Computing Boolean Functions via Quantum Hashing

Farid Ablayev(✉) and Alexander Vasiliev

Kazan Federal University, Kazan, Russian Federation
fablayev@gmail.com

Abstract. In this paper we show a computational aspect of the quantum hashing technique. In particular we apply it for computing Boolean functions in the model of read-once quantum branching programs based on the properties of specific polynomial presentation of those functions.

1 Introduction

Hashing is widely used in computer science, it is especially useful in cryptographic protocols and data integrity check. In [1] we have introduced a non-binary quantum hash function for cryptographic scenarios. For instance, the proposed quantum hashing is a suitable one-way function for quantum digital signature protocol from [10]. In this paper we consider another application of the quantum hashing and use it to construct efficient quantum algorithms in a restricted computational model.

Due to severe limits of existing physical implementations of quantum computer it is natural to consider the restricted models of quantum computations. The one we consider in this paper is based upon *quantum branching programs*. Two variants of quantum branching programs were introduced by Ablayev, Gainutdinova, Karpinski [3] (*leveled programs*), and by Nakanishi, Hamaguchi, Kashiwabara [12] (*non-leveled programs*). Later it was shown by Sauerhoff [14] that these two models are polynomially equivalent. The most commonly used restricted variant of quantum branching programs is the model of *Ordered Read-Once Quantum Branching Programs*. In computer science this model is also known as Ordered Binary Decision Diagrams (OBDDs). This restriction implies that each input variable may be read at most once, which is the least possible for any function essentially depending on its variables. Thus, the read-once restriction corresponds to minimizing of computational steps for quantum algorithms.

Essentially, the model of quantum OBDDS is a non-uniform equivalent of one-way quantum finite automata (QFA) and thus the technique given in this paper can be used in the QFA model as well.

In order to compute Boolean functions in the quantum OBDD model we exploit the specific polynomial presentation, which we have called *characteristic* [5]. The polynomial presentations of Boolean functions are widely used in theoretical computer science. For instance, an algebraic transformation of Boolean functions has been applied in [11] and [7] for verification of Boolean functions. In the quantum

C.S. Calude et al. (Eds.): Gruska Festschrift, LNCS 8808, pp. 149–160, 2014.
DOI: 10.1007/978-3-319-13350-8_11

setting polynomial representations were used for proving lower bounds on communication complexity in [8] as well as for investigating query complexity in [16]. Our approach combines the ideas similar to the definition of characteristic polynomial from [11], [7] and to the notion of *zero-error polynomial* (see, e.g. [16]).

In this paper we show how the proposed quantum hashing can be used to compute Boolean functions given by their polynomials in a very restricted computational model of quantum OBDDs. Due to the known general lower bound on the complexity of quantum OBDDs some of our algorithms turn out to be optimal.

2 Quantum Branching Programs

We use the notation $|i\rangle$ for the vector from Hilbert space $\mathcal{H}^d$, which has a 1 on the i-th position and 0 elsewhere. An orthonormal basis $|1\rangle,\ldots,|d\rangle$ is usually referred to as the *standard computational basis*. In this paper we consider all quantum transformations and measurements with respect to this basis.

Definition 1. *A Quantum Branching Program Q over the Hilbert space $\mathcal{H}^d$ is defined as*

$$Q = \langle T, |\psi_0\rangle, \mathrm{Accept}\rangle \ , \tag{1}$$

where T is a sequence of l instructions: $T_j = \left(x_{i_j}, U_j(0), U_j(1)\right)$ is determined by the variable x_{i_j} tested on the step j, and $U_j(0)$, $U_j(1)$ are unitary transformations in $\mathcal{H}^d$.

