uniformly distributed manner.

Theorem 9. $C_{UD}(\Re 4) = 3$.

Proof. Proof of this fact consists of two steps. First, we show that it is not possible to build a classical randomized decision tree of depth $d = 2$, which computes $\Re 4$ in uniformly distributed manner. Second, we present a tree, which computes $\Re 4$ using three queries.

Lemma 1. *It is not possible to build a classical randomized decision tree of depth $d = 2$, which computes $\Re 4$ in uniformly distributed manner.*

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 input $X = 0000$ result "1" should be output with probability $p = 1$. It means that there exists a path from root to leaf with value "1", which goes through some two variables: $x_A = 0$ and $x_B = 0$. This path is depicted in Fig. 10. The fact is that it is not possible to select A and B to avoid contradictions with other inputs. Table 5 shows all possible selections of A and B, together with such input Y, which has the same values in positions A and B as $X = 0000$. For these inputs, algorithm goes the same path as for $X = 0000$ and finishes in a leaf with value "1", which is incorrect for Y, thus causing a contradiction with a correct output value for Y. So, it is not possible to build a classical randomized decision tree of depth $d = 2$, which computes $\Re 4$ in uniformly distributed manner. □

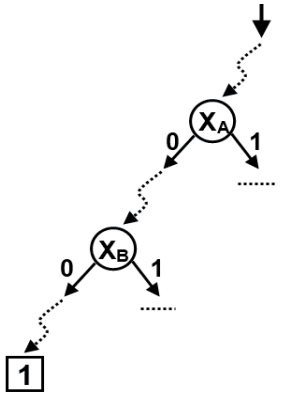

Fig. 10. Path for input X=0000 in a potential classical randomized decision tree of depth $d = 2$ for computing $\Re 4$ in uniformly distributed manner

Table 5

All possible selections of A and B, each causing a contradiction

A	B	Input *Y,* for which algorithm goes through the same path (Fig. 10), which contradicts with X=0000 in output value	$\Re 4$ (X)
1	2	0011	{3}
1	3	0101	{2}
1	4	0110	{4}
2	3	1001	{4}
2	4	1010	{2}
3	4	1100	{3}

Lemma 2. *There exists a classical randomized decision tree, which computes* $\Re 4$ *in uniformly distributed manner using three queries.*

Proof. Classical probabilistic decision tree that computes $\Re 4$ in uniformly distributed manner is shown in Fig. 11.

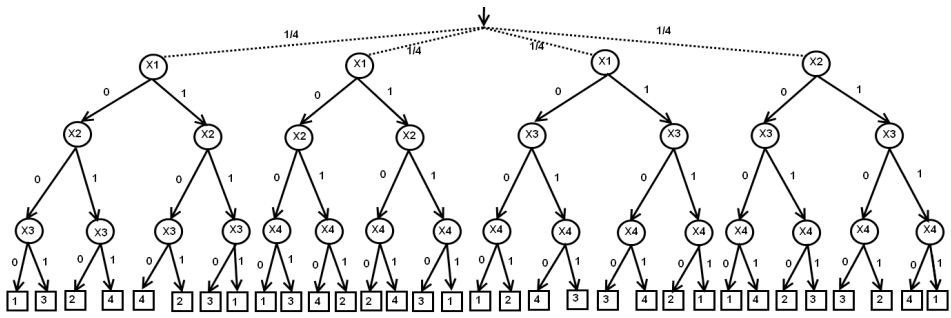

Fig. 11. Probabilistic decision tree that computes $\Re 4$ in uniformly distributed manner

Theorem 10. $Q_{UD}(\Re 4) = 1$.

Proof. Quantum query algorithm $Q4$, which computes $\Re 4$ in the same uniformly distributed manner with one query, is presented in Fig. 12. □

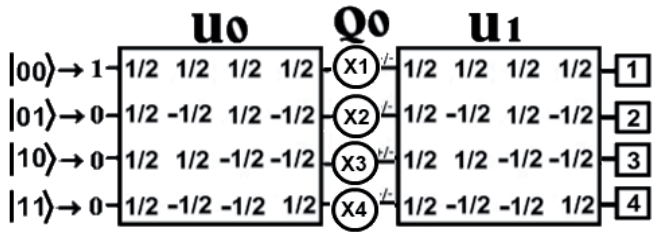

Fig. 12. Quantum query algorithm $Q4$ for computing $\Re 4$ in uniformly distributed manner

We would like to note that definition of the relation $\Re 4$ and algorithm $Q4$ in some sense look similar to the definition and solution of the well-known Deutsch-Jozsa problem [12,13]. Careful reader could figure out this similarity by oneself. However, as we demonstrate further, generalization of that relation is not of that kind anymore.

