Int J Software Informatics, Volume 7, Issue 4 (2013), pp.485–500 Tel: +86-10-62661040
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
©2013 by ISCAS. All rights reserved.
Email: ijsi@iscas.ac.cn, ijsi2007@gmail.com

Algorithms for Quantum Branching Programs Based on Fingerprinting

Farid Ablayev[1,2] and Alexander Vasiliev[1,2]

[1] (Institute for Informatics, Kazan, Russian Federation, Russian)
[2] (Kazan Federal University, Kazan, Russian Federation)

Abstract In the paper, we develop a method for constructing quantum algorithms for computing Boolean functions by quantum ordered read-once branching programs (quantum OBDDs). Our method is based on fingerprinting technique and representation of Boolean functions by their characteristic polynomials. We use circuit notation for branching programs for desired algorithms presentation. For several known functions our approach provides optimal QOBDDs. Namely we consider such functions as MOD_m, EQ_n, $Palindrome_n$, and $PERM_n$ (testing whether given Boolean matrix is the Permutation Matrix). We also propose a generalization of our method and apply it to the Boolean variant of the *Hidden Subgroup Problem*.
Key words: quantum informatics; quantum branching programs; QOBDD; fingerprinting; characteristic polynomials

Ablayev F, Vasiliev A. Algorithms for quantum branching programs based on fingerprinting. *Int J Software Informatics*, **Vol.7, No.4 (2013): 485–500. http://www.ijsi.org/1673-7288/7/i171.htm**

1 Introduction

During the last two decades different types of quantum computation models based on Turing Machines, automata, and circuits have been considered. For some of them different examples of functions were presented for which quantum models appear to be much more (exponentially) efficient than their classical counterparts.

In this paper we consider computation of Boolean functions by a quantum model of Branching programs. Branching programs (BP) model is one of the standard non-uniform models of computation and allow to describe sequential computations in an intuitively straightforward way in terms of commands "`if then else`" and "`go to`" only. Together with their simple "programming structure", this is the reason why BPs are interesting for developing algorithms for Boolean functions. BPs have proven useful in a variety of domains, such as hardware verification, model checking, and other CAD applications; see for example the book by Wegener[17]. BPs have another important attraction – this is a convenient model for defining different restricted variants (leveled, oblivious, constant width, read-once), which we consider in this paper.

This work is sponsored by Russian Foundation for Basic Research under Grants 09-01-97004 and 11-07-00465.
Corresponding author: Farid Ablayev, Email: fablayev@gmail.com
Received 2009-10-16; Revised 2011-08-08; Accepted 2011-08-09.

The complexity measures for BPs are closely related to machine and circuit models of computation:

– Sequences of functions that can be computed by BPs of polynomial size can also be computed within logarithmic space by the non-uniform variant of Turing machines and vice versa[15]. In terms of complexity classes, this means that the class of sequences of functions with polynomial-size BPs is equal to the class $LSPACE/poly$.

– Even oblivious branching programs of constant width – the non-uniform equivalent of finite-state automata – are surprisingly powerful. Indeed Barrington showed[7] that branching programs of width 5 are already as powerful as polynomial size circuits of logarithmic depth.

An oblivious read-once variant of BPs is well-known in theory and practice as *Ordered Binary Decision Diagrams* (OBDDs). We refer to Ref. [17] for more information.

In this paper we consider a quantum counterpart of OBDDs. The main reason for the investigation of restricted models of quantum computers was proposed by Ambainis and Freivalds in 1998[1]. Considering one-way quantum finite automata, they suggested that first quantum-mechanical computers would consist of a comparatively simple and fast quantum-mechanical part connected to a classical computer.

Two models of *quantum branching programs* were introduced by Ablayev, Gainutdinova, Karpinski[2] (*leveled programs*), and by Nakanishi, Hamaguchi, Kashiwabara[14] (*non-leveled programs*). Later it was shown by Sauerhoff[16] that these two models are polynomially equivalent.

For this model we develop the *fingerprinting* technique introduced in Ref. [6]. The basic ideas of this approach are due to Freivalds (e.g. see the book[12]). It was later successfully applied in the *quantum automata* setting by Ambainis and Freivalds in 1998[1] (later improved in Ref. [5]). Subsequently, the same technique was adapted for the quantum branching programs by Ablayev, Gainutdinova and Karpinski in 2001[2], and was later generalized in Ref. [6].

Fingerprinting technique for computing the Equality function in communication computation model in the quantum setting was explicitly described by Buhrman et al.[8] and later generalized by Yao[19].