Vectors $|\psi\rangle \in \mathcal{H}^d$ are called states (state vectors) of Q, $|\psi_0\rangle \in \mathcal{H}^d$ is the initial state of Q, and Accept $\subseteq \{1, 2, \ldots d\}$ *is the set of indices of accepting basis states.*

We define a computation of Q on an input $\sigma = \sigma_1 \ldots \sigma_n \in \{0,1\}^n$ as follows:

1. *A computation of Q starts from the initial state $|\psi_0\rangle$;*
2. *The j-th instruction of Q reads the input symbol σ_{i_j} (the value of x_{i_j}) and applies the transition matrix $U_j = U_j(\sigma_{i_j})$ to the current state $|\psi\rangle$ to obtain the state $|\psi'\rangle = U_j(\sigma_{i_j})|\psi\rangle$;*
3. *The final state is*

$$|\psi_\sigma\rangle = \left(\prod_{j=l}^{1} U_j(\sigma_{i_j})\right) |\psi_0\rangle \ . \tag{2}$$

4. *After the l-th (last) step of quantum transformation Q measures its configuration $|\psi_\sigma\rangle = (\alpha_1, \ldots, \alpha_d)^T$, and the input σ is accepted with probability*

$$Pr_{\mathrm{accept}}(\sigma) = \sum_{i \in \mathrm{Accept}} |\alpha_i|^2 \ . \tag{3}$$

Note, that using the set *Accept* we can construct M_{accept} – a projector on the accepting subspace $\mathcal{H}^d_{accept}$ (i.e. a diagonal zero-one projection matrix, which determines the final projective measurement). Thus, the accepting probability can be re-written as

$$Pr_{accept}(\sigma) = \langle \psi_\sigma M^\dagger_{accept} \,|\, M_{accept}\psi_\sigma\rangle = ||M_{accept}|\psi_\sigma\rangle||_2^2 \ . \tag{4}$$

Circuit Representation. Quantum algorithms are usually given by using quantum circuit formalism [9], [17], because this approach is quite straightforward for describing such algorithms.

We propose, that a QBP represents a classically-controlled quantum system. That is, a QBP can be viewed as a quantum circuit aided with an ability to read classical bits as control variables for unitary operations.

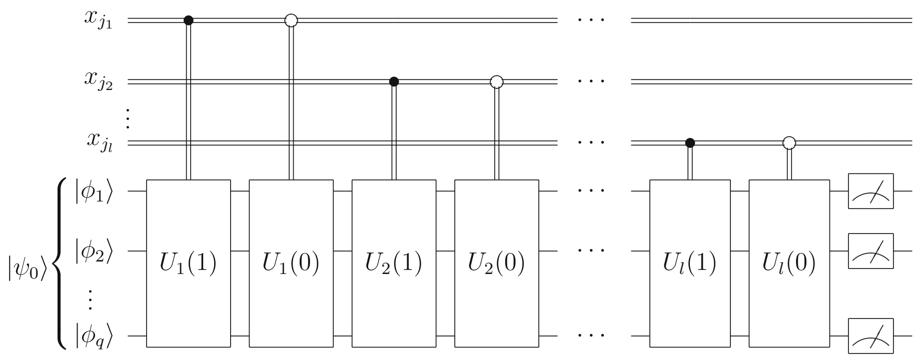

Fig. 1. Circuit presentation of a quantum branching program. Here $x_{i_1}, \ldots, x_{i_l}$ is the sequence of (not necessarily distinct) variables denoting classical control (input) bits. Using the common notation single wires carry quantum information and double wires denote classical information and control.

Example. As an example consider the Boolean function $MOD_m(x_1, \ldots, x_n)$ which tests whether the number of ones in it's input is a multiple of m. For this function the simple algorithm can be proposed (see Figure 2).

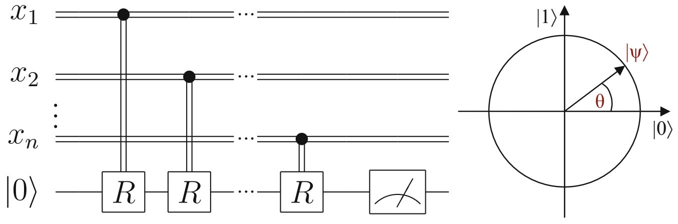

Fig. 2. Quantum branching program for MOD_m Boolean function. Here R denotes the rotation by an angle $\theta = \pi/m$ about the $\hat{y}$ axis of the Bloch sphere.