5.7 First generalization of the relation $\Re 4$

Let us define the relation $\Re_{4N}: \{0,1\}^{4N} \to \{1, 2, 3, 4\}$. Imagine that $4N$ variables are put on four vertical lines (v-lines) in such a way that:

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

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

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

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

3rd have even number of ”1”s, or vice versa: 1st and 4th – even and 2nd and 3rd – odd. For example, input strings 00000110, 00001001, 00111010, 10011111 have the result set {4}.

5. In all other cases, $\Re_{4N}(X) = \{1,2,3,4\}$.

Theorem 11. $Q_{UD}(\Re_{4N}) = N$.

Proof. Quantum algorithm that computes relation in the uniformly distributed manner is presented in Fig. 13. Each quantum query Q_i is defined by the following unitary matrix: □

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

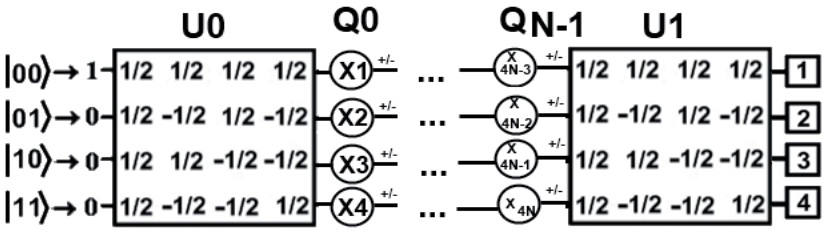

Fig. 13. Quantum query algorithm for computing $\Re_{4N}$ in uniformly distributed manner

Theorem 12. $Q_{UD}(\Re_{4N}) \geq 3N$.

Proof. Let us assume there exists a classical decision tree that computes relation $\Re_{4N}$ by asking $3N-1$ questions. We use all zeros input $X = \vec{0}$ to demonstrate a contradiction. Suppose we queried arbitrary $3N-1$ variables, $N+1$ variables remain unquestioned.

On $4N$-zeros input $X = \vec{0}$ algorithm has to output value ”1” because all v-lines contain zero number of ”1”s. Then, we consider only such inputs that have ”0” in all queried $3N-1$ variables and exactly two ”1”s among remaining unquestioned variables. For all such inputs, algorithm will follow the same path and will finish in the same leaves with output value ”1”.

However, all $N+1$ unquestioned variables cannot be located on one v-line, simply because each v-line consists of N variables. So, there is an input for which two ”1”s among unquestioned variables are located on different v-lines. As we know, the result set in such case is {2} or {3} or {4}. Thus, algorithm outputs incorrect value for this input, this fact contradicts with the initial assumption and implies $Q_{UD}(\Re_{4N}) \geq 3N$. □

5.8 Second generalization of the relation ℜ4

Suppose we are given a relation of N variables $\Re_N : \{0,1\}^N \to \{1,2,...,N\}$, where N is power of 2. This time, we do not provide full definition of the relation; it follows from properties of quantum algorithm described below. We would only like to demonstrate that such generalization is technically possible.

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

Theorem 13. *There is a quantum query algorithm computing specific relation* $\Re_N$ *in a randomly distributed manner asking one question only:* $Q_{RD}(\Re_N) = 1$.

Proof. We add more qubits and sequentially assign variables to amplitudes. Given $N = 2^k, k \in \mathbb{N}$, quantum algorithm starts with k-qubit zero state $|0\rangle$, then applies $N \times N$ Hadamard matrix, N-variable query and finally applies $N \times N$ Hadamard matrix once again. Algorithm is depicted in Fig. 14. □

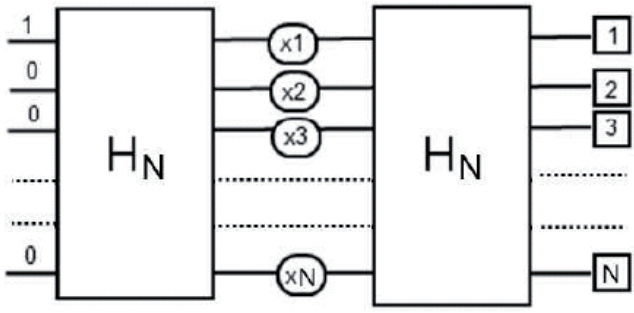

Fig. 14. Generalization of the quantum query algorithm for computing $\mathbf{R_N}$

Theorem 14. $\frac{N}{2} + 1 \le C_{RD}(\Re_N) \le N$.