In the paper we present a large enough family of Boolean functions for which our variant of fingerprinting technique provides effective (sometimes provably optimal) quantum OBDDs. Such a family of Boolean functions is described in terms of their polynomial presentations known as *characteristic polynomials*. Our definition of the characteristic polynomial differs from that of Ref. [10] and Ref. [4], though it uses similar ideas.

We display several known functions for which our method provides optimal quantum OBDDs. Namely, these functions are MOD_m, EQ_n, $Palindrome_n$, and $PERM_n$.

2 Preliminaries

At first we present the classical model of branching programs and then generalize it to the quantum setting. A good source of information on branching programs is Wegener's book[17]; for an introduction to quantum computation see Nielsen and Chuang[13].

A *branching program* is a finite directed acyclic graph which will be used to recognize some subset of $\{0,1\}^n$. Each node (except for a sink node) is labeled with an integer $1 \leq i \leq n$ and has two outgoing arrows labeled 0 and 1. This pair of edges corresponds to querying the i'th bit x_i of the input, and making a transition along one outgoing edge or the other depending on the value of x_i. There is a single source node, called the start node, and a subset Accept of the sink nodes corresponding to accepting nodes. An input x is *accepted* if and only if it induces a chain of transitions leading to a node in Accept, and the set of such inputs is the language accepted by the program.

A branching program is *oblivious* if the nodes can be partitioned into levels $V_1, \ldots, V_\ell$ and a level $V_{\ell+1}$ such that the nodes in $V_{\ell+1}$ are the sink nodes, nodes in each level V_j with $j \leq \ell$ have outgoing edges only to nodes in the next level V_{j+1}, and all nodes in a given level V_j query the same bit x_{i_j} of the input. Such a program is said to have *length* ℓ, and *width* k if each level has at most k nodes. The *size* of a branching program is defined as the number of its internal nodes.

It is known that any BP can be transformed into oblivious BP with only polynomial increase in size.

An oblivious branching program is called an *Ordered Binary Decision Diagram* (OBDD) if on each path from the start node to a sink each variable appears at most once.

Linear branching programs. The definition of a *linear branching program* is based on the oblivious model. This is a generalization of the definition of quantum branching program presented in Ref.[2]. Deterministic, probabilistic, and quantum oblivious branching programs are special cases of linear branching programs.

Let $\mathbf{V}^d$ be a d-dimensional vector space. We use $|\psi\rangle$ to denote column vectors from $\mathbf{V}^d$, and $\langle\psi_1|\psi_2\rangle$ denotes the inner product. We use the notation $|i\rangle$ for the vector, which has a 1 on the i-th position and 0 elsewhere.

Definition 1. A Linear Branching Program P on the variable set $X_n = \{x_1, \ldots, x_n\}$ over $\mathbf{V}^d$ is defined as

$$P = \langle T, |\psi_0\rangle, \mathrm{Accept}\rangle$$

where T is a sequence of l instructions: $T_j = \big(x_{i_j}, U_j(0), U_j(1)\big)$ determined by $x_{i_j} \in X_n$ tested at step j, and $U_j(0)$, $U_j(1)$ are $d \times d$ matrices.

Vectors $|\psi\rangle \in \mathbf{V}^d$ are called states (state vectors) of P, $|\psi_0\rangle \in \mathbf{V}^d$ is the initial state of P, and Accept $\subseteq \{1, \ldots, d\}$ is the accepting set.

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

1. A computation of P starts from the initial state $|\psi_0\rangle$;

2. The j'th instruction of P queries a variable x_{i_j}, and applies $U_j = U_j(\sigma_{i_j})$ to the current state $|\psi\rangle$ to obtain the state $|\psi'\rangle = U_j(\sigma_{i_j})\,|\psi\rangle$;

3. The final state is

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

Deterministic branching programs. A *deterministic branching program* (DPB) as a linear branching program over the vector space $\mathbb{R}^d$. A state $|\psi\rangle$ of such a program is a Boolean vector with exactly one 1. The matrices U_j correspond to permutations of order d, and so have exactly one 1 in each column.

A DBP accepts an input σ iff $|\psi(\sigma)\rangle = |i\rangle$ with $i \in$ Accept.

Quantum branching programs. We define a *quantum branching program* (QBP) is a linear branching program over the Hilbert space $\mathcal{H}^d$. The states for such a program are complex vectors with $\| |\psi\rangle \|_2 = 1$, and U_j are complex-valued unitary matrices.

After the l-th (last) step of quantum transformation, P measures its configuration $|\psi(\sigma)\rangle = (\alpha_1, \ldots, \alpha_d)^T$, and the input σ is accepted iff the result of the measurement is $|i\rangle$ with $i \in$ Accept.