The algorithm starts with a qubit in basis state $|0\rangle$. At j-th step the value of x_j is tested. Upon input symbol 0 identity transformation I is applied. But if the value of x_j is 1, then the state of the qubit is transformed by the operator R, rotating it by the angle proportional to π/m.

The final state is measured in the standard computational basis. The input $\sigma = \sigma_1 \dots \sigma_n$ is accepted if the result is the basis state $|0\rangle$, otherwise the input σ is rejected. For arbitrary input σ the acceptance probability equals to

$$Pr_{accept}(\sigma) = \cos^2\left(\frac{\pi \sum_i \sigma_i}{m}\right) . \tag{5}$$

Thus, if $MOD_m(\sigma) = 1$ then $Pr_{accept}(\sigma) = 1$. If $MOD_m(\sigma) = 0$ then the probability of erroneously obtaining the $|0\rangle$ can be close to 1, but this can be improved by using more qubits.

Complexity Measures. The width of a QBP Q, denoted by width(Q), is the dimension d of the corresponding state space $\mathcal{H}^d$, and the length of Q, denoted by length(Q), is the number l of instructions in the sequence T.

In this paper we're mostly interested in another important complexity for a QBP Q – a number of quantum bits, denoted by qubits(Q), physically needed to implement a corresponding quantum system with classical control. From definition it follows that $\log \text{width}(Q) \leq \text{qubits}(Q)$.

Acceptance Criteria. A QBP *Q computes the Boolean function f with bounded error* if there exists an $\epsilon \in (0, 1/2)$ (called *margin*) such that for all inputs the probability of error is bounded by $1/2 - \epsilon$.

In particular, we say that a QBP *Q computes the Boolean function f with one-sided error* if there exists an $\epsilon \in (0, 1)$ (called *error*) such that for all $\sigma \in f^{-1}(1)$ the probability of Q accepting σ is 1 and for all $\sigma \in f^{-1}(0)$ the probability of Q erroneously accepting σ is less than ϵ.

Read-Once Branching Programs. Read-once BPs is a well-known restricted variant of branching programs [15].

Definition 2. *We call a* QBP *Q a quantum* OBDD (QOBDD) *or read-once* QBP *if each variable $x \in \{x_1, \dots, x_n\}$ occurs in the sequence T of transformations of Q at most once.*

For the rest of the paper we are only interested in QOBDDs, i.e. the length of all programs would be n (the number of input variables). Note that for OBDD model $size(Q) = n \cdot width(Q)$ and therefore we are mostly interested in the width of quantum OBDDs.

The "obliviousness" is inherent for a QBP and therefore this definition is consistent with the usual notion of an OBDD.

General Lower Bound. The following general lower bound on the width of QOBDDs was proven in [4].

Theorem 1. *Let $f(x_1, \ldots, x_n)$ be a Boolean function computed by a quantum read-once branching program Q with bounded error for some margin ϵ. Then*

$$\mathrm{width}(Q) \geq \frac{\log \mathrm{width}(P)}{2 \log \left(1 + \frac{1}{\epsilon}\right)}, \tag{6}$$

where P is a deterministic OBDD of minimal width computing $f(x_1, \ldots, x_n)$.

That is, the width of a quantum OBDD cannot be asymptotically less than logarithm of the width of the minimal deterministic OBDD computing the same function. And since the deterministic width of many "natural" functions is exponential [15], we obtain the linear lower bound for these functions.

Let bits(P) be the number of bits (memory size) required to implement the minimal deterministic OBDD P for f and Q is an arbitrary quantum OBDD computing the same function.

Then Theorem 1 implies the following lower bound in terms of the number of bits and qubits as the complexity measure.

Corollary 1.

$$\mathrm{qubits}(Q) = \Omega(\log \mathrm{bits}(P)) \ . \tag{7}$$

3 Quantum Hashing

In this section we recall a quantum hashing function from [1].