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

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

If all $x_i = 0$, then $\alpha_1 = 1$, so for the input $X = 00\ldots0$, algorithm outputs ”1” with probability $p = 1$. Let us suppose exactly $N/2$ variables are ”1”s and $\frac{N}{2}$ are ”0”s. In this case, α_1 is precisely zero for all possible combinations. It means that probability to observe result value ”1” for any such input is $p = 0$.

Classical algorithm has to behave in the same way: for input $X = 00\ldots0$, value "1" has to be produced with probability $p = 1$, but for all inputs with exactly $N/2$ "1"s, result value "1" is not allowed to be output at all. This implies we are unable to recognize relation classically by asking only $N/2$ variable values, at least $N/2+1$ queries are required. □

5.9 Third Example of Computing a Relation

In this section, we demonstrate our last example of computing a relation. We present a quantum query algorithm that computes the relation asking two queries in uniformly distributed manner; while classically at least five queries are necessary to compute the same relation.

Important fact is that the structure of a relation and the algorithm for computing it are not based on *XOR* operation. In the area of quantum query algorithms for computing total functions, all examples, that we are aware of at the moment, where quantum query complexity is two times less than classical query complexity are directly based on utilization of *XOR* operation.

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

Relation $\Re 5 : \{0,1\}^6 \rightarrow \{1,2,3,4\}$ is defined by the following set of rules:

- if $x_1 = x_2$ & $x_3 = 0$ & $x_1 = x_5$, then $\Re 5(X) = \{1,2\}$;
- if $x_1 = x_2$ & $x_3 = 0$ & $x_1 \neq x_5$, then $\Re 5(X) = \{3,4\}$;
- if $x_1 = x_2$ & $x_3 = 1$ & $x_1 = x_6$, then $\Re 5(X) = \{2,3\}$;
- if $x_1 = x_2$ & $x_3 = 1$ & $x_1 \neq x_6$, then $\Re 5(X) = \{1,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 = x_4$ & $x_5 = 0$, then $\Re 5(X) = \{1,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 = x_4$ & $x_5 = 1$, then $\Re 5(X) = \{2,3\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 = x_4$ & $x_5 = 0$, then $\Re 5(X) = \{2,3\}$;
- if $x_1 \neq x_2$ & $x_3 = 0$ & $x_1 \neq x_4$ & $x_5 = 1$, then $\Re 5(X) = \{1,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 = x_4$ & $x_6 = 0$, then $\Re 5(X) = \{3,4\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 = x_4$ & $x_6 = 1$, then $\Re 5(X) = \{1,2\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 \neq x_4$ & $x_6 = 0$, then $\Re 5(X) = \{1,2\}$;
- if $x_1 \neq x_2$ & $x_3 = 1$ & $x_1 \neq x_4$ & $x_6 = 1$, then $\Re 5(X) = \{3,4\}$.

Theorem 15. $5 \leq C_{UD}(\Re 5) \leq 6$.

Proof. Let us assume there exists a classical randomized decision tree that computes relation $\Re 5$ by asking four queries. We analyze algorithm behavior for the certain input X=010000. According to the definition of the uniformly distributed algorithm, decision tree must output correct values from the result set for each input with equal probability. It means that there has to be a path in the tree, which goes through at most four variable nodes, follows arrows with variable values corresponding to values of X=010000 and finishes in a leaf with the output value «4». On the other hand, for any choice of four

variables for that path, there exists another input X, which equals X in selected four variables values, but does not have «4» among the result set values according to the definition of relation $\Re 5$. All such contradicting inputs are listed in Table 6. For any such input, computation goes through the same path in the decision tree as for X=010000 and finishes in a leaf with incorrect value «4». It is a contradiction, so assumption is wrong and classical randomized decision tree that computes relation $\Re 5$ using only four queries does not exist. □

Table 6

Proof of Theorem 15: contradictions in result set values

Path variables	Input X' contradicting with X=010000	$\Re 5(X')$
x_1,x_2,x_3,x_4	010010	{2,3}
x_1,x_2,x_3,x_5	010100	{2,3}
x_1,x_2,x_3,x_6	010010	{2,3}
x_1,x_2,x_4,x_5	011001	{1,2}
x_1,x_2,x_4,x_6	010010	{2,3}
x_1,x_2,x_5,x_6	010100	{2,3}
x_1,x_3,x_4,x_5	000000	{1,2}
x_1,x_3,x_4,x_6	000000	{1,2}
x_1,x_3,x_5,x_6	000000	{1,2}
x_1,x_4,x_5,x_6	000000	{1,2}
x_2,x_3,x_4,x_5	000000	{1,2}
x_2,x_3,x_4,x_6	000000	{1,2}
x_2,x_3,x_5,x_6	000000	{1,2}
x_2,x_4,x_5,x_6	000000	{1,2}
x_3,x_4,x_5,x_6	000000	{1,2}

Theorem 16. $Q_{UD}(\Re 5) = 2$.