The probability $Pr_{accept}(\sigma)$ of P accepting an input σ is defined by

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

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

$$Pr_{\text{accept}}(\sigma) = \langle \psi(\sigma) M^{\dagger}_{\text{accept}} \,|\, M_{\text{accept}} \psi(\sigma) \rangle = ||M_{\text{accept}} \,|\psi_\sigma\rangle\,||^2_2.$$

Computing Boolean functions. From the definition, it follows that a quantum branching program on the variable set X_n computes some Boolean function of n variables.

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.

Circuit representation. Quantum algorithms are usually given by using quantum circuit formalism[9, 18], 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.

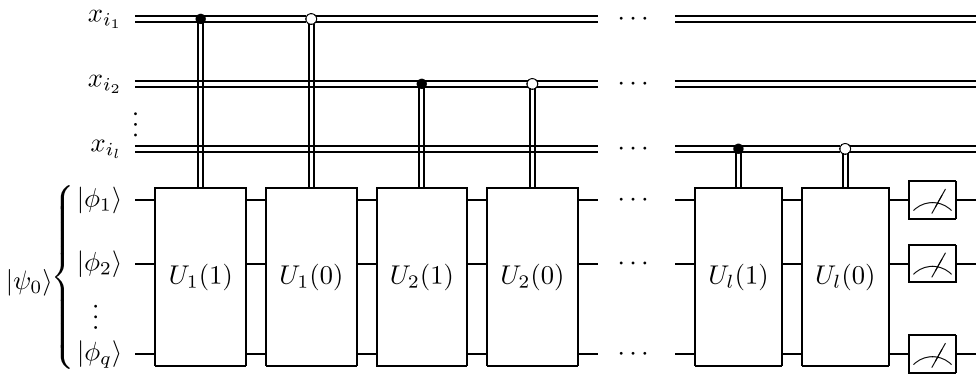

Figure 1. Circuit presentation of a quantum branching program.

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

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. There is one more commonly used complexity measure – the size of Q, which we define as $size(Q) = width(Q) \cdot length(Q)$.

Note that for a QBP Q in the circuit setting another important complexity measure explicitly comes out – a number of quantum bits, denoted by $qubits(Q)$, needed to encode the state of Q. From the definition, it follows that $qubits(Q) = \lceil \log width(Q) \rceil$.

Read-once branching programs.

Definition 2. We call a linear branching program P an OBDD or read-once LBP if each variable $x \in \{x_1, \ldots, x_n\}$ occurs in the sequence T of transformations of P at most once.

Note that the "obliviousness" is inherent for an LBP and therefore this definition is consistent with the usual notion of an OBDD defined in terms of graphs.

For the rest of the paper OBDD will denote deterministic OBDDs and QOBDD will be used for quantum ones. Additionally, we could have defined probabilistic branching programs and probabilistic OBDDs. However, in this paper we are mostly interested in establishing upper bounds for QOBDD model via fingerprinting technique and showing they are tight for several individual functions using the following general lower bound.

General Lower Bound. The following general lower bound on the width of QOBDDs was proven in [3].

Theorem 2.1. 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

$$\text{width}(Q) \geqslant \frac{\log \text{width}(P)}{2 \log\left(1 + \frac{1}{\epsilon}\right)}$$

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[17], we obtain a linear lower bound for these functions.

2.1 Quantum fingerprinting

Generally[12], *fingerprinting* – is a technique that allows to present objects (words over some finite alphabet) by their *fingerprints*, which are significantly smaller than the originals. It is used in randomized and quantum algorithms to test *equality* of some objects (binary strings) with one-sided error by simply comparing their fingerprints.

In this paper we develop a variant of the fingerprinting technique adapted for quantum branching programs. At the heart of the method is the representation of Boolean functions by polynomials of special type, which we call *characteristic*.

2.1.1 Characteristic polynomials for quantum fingerprinting

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

This definition is similar to the algebraic transformation of Boolean function from [10] and [4], though it is adapted for the fingerprinting technique.

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 $\mathbb{Z}_{2^n}$:

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

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

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

We use this presentation of Boolean functions for our fingerprinting technique which generalizes the algorithm for MOD_m function by Ambainis and Nahimovs[5].

2.1.2 Fingerprinting technique

For a Boolean function f we choose an error rate $\epsilon > 0$ and pick a characteristic polynomial g over the ring $\mathbb{Z}_m$. Then, for an arbitrary binary string $\sigma = \sigma_1 \dots \sigma_n$ we create its fingerprint $|h_\sigma\rangle$ composing $t = 2^{\lceil \log_2((2/\epsilon)\ln(2m))\rceil}$ single qubit fingerprints $\left|h^i_\sigma\right\rangle$:

$$\left|h^i_\sigma\right\rangle = \cos\frac{2\pi k_i g(\sigma)}{m}|0\rangle + \sin\frac{2\pi k_i g(\sigma)}{m}|1\rangle$$
$$|h_\sigma\rangle = \frac{1}{\sqrt{t}}\sum_{i=1}^{t}|i\rangle\left|h^i_\sigma\right\rangle$$

That is, the last qubit is rotated by t different angles about the $\hat{y}$ axis of the Bloch sphere.