Let $q = 2^n$ and $B = \{b_1, b_2, \ldots, b_d\} \subset \mathbb{Z}_q$. We define a quantum hash function $\psi_{q,B} : \{0,1\}^n \to (\mathcal{H}^2)^{\otimes(\log d+1)}$ as follows. For an input $x \in \{0,1\}^n$ we let

$$|\psi_{q,B}(x)\rangle = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} |i\rangle \left(\cos \frac{2\pi b_i x}{q} |0\rangle + \sin \frac{2\pi b_i x}{q} |1\rangle \right) \ . \tag{8}$$

It follows from this definition that the quantum hash $|\psi_{q,B}(x)\rangle$ of an n-bit string x consists of $\log d + 1$ qubits. We will show that d can be about $O(n)$ without loosing the quality of hashing.

The set $B = \{b_1, b_2, \ldots, b_d\}$ of hashing parameters not only defines the size of the hash but also gives the function $\psi_{q,B}$ an ability to withstand collisions, i.e. to distinguish different hashes with bounded error probability. We have called this property *δ-resistance.*

Formally, for $\delta \in (0,1)$ we call a function $\psi : \mathbb{X} \to (\mathcal{H}^2)^{\otimes s}$ δ-resistant if for any pair w, w' of different inputs

$$|\langle \psi(w) \,|\, \psi(w') \rangle| \leq \delta \ . \tag{9}$$

The value of δ for the hash function $\psi_{q,B}$ entirely depends on q (which is fixed here by the size of the input) and the set B, i.e. $\delta = \delta(q, B)$. In [1] we have shown a construction for the set of polylogarithmic size (in n) based on [13]. We have also proved the following result.

Theorem 2. *For arbitrary $\delta \in (0,1)$ there exists a set $B = \{b_1, b_2, \dots, b_d\}$ of size $d = \lceil (2/\delta^2) \ln(2q) \rceil$ such that quantum hash function $\psi_{q,B}$ is a δ-resistant.*

In other words, for arbitrary $\delta \in (0,1)$ it is possible construct a δ-resistant quantum hash function $\psi_{q,B}$ that would produce an $\log d + 1 = O(\log \log q) = O(\log n)$-qubit hash out of n-bit input.

Implementation of the quantum hashing. In order to describe the implementation of the quantum hashing we introduce the following notations.

We define a *Compound Controlled Rotation* operator (CCR):

$$CCR_{q,B}(\theta) = CCR_{q,B,1}(\theta) \cdot CCR_{q,B,2}(\theta) \cdots CCR_{q,B,d}(\theta), \qquad (10)$$

where operator $CCR_{q,B,i}(\theta)$ rotates the target qubit by the angle θ if the control qubits were in the state $|i\rangle$ and is given in Figure 3.

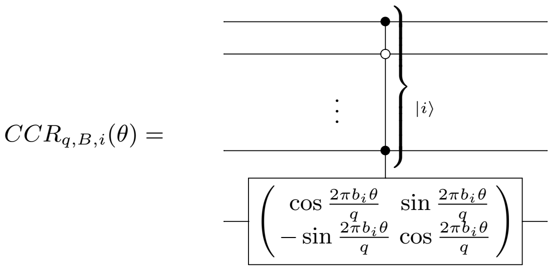
$CCR_{q,B,i}(\theta) =$
$|i\rangle$
$\begin{pmatrix} \cos\frac{2\pi b_i\theta}{q} & \sin\frac{2\pi b_i\theta}{q} \\ -\sin\frac{2\pi b_i\theta}{q} & \cos\frac{2\pi b_i\theta}{q} \end{pmatrix}$

Fig. 3. A circuit for the operator $CCR_{q,B,i}(\theta)$, that rotates the target qubit by the angle θ if the control qubits were in the state $|i\rangle$. Here, the single-qubit rotation is made around the $\hat{y}$ axis of the Bloch sphere.

The compound operator $CCR_{q,B}(\theta)$ obviously has the following properties.