Proof. Quantum query algorithm that computes $\Re 5$ with two queries is presented in Fig. 15. Sign "+" inside question circle signifies that none variable impacts the value of corresponding amplitude. □

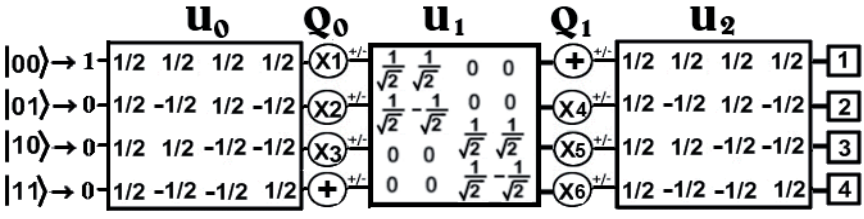

Fig. 15. Quantum query algorithm $Q5$ for computing $\Re 5$ in uniformly distributed manner

6 A Note on Computing Relations in a Definite Manner

In this section, we discuss the first type of query algorithms for relations, which compute relations in a definite manner. Are there prospects to obtain a large separation between classical and quantum complexity?

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

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

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

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

7 Conclusion

In this paper, we considered computing mathematical relations instead of Boolean functions in a query model. Various general computational problems and tasks of certain type in software engineering may be represented in terms of relations. We proposed three types of a query algorithm for computing relations with different output behavior.

We demonstrated several examples of computing relations in classical and quantum versions of a query model.

In the first example, the definition of relation is based on *XOR* operation. We generalized the basic relation and obtained an example, when quantum query algorithm computes relation with N^2 variables using N queries, while $2N$ queries are required in the classical case. This result repeats the largest separation between quantum exact and classical deterministic query complexity for functions that is known for today.

In the second example, a quantum query complexity for relation is more than two times less than classical query complexity for the same relation. However, the considered relation has a property of having inputs for which the result set consists of all range set elements.

In the third example, we considered finite six-variable relation, which is not based on *XOR* operation and there are no inputs for which result set consists of all range set elements. These properties make this relation very interesting. For this relation, quantum query complexity is also more than two times less than classical query complexity.

Finally, we discussed the specifics of computing relations in a definite manner and concluded that the task of computing relations in a distributed manner is more promising for enlarging the gap between classical and quantum query complexity.

Results presented in this paper build a foundation for further investigation. The main goal, which we are looking to achieve, is to construct examples with larger complexity separation between classical and quantum query algorithm complexity. The most important work direction is to develop a technique for proving complexity lower bounds for computing relations in a classical query model.

8 Acknowledgments

We would like to thank our supervisor Rūsiņš Freivalds for familiarizing us with quantum computation and for permanent support and advising.

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

References

1. Buhrman, H., de Wolf, R.: Complexity Measures and Decision Tree Complexity: A Survey. Theoretical Computer Science, v. 288(1): 21-43 (2002)
2. Shor, P. W.: Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing, 26(5):1484-1509 (1997)
3. Grover L.K.: From Schrödinger‘s equation to quantum search algorithm, American Journal of Physics, 69(7): 769-777 (2001)de Wolf, R.: Quantum Computing and Communication Complexity. University of Amsterdam (2001)
4. Ambainis, A.: Quantum query algorithms and lower bounds (survey article). In Proceedings of FOTFS III, Trends on Logic, vol. 23, pp. 15-32 (2004)
5. Kaye, R., Laflamme, R., Mosca, M.: An Introduction to Quantum Computing. Oxford (2007)
6. Ambainis, A., Childs, A., Reichardt, B., Spalek, R., Zhang, S.: Any AND-OR formula of size N can be evaluated in time O(N^{1/2+epsilon}) on a quantum computer. SIAM J. Comput. Volume 39, Issue 6, pp. 2513-2530 (2010).
7. Vasilieva, A., Mischenko-Slatenkova, T.: Quantum Query Algorithms for Conjunctions. Proc. of the UC 2010, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol. 6079/2010, ISBN: 978-3-642-13522-4, pp. 140-151 (2010)
8. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press, (1997)
9. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
10. Weisstein, E. W.: Relation. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/Relation.html
11. D. Deutsch and R. Jozsa: Rapid solutions of problems by quantum computation. In Proceedings of the Royal Society of London, volume A 439, pp. 553-558 (1992)
12. R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca: Quantum algorithms revisited. In Proceedings of the Royal Society of London, volume A 454, pp. 339–354 (1998)