The chosen parameters $k_i \in \{1, \dots, m-1\}$ for $i \in \{1, \dots, t\}$ are "good" following the notion of Ref. [1].

Definition 4. A set of parameters $K = \{k_1, \dots, k_t\}$ is called "good" for some integer $b \neq 0 \bmod m$ if

$$\frac{1}{t^2}\left(\sum_{i=1}^{t}\cos\frac{2\pi k_i b}{m}\right)^2 < \epsilon.$$

The left side of inequality is the squared amplitude of the basis state $|0\rangle^{\otimes \log t}|0\rangle$ if $b = g(\sigma)$ and the operator $H^{\otimes \log t} \otimes I$ has been applied to the fingerprint $|h_\sigma\rangle$. Informally, that kind of set guarantees, that the probability of error will be bounded by a constant below 1.

The following lemma proves the existence of a "good" set and generalizes the proof of the corresponding statement from [5] (for the proof see [6]).

Lemma 2. There is a set K with $|K| = t = 2^{\lceil \log_2((2/\epsilon)\ln(2m))\rceil}$ which is "good" for all integer $b \neq 0 \bmod m$.

We use this result for our fingerprinting technique choosing the set $K = \{k_1, \dots, k_t\}$ which is "good" for all $b = g(\sigma) \neq 0$. That is, it allows to distinguish those inputs whose image is 0 modulo m from the others.

3 Algorithms for QBPs Based on Fingerprinting

In this section we describe the class of Boolean functions that can be efficiently computed in the quantum OBDD model using the fingerprinting technique described in subsection 2.1.2.

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

Theorem 3.1. Let $\epsilon \in (0, 1)$. If g is a linear polynomial over $\mathbb{Z}_m$, i.e. $g = c_1x_1 + \dots c_nx_n + c_0$, then f can be computed with one-sided error ϵ by a quantum OBDD of width $O\left(\frac{\log m}{\epsilon}\right)$.

Proof. Here is the algorithm in the circuit notation:

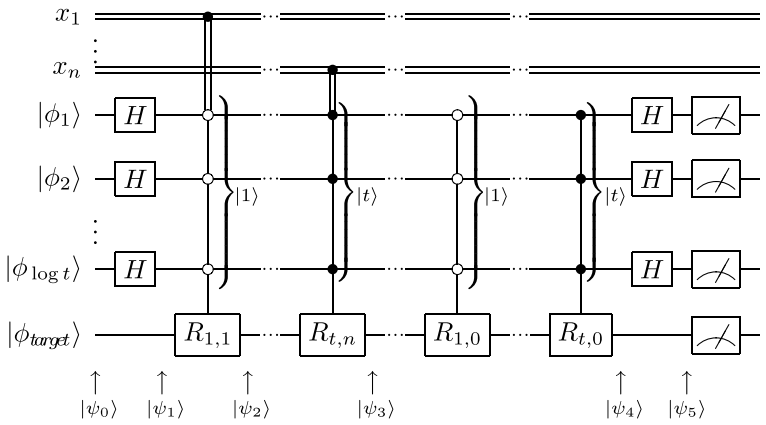

Figure 2. A Quantum Branching Program for computing Boolean function given by its linear characteristic polynomial.

Initially qubits $|\phi_1\rangle\otimes|\phi_2\rangle\otimes\cdots\otimes|\phi_{\log t}\rangle\otimes|\phi_{target}\rangle$ are in the state $|\psi_0\rangle = |0\rangle^{\otimes \log t}|0\rangle$. For $i \in \{1,\ldots,t\}$, $j \in \{0,\ldots,n\}$ we define rotations $R_{i,j}$ as

$$R_{i,j} = R_{\hat{y}}\left(\frac{4\pi k_i c_j}{m}\right),$$

where c_j are the coefficients of the linear polynomial for f and the set of parameters $K = \{k_1,\ldots,k_t\}$ is "good" according to the Definition 4 with $t = 2^{\lceil \log_2((2/\epsilon)\ln(2m))\rceil}$.

Let $\sigma = \sigma_1\ldots\sigma_n \in \{0,1\}^n$ be an input string.