Property 1. CCR-property

$$\begin{aligned} &CCR_{q,B}(0) = I, \\ &CCR_{q,B}(\theta)|\psi_{q,B}(x)\rangle = |\psi_{q,B}(x+\theta)\rangle, \\ &CCR_{q,B}(\theta_1)CCR_{q,B}(\theta_2) = CCR_{q,B}(\theta_1+\theta_2) \ . \end{aligned} \qquad (11)$$

Thus, the procedure of quantum hashing the input w by the function $\psi_{q,B}$ consists of the following steps:

0. Initialization of the $\log d + 1$ qubits in the state $|0\dots0\rangle|0\rangle$.
1. Application of Hadamard transform to the first $\log d$ qubits:

$$\frac{1}{\sqrt{d}} \sum_{i=1}^{d} |i\rangle|0\rangle = |\psi_{q,B}(0)\rangle \ . \qquad (12)$$

2. Application of $CCR_{q,B}(w)$ creates the quantum hash of the input bit string $w = w_0 \dots w_{n-1}$, which is also treated as a number $w = w_0 + w_1 2^1 + \dots + w_{n-1} 2^{n-1}$:

$$CCR_{q,B}(w)|\psi_{q,B}(0)\rangle = |\psi_{q,B}(w)\rangle \ . \tag{13}$$

Note, that for the model of quantum branching programs this step consists of n substeps: for each input bit w_j there is an instruction $\langle w_j, I, CCR_{q,B}(2^j)\rangle$ of the quantum branching program, i.e. when $w_j = 1$ we apply $CCR_{q,B}(2^j)$, and do nothing otherwise. Obviously,

$$CCR_{q,B}(w) = CCR_{q,B}(w_0) \cdot CCR_{q,B}(w_1 2^1) \cdots CCR_{q,B}(w_{n-1} 2^{n-1}) \ . \tag{14}$$

Thus, an overall number of controlled rotations $CCR_{q,B,i}(\theta)$ is $nd = O(n^2)$.

From the description above it follows that the input bits are read only once, and the quantum branching program is actually a quantum OBDD. An illustrative presentation for this program is given in Figure 4.

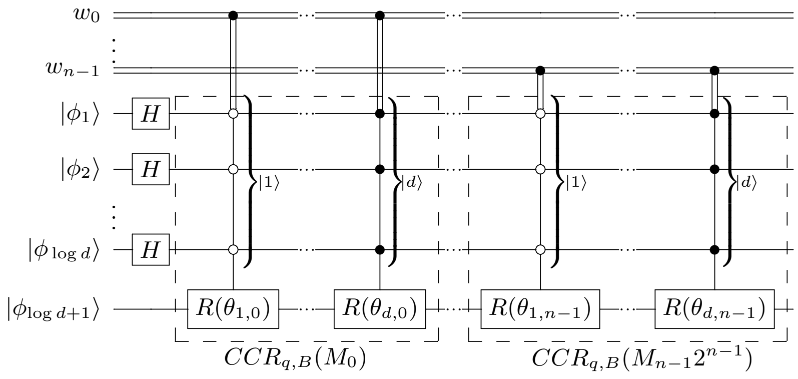

Fig. 4. A quantum OBDD in circuit presentation, that hashes an n-bit input $w = w_0 \dots w_{n-1}$ into the state $|\psi_{q,B}(M)\rangle$ of $O(\log n)$ qubits. $R(\theta_{i,j})$ denotes a rotation by an angle $\frac{4\pi b_i 2^j}{q}$ around the $\hat{y}$ axis of the Bloch sphere.

Physical implementation. In [2,6] we have proposed an effective physical implementation of compound multiply controlled operators for the model of solid state quantum computer on multiatomic ensembles in the QED cavity. $CCR_{q,B,i}(\theta)$ is exactly such operator and thus can be accelerated in this architecture.

REVERSE-test. Whenever we need to check if a quantum state $|\psi(w)\rangle$ is a hash of a classical string v, one can use the procedure that we call a *REVERSE-test* [1].

Essentially the test applies the procedure that inverts the creation of a quantum hash, i.e. it "uncomputes" the hash to the initial state (usually the all-zero state).