The first layer of Hadamard operators transforms the state $|\psi_0\rangle$ into

$$|\psi_1\rangle = \frac{1}{\sqrt{t}}\sum_{i=1}^{t}|i\rangle|0\rangle.$$

Next, upon input symbol 0 identity transformation I is applied. But if the value of x_j is 1, then the state of the last qubit is transformed by the operator $R_{i,j}$, rotating it by the angle proportional to c_j. Moreover, the rotation is done in each of t subspaces with the corresponding amplitude $1/\sqrt{t}$. Such a parallelism is implemented by the controlled operators $C_i(R_{i,j})$, which transform the states $|i\rangle|\cdot\rangle$ into $|i\rangle R_{i,j}|\cdot\rangle$, and leave others unchanged. For instance, having read the input symbol $x_1 = 1$, the system would evolve into state

$$\begin{aligned}|\psi_2\rangle &= \tfrac{1}{\sqrt{t}}\sum_{i=1}^{t} C_i(R_{i,1})|i\rangle|0\rangle = \tfrac{1}{\sqrt{t}}\sum_{i=1}^{t}|i\rangle R_{i,1}|0\rangle \\ &= \tfrac{1}{\sqrt{t}}\sum_{i=1}^{t}|i\rangle\left(\cos\tfrac{2\pi k_i c_1}{m}|0\rangle + \sin\tfrac{2\pi k_i c_1}{m}|1\rangle\right)\end{aligned}.$$

Thus, after having read the input σ the amplitudes would "collect" the sum $\sum_{j=1}^{n} c_j\sigma_j$

$$|\psi_3\rangle = \tfrac{1}{\sqrt{t}}\sum_{i=1}^{t}|i\rangle\left(\cos\tfrac{2\pi k_i\sum_{j=1}^{n}c_j\sigma_j}{m}|0\rangle + \sin\tfrac{2\pi k_i\sum_{j=1}^{n}c_j\sigma_j}{m}|1\rangle\right).$$

At the next step we perform the rotations by the angle $\frac{4\pi k_i c_0}{m}$ about the $\hat{y}$ axis of the Bloch sphere for each $i \in \{1, \ldots, t\}$. Therefore, the state of the system would be

$$|\psi_4\rangle = \frac{1}{\sqrt{t}} \sum_{i=1}^{t} |i\rangle \left(\cos \frac{2\pi k_i g(\sigma)}{m} |0\rangle + \sin \frac{2\pi k_i g(\sigma)}{m} |1\rangle \right).$$

Applying $H^{\otimes \log t} \otimes I$ we obtain the state

$$|\psi_5\rangle = \left(\frac{1}{t} \sum_{i=1}^{t} \cos \frac{2\pi k_i g(\sigma)}{m} \right) |0\rangle^{\otimes \log t} |0\rangle + \\ + \gamma |0\rangle^{\otimes \log t} |1\rangle + \sum_{i=2}^{t} |i\rangle \left(\alpha_i |0\rangle + \beta_i |1\rangle \right),$$

where γ, α_i, and β_i are some unimportant amplitudes.

The input σ is accepted if the measurement outcome is $|0\rangle^{\otimes \log t} |0\rangle$. Clearly, the accepting probability is

$$Pr_{accept}(\sigma) = \frac{1}{t^2} \left(\sum_{i=1}^{t} \cos \frac{2\pi k_i g(\sigma)}{2^n} \right)^2.$$

If $f(\sigma) = 1$ then $g(\sigma) = 0$ and the program accepts σ with probability 1. Otherwise, the choice of the set $K = \{k_1, \ldots, k_t\}$ guarantees that

$$Pr_{accept}(\sigma) = \frac{1}{t^2} \left(\sum_{i=1}^{t} \cos \frac{2\pi k_i g(\sigma)}{2^n} \right)^2 < \epsilon.$$

Thus, f can be computed by a q-qubit quantum OBDD, where $q = \log t + 1$. The width of the program is $2^q = 2t = O\left(\frac{\log m}{\epsilon}\right)$. □

The following functions have the aforementioned linear polynomials and thus are effectively computed via the fingerprinting technique.

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 $\Omega(m)$[17]. 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.

$Palindrome_n(x_1, \ldots, x_n)$ This function tests the symmetry of the input, i.e. whether $x_1 x_2 \ldots x_{\lfloor n/2 \rfloor} = x_n x_{n-1} \ldots x_{\lceil n/2 \rceil + 1}$ 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} $(1 \leqslant i, j \leqslant n)$. 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[17]. The width of the best known probabilistic OBDD, computing this function with one-sided error, is $O(n^4 \log n)$[17]. 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 table provides the comparison of the width of quantum and deterministic OBDDs for the aforementioned functions.