Formally, let the procedure of quantum hashing the string w be given by unitary transformation $U(w)$, applied to initial state $|0\rangle$, i.e. $|\psi(w)\rangle = U(w)|0\rangle$. Then the REVERSE-test, given v and $|\psi(w)\rangle$, applies $U^{-1}(v)$ to the state $|\psi(w)\rangle$ and measures the resulting state. It outputs $v = w$ iff the measurement outcome is $|0\rangle$. So, if $v = w$, then $U^{-1}(v)|\psi(w)\rangle$ would always give $|0\rangle$, and REVERSE-test would give the correct answer. Otherwise, by δ-resistance property

$$\langle 0 | U^{-1}(v)\psi(w)\rangle < \delta \ , \tag{15}$$

which bounds the probability of erroneously outputting $v = w$.

Overall, this test has one-sided error bounded by δ^2 if the quantum hash function is δ-resistant.

In case of quantum hash function $CCR_{q,B}$ the REVERSE-test consists of the following steps.

1. Application of $CCR_{q,B}(-v)$ to the state $|\psi(w)\rangle$.
2. Application of the Hadamard transform to all but the last qubit.
3. Measurement of the resulting state.

4 Characteristic Polynomials for Boolean Functions

In this section we recall the definition of the characteristic polynomial for a Boolean function proposed in [5].

Definition 3. *We call a polynomial $g(x_1, \dots, x_n)$ over the ring $\mathbb{Z}_q$ a characteristic polynomial of a Boolean function $f(x_1, \dots, x_n)$ and denote it g_f when for all $\sigma \in \{0,1\}^n$ $g_f(\sigma) = 0$ iff $f(\sigma) = 1$.*

Note, that such a polynomial always exists.

Lemma 1. *For any Boolean function f of n variables there exists a characteristic polynomial g_f over $\mathbb{Z}_{2^n}$.*

Proof. One way to construct such characteristic polynomial g_f is transforming a sum of products representation for $\neg f$.

Let $K_1 \vee \ldots \vee K_l$ be a sum of products for $\neg f$ and let $\tilde{K_i}$ be a product of terms from K_i (negations $\neg x_j$ are replaced by $1 - x_j$). Then $\tilde{K_1} + \ldots + \tilde{K_l}$ is a characteristic polynomial over $\mathbb{Z}_{2^n}$ for f since it equals 0 $\iff$ all of $\tilde{K_i}$ (and thus K_i) equal 0. This happens only when the negation of f equals 0.

Generally, there are many polynomials for the same function. For example, the function EQ_n, which tests the equality of two n-bit binary strings, has the following polynomial over $\mathbb{Z}_{2^n}$:

$$\sum_{i=1}^{n} (x_i(1-y_i) + (1-x_i)y_i) = \sum_{i=1}^{n} (x_i + y_i - 2x_iy_i) \ . \tag{16}$$

On the other hand, the same function can be represented by the polynomial

$$\sum_{i=1}^{n} x_i 2^{i-1} - \sum_{i=1}^{n} y_i 2^{i-1} \ . \tag{17}$$

5 Computing Boolean Functions with Quantum Hashing

Now we describe the class of Boolean functions that can be efficiently computed in the quantum OBDD model using the quantum hashing technique.

Let $f(x_1, \ldots, x_n)$ be a Boolean function and g be its characteristic polynomial. The following theorem holds.

Theorem 3. *Let $\delta \in (0,1)$. If there exists is a linear polynomial g for a Boolean function f over $\mathbb{Z}_q$, then f can be computed with one-sided error δ^2 by a quantum OBDD on $O(\log\log q + \log 1/\epsilon)$ qubits.*

Proof. The key idea is to evaluate the characteristic polynomial and hash the result simultaneously while reading the input. Below we show that this can be easily done when the polynomial is linear. After the hash is prepared the value of the Boolean function f can be obtained by performing the REVERSE-test, checking whether 0 is hashed or not.

Since the polynomial g is linear, i.e. $g = c_1x_1 + \ldots c_nx_n + c_0$, hashing its value can be done by a sequence of $CCR_{q,B}$ operators:

$$CCR_{q,B}(w) = CCR_{q,B}(c_0) \cdot CCR_{q,B}(c_1x_1) \cdots CCR_{q,B}(c_nx_n) \ , \tag{18}$$

and this is easily done while reading the input only once.