Table 1 The comparison of the width of quantum and deterministic OBDDs for several known functions

	OBDD	QOBDD
MOD_m	$\Omega(m)$	$O(\log m)$
MOD'_m	$\Omega(m)$	$O(\log m)$
EQ_n	$2^{\Omega(n)}$	$O(n)$
$Palindrome_n$	$2^{\Omega(n)}$	$O(n)$
$PERM_n$	$\Omega(2^n n^{-5/2})$	$O(n \log n)$

The following functions have linear polynomials as well, but we're 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

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

3.1 Functions without Linear Polynomials

It appears that some functions don't have a linear characteristic polynomial at all. For example, consider a disjunction of n variables $f = x_1 \vee x_2 \vee \ldots \vee x_n$. If f had some linear characteristic polynomial $g_f = c_0 + c_1 x_1 + \ldots + c_n x_n$ then the following would hold:

$$\begin{aligned}
&g_f(0,0,0,\ldots,0) = c_0 \neq 0 \bmod m \\
&g_f(1,0,0,\ldots,0) = c_1 + c_0 = 0 \Rightarrow c_1 = -c_0 \bmod m \\
&g_f(0,1,0,\ldots,0) = c_2 + c_0 = 0 \Rightarrow c_2 = -c_0 \bmod m \\
&g_f(1,1,0,\ldots,0) = c_1 + c_2 + c_0 = -c_0 - c_0 + c_0 = -c_0 = 0 \bmod m.
\end{aligned}$$

This contradiction proves the absence of a linear presentation for f. However, the negation of f has a linear characteristic polynomial (e.g. $g_{\neg f} = \sum_{i=1}^{n} x_i$ over $\mathbb{Z}_{n+1}$).

The example above illustrates a straightforward approach, when we try to analyze the system of equations and inequalities obtained by substituting Boolean inputs to the presumable linear polynomial. A more general approach is a matter of further research.

4 Generalized Approach

The fingerprinting technique described earlier allows us to test a single property of the input encoded by a characteristic polynomial. Using the same ideas we can test the conjunction of several conditions encoded by a group of characteristic polynomials which we call a *characteristic* of a function.

Definition 5. We call a set χ_f^m of polynomials over $\mathbb{Z}_m$ a *characteristic* of a Boolean function f if for all polynomials $g \in \chi_f^m$ and all $\sigma \in \{0,1\}^n$ it holds that $g(\sigma) = 0$ iff $\sigma \in f^{-1}(1)$.

From Lemma 1 it follows that for each Boolean function there is always a characteristic consisting of a single characteristic polynomial.

We say that a characteristic is *linear* if all of its polynomials are linear.

Now we can generalize the Fingerprinting technique from section 2.1.2.

Generalized Fingerprinting technique. For a Boolean function f we choose an error rate $\epsilon > 0$ and pick a characteristic $\chi_f^m = \{g_1, \ldots, g_l\}$. Then for arbitrary binary string $\sigma = \sigma_1 \ldots \sigma_n$ we create its fingerprint $|h_\sigma\rangle$ composing $t \cdot l$ ($t = 2^{\lceil \log_2((2/\epsilon)\ln(2m))\rceil} = O\left(\frac{\log m}{\epsilon}\right)$) single qubit fingerprints $\left|h_\sigma^{i,j}\right\rangle$ ($1 \leqslant i \leqslant t$, $1 \leqslant j \leqslant l$):

$$\left|h_\sigma^{i,j}\right\rangle = \cos\frac{\pi k_i g_j(\sigma)}{m}|0\rangle + \sin\frac{\pi k_i g_j(\sigma)}{m}|1\rangle$$
$$|h_\sigma\rangle = \frac{1}{\sqrt{t}}\sum_{i=1}^{t}|i\rangle\left|h_\sigma^{i,1}\right\rangle\left|h_\sigma^{i,2}\right\rangle\ldots\left|h_\sigma^{i,l}\right\rangle$$

Theorem 4.1. If χ_f^m is a linear characteristic then f can be computed by a quantum OBDD of width $O\left(\frac{2^{|\chi_f^m|}\log m}{\epsilon}\right)$.

Proof. The proof of this result somewhat generalizes the proof of Theorem 2 to the case of several target qubits. Here is the circuit:

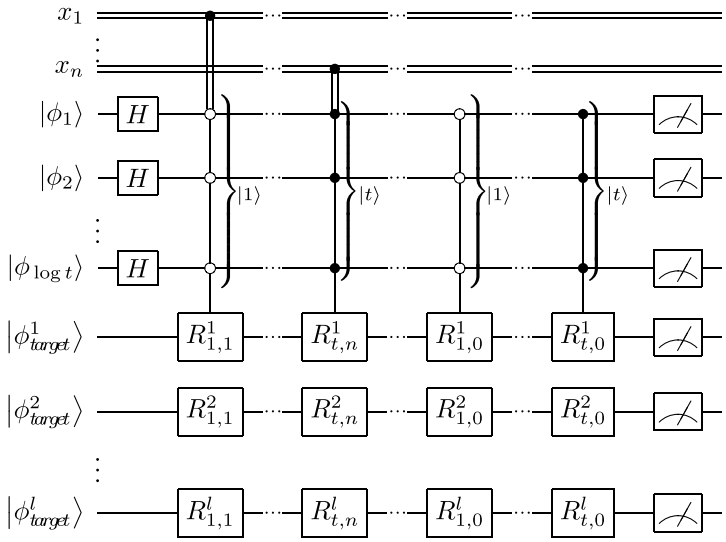

Figure 3. A Quantum OBDD for computing Boolean function given by its linear characteristic.

Thus, the proof follows the lines of the proof of Theorem 2. The differences are:

1. We exploit $l = |\chi_f^m|$ target qubits, one for each linear polynomial in χ_f^m.
2. We use single qubit rotations $R_{i,j}^k = R_{\hat{y}}\left(\frac{2\pi k_i c_{j,k}}{m}\right)$, where $c_{j,k}$ is the coefficient of x_j in k-th polynomial.
3. There is no final layer of Hadamard transformations.

Hence, upon the input $\sigma = \sigma_1 \ldots \sigma_n$ this algorithm creates the fingerprint $|h_\sigma\rangle$.

Afterwards, we measure $|h_\sigma\rangle$ in the standard computational basis and accept the input if the outcome of the last l qubits is the all-zero state. Therefore, the probability

of accepting σ is

$$Pr_{accept}(\sigma) = \frac{1}{t}\sum_{i=1}^{t} \cos^2 \frac{\pi k_i g_1(\sigma)}{m} \cdots \cos^2 \frac{\pi k_i g_l(\sigma)}{m}.$$

If $f(\sigma) = 1$ then all of $g_i(\sigma) = 0$ and we will always accept.

If $f(\sigma) = 0$ then there is at least one such j that $g_j(\sigma) \neq 0$ and the choice of the "good" set K guarantees that the probability of the erroneously accepting is bounded by

$$\begin{aligned} Pr_{accept}(\sigma) &= \tfrac{1}{t}\sum_{i=1}^{t} \cos^2 \tfrac{\pi k_i g_1(\sigma)}{m} \cdots \cos^2 \tfrac{\pi k_i g_l(\sigma)}{m} \\ &\leqslant \tfrac{1}{t}\sum_{i=1}^{t} \cos^2 \tfrac{\pi k_i g_j(\sigma)}{m} = \tfrac{1}{t}\sum_{i=1}^{t} \tfrac{1}{2}\left(1 + \cos \tfrac{2\pi k_i g_j(\sigma)}{m}\right) \\ &= \tfrac{1}{2} + \tfrac{1}{2t}\sum_{i=1}^{t} \cos \tfrac{2\pi k_i g_j(\sigma)}{m} \\ &\leqslant \tfrac{1}{2} + \tfrac{\sqrt{\epsilon}}{2}. \end{aligned}$$

The number of qubits used by this QBP is $q = \log t + l$, $l = |\chi_f^m|$. Therefore, the width of the program is $2^q = t \cdot 2^l = O\left(\frac{2^{|\chi_f^m|}\log m}{\epsilon}\right)$. □

Since m is usually exponential in n, this approach can be effectively used when the size of a characteristic is $O(\log\log m)$.

The generalized approach can be used to construct an effective quantum OBDD for the Boolean variant of the *Hidden Subgroup Problem*.

4.1 The upper bound for Hidden Subgroup Function

This problem was first defined and considered in Ref. [11], where the following Boolean variant of the *Hidden Subgroup Problem* was defined.

Definition 6. Let K be a normal subgroup of a finite group G. Let X be a finite set. For a sequence $\chi \in X^{|G|}$ let $\sigma = bin(\chi)$ be its representation in binary. If σ encodes no correct sequence $\chi = \chi_1 \ldots \chi_{|G|}$, then *Hidden Subgroup* function of σ is set to be zero, otherwise:

$$\mathrm{HSP}_{G,K}(\sigma) = \begin{cases} 1, \text{if} \quad \forall\, a, b \in G\ (aK = bK \iff \chi_a = \chi_b); \\ 0, \text{otherwise.} \end{cases}$$

Let f be the function encoded by the input sequence. We want to know if a function $f : G \to X$ "hides" the subgroup K in the group G. Our program receives G and K as *parameters*, and function f as an *input string* containing values of f it takes on G. The values are arranged in lexicographical order. See Definition 6.