Then the REVERSE-test applies $CCR_{q,B}(0)$, which is identity operator, and finishes with Hadamard transform and measurements. It outputs the correct answer with the one-sided error probability δ^2.

Thus, f can be computed with one-sided error δ^2 by a quantum OBDD on s qubits, where $s = O(\log\log q + \log 1/\epsilon)$.

5.1 Examples

The following functions have the aforementioned linear polynomials and thus are effectively computed with quantum hashing.

MOD_m The function MOD_m tests whether the number of 1's in the input is 0 modulo m. The linear polynomial over $\mathbb{Z}_m$ for this function is

$$\sum_{i=1}^{n} x_i.$$

The lower bound for the width of deterministic OBDDs computing this function is $\Omega(m)$ [15]. Thus, our method provides an exponential advantage of quantum OBDD over any deterministic one.

MOD'_m This function is the same as MOD_m, but the input is treated as binary number. Thus, the linear polynomial is

$$\sum_{i=1}^{n} x_i 2^{i-1}.$$

The lower and upper bounds are equal to those of MOD_m.

EQ_n The function EQ_n, which tests the equality of two n-bit binary strings, has the following polynomial over $\mathbb{Z}_{2^n}$

$$\sum_{i=1}^{n} x_i 2^{i-1} - \sum_{i=1}^{n} y_i 2^{i-1}.$$

This function is easy in the deterministic case for a clever choice of the variable ordering. But for the ordering, where all of x's are tested first, it is exponentially hard. In quantum setting, this function can be effectively computed regardless of the variable ordering.

$Palindrome_n(x_1, \dots, x_n)$ This function tests the symmetry of the input, i.e. whether $x_1 x_2 \dots x_{\lfloor n/2 \rfloor} = x_n x_{n-1} \dots x_{\lceil n/2 \rceil + 1}$ or not. The polynomial over $\mathbb{Z}_{2^{\lfloor n/2 \rfloor}}$ is

$$\sum_{i=1}^{\lfloor n/2 \rfloor} x_i 2^{i-1} - \sum_{i=\lceil n/2 \rceil}^{n} x_i 2^{n-i}.$$

The situation with lower and upper bounds for this function is similar to that of EQ_n.

$PERM_n$ The *Permutation Matrix* test function ($PERM_n$) is defined on n^2 variables x_{ij} ($1 \le i, j \le n$). It tests whether the input matrix contains exactly one 1 in each row and each column. Here is a polynomial over $\mathbb{Z}_{(n+1)^{2n}}$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \left((n+1)^{i-1} + (n+1)^{n+j-1} \right) - \sum_{i=1}^{2n} (n+1)^{i-1}.$$

Note, that this function cannot be effectively computed by a deterministic OBDD – the lower bound is $\Omega(2^n n^{-5/2})$ regardless of the variable ordering [15]. The width of the best known probabilistic OBDD, computing this function with one-sided error, is $O(n^4 \log n)$ [15]. Our algorithm has the width $O(n \log n)$. Since the lower bound $\Omega(n - \log n)$ follows from Theorem 1, our algorithm is almost optimal.

The following functions have linear polynomials as well, but we are not aware of exponential lower bounds in the deterministic case.

$Period_n^s(x_0,\ldots,x_{n-1})$ This function equals 1 iff $x_i = x_{i+s \bmod n}$ for all $i \in \{0,\ldots,n-1\}$. The polynomial over $\mathbb{Z}_{2^n}$ is

$$\sum_{i=0}^{n-1} x_i \left(2^i - 2^{i-s \bmod n}\right).$$

$Semi-Simon_n^s(x_0,\ldots,x_{n-1})$ This function equals 1 iff $x_i = x_{i\oplus s}$ for all $i \in \{0,\ldots,n-1\}$. The polynomial over $\mathbb{Z}_{2^n}$ is

$$\sum_{i=0}^{n-1} x_i \left(2^i - 2^{i\oplus s}\right).$$

Acknowledgments. The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University. Work was in part supported by the Russian Foundation for Basic Research (under the grants 12-07-97016, 14-07-00878).