We make two assumptions. First, we assume that the set X contains exactly $(G : K)$ elements. Indeed, having read the function f, encoded in the input sequence σ, we have X to be the set of all different values that f takes. Obviously, if $|X|$ is different from $(G : K)$, then $\mathrm{HSP}_{G,K}(\sigma) = 0$. The second assumption, is that we replace all values of f by numbers from 1 through $(G : K)$. Thus, $\mathrm{HSP}_{G,K}(x_1, \ldots, x_n)$ is a Boolean function of $n = |G|\lceil \log (G : K)\rceil$ variables. With these two assumptions the following theorem holds.

Theorem 4.2. Function $\mathrm{HSP}_{G,K}(x)$ can be computed with one-sided error by a quantum OBDD of width $O(n)$.

Proof. First we shall prove the following lemma.

Lemma 3. In order to correctly compute $\mathrm{HSP}_{G,K}(x)$ it is enough to perform the following calculations.

1. For every coset we check equalities for all input sequence values that have indices from this coset;

2. From every coset we choose a representative, and check if the sum of values of f on all the representatives equals to the following value

$$S = \sum_{i=1}^{G:K} i = \frac{(G:K)((G:K)+1)}{2}.$$

Proof. One direction is straightforward. The other direction is also not difficult. Suppose we have the two conditions of the lemma satisfied. Let aK and bK be two different cosets with elements $d \in aK$ and $c \in bK$, such that $\sigma_d = \sigma_c$. We fix $c \in bK$. There are two cases possible:

1. For all $d \in aK (\sigma_d = \sigma_c)$;

2. There exists $d' \in aK (\sigma_d \neq \sigma_c)$.

Apparently in the first case we indeed could choose any of the elements of a coset to check inequalities. In the second case the first condition of the lemma would fail. The reasoning for bK is analogous.

When the values of f are different on different cosets, obviously, the sum of these values is the sum of numbers from 1 through $G:K$. Therefore, $\mathrm{HSP}_{G,K}(\sigma) = 1$ iff both conditions of the lemma are satisfied. □

According to the previous lemma, $\mathrm{HSP}_{G,K}(x)$ has a characteristic consisting of two polynomials over $\mathbb{Z}_{2^n}$, checking conditions of the lemma. We shall construct them explicitly to show they are linear.

We shall adopt another indexation of χ when convenient: $\chi_{a,q}$ is a value of f on the q-th element of the coset aK.

Therefore, for a binary input symbol x_j we define

- $a = a(j)$ for the number of the corresponding coset;

- $q = q(j)$ for the number of the corresponding element of the coset a;

- $r = r(j)$ for the number of bit in the binary representation of $\chi_{a,q}$

and start indexation from 0. Thus $a \in \{0, \ldots, (G:K)-1\}, q \in \{0, \ldots, |aK|-1\}$.

In this notation the polynomials are:

1. $g_1(x) = \sum_a \sum_q 2^{(|K|a+q)\lceil \log G:K \rceil}(\chi_{a,q} - \chi_{a,q-1 \bmod |K|})$. Thus, $g_1(x) = 0$ iff for every coset a function f maps all the elements of a onto the same element of X.

2. $g_2(x) = \left(\sum_{j=1}^{(G:K)} \chi_{i_j}\right) - S$, where χ_{i_j} is the representative chosen from the j-th coset. Therefore, $g_2(x)$ checks whether the images of elements from different cosets are distinct.

By the generalized fingerprinting technique we can construct quantum OBDD of width $O(n)$, computing $\mathrm{HSP}_{G,K}(x)$ with one-sided error. □

Note that our algorithm completely ignores the fundamental difference between the Abelian and non-Abelian cases of the original Hidden Subgroup Problem. The reason for this is in the type of Boolean variant we consider here. The Boolean function $\mathrm{HSP}_{G,K}(x)$ doesn't compute the hidden subgroup, it just tests whether f hides the subgroup or not. This test can be efficiently performed because it can be reduced to the equality checks. Thus, the aforementioned difference between the Abelian and non-Abelian cases doesn't play any role here.

References

[1] Ambainis A, Freivalds R. 1-way quantum finite automata: strengths, weaknesses and generalizations. Proc. of the 39th IEEE Conference on Foundation of Computer Science. FOCS '98. Washington, DC, USA. IEEE Computer Society. 1998. 332–342.

[2] Ablayev F, Gainutdinova A, Karpinski M. On computational power of quantum branching programs. FCT. 2001. 59–70.

[3] 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, 2005, 203(12): 145–162.

[4] Agrawal V, Lee D, Wozniakowski H. Numerical computation of characteristic polynomials of boolean functions and its applications. Numerical Algorithms, 1998, 17