References

1. Ablayev, F.M., Vasiliev, A.V.: Cryptographic quantum hashing. Laser Physics Letters 11(2), 025202 (2014). http://stacks.iop.org/1612-202X/11/i=2/a=025202
2. Ablayev, F., Andrianov, S., Moiseev, S., Vasiliev, A.: Encoded universality of quantum computations on the multi-atomic ensembles in the qed cavity. Tech. Rep. arXiv:1109.0291 [quant-ph]. Cornell University Library (September 2011). http://arxiv.org/abs/1109.0291
3. Ablayev, F., Gainutdinova, A., Karpinski, M.: On Computational Power of Quantum Branching Programs. In: Freivalds, R. (ed.) FCT 2001. LNCS, vol. 2138, pp. 59–70. Springer, Heidelberg (2001)
4. Ablayev, F., Gainutdinova, A., Karpinski, M., Moore, C., Pollett, C.: On the computational power of probabilistic and quantum branching programs of constant width. Information and Computation 203, 145–162 (2005). http://dx.doi.org/10.1016/j.ic.2005.04.003
5. Ablayev, F., Vasiliev, A.: Algorithms for quantum branching programs based on fingerprinting. Electronic Proceedings in Theoretical Computer Science 9, 1–11 (2009). http://arxiv.org/abs/0911.2317
6. Ablayev, F., Andrianov, S., Moiseev, S., Vasiliev, A.: Quantum computer with atomic logical qubits encoded on macroscopic three-level systems in common quantum electrodynamic cavity. Lobachevskii Journal of Mathematics 34(4), 291–303 (2013). http://dx.doi.org/10.1134/S1995080213040094
7. Agrawal, V., Lee, D., Wozniakowski, H.: Numerical computation of characteristic polynomials of boolean functions and its applications. Numerical Algorithms 17, 261–278 (1998). http://dx.doi.org/10.1023/A:1016632423579
8. Buhrman, H., Cleve, R., Watrous, J., de Wolf, R.: Quantum fingerprinting. Phys. Rev. Lett. 87(16), 167902 (2001). www.arXiv.org/quant-ph/0102001v1
9. Deutsch, D.: Quantum computational networks. Royal Society of London Proceedings Series A 425, 73–90 (1989). http://dx.doi.org/10.1098/rspa.1989.0099

10. Gottesman, D., Chuang, I.: Quantum digital signatures. Tech. Rep. arXiv:quant-ph/0105032. Cornell University Library (November 2001). http://arxiv.org/abs/quant-ph/0105032
11. Jain, J., Abraham, J.A., Bitner, J., Fussell, D.S.: Probabilistic verification of boolean functions. Formal Methods in System Design **1**, 61–115 (1992)
12. Nakanishi, M., Hamaguchi, K., Kashiwabara, T.: Ordered Quantum Branching Programs Are More Powerful than Ordered Probabilistic Branching Programs under a Bounded-Width Restriction. In: Du, D.-Z., Eades, P., Sharma, A.K., Lin, X., Estivill-Castro, V. (eds.) COCOON 2000. LNCS, vol. 1858, pp. 467–476. Springer, Heidelberg (2000)
13. Razborov, A.A., Szemeredi, E., Wigderson, A.: Constructing small sets that are uniform in arithmetic progressions. Combinatorics, Probability & Computing **2**, 513–518 (1993)
14. Sauerhoff, M., Sieling, D.: Quantum branching programs and space-bounded nonuniform quantum complexity. Theoretical Computer Science 334(1–3), 177–225 (2005). http://arxiv.org/abs/quant-ph/0403164
15. Wegener, I.: Branching Programs and Binary Decision Diagrams. SIAM Monographs on Discrete Mathematics and Applications. SIAM Press (2000)
16. de Wolf, R.: Quantum Computing and Communication Complexity. Ph.D. thesis, University of Amsterdam (2001)
17. Yao, A.C.C.: Quantum circuit complexity. In: Proceedings of Thirty-fourth IEEE Symposium on Foundations of Computer Science, pp. 352–361. IEEE Computer Society, Palo Alto (1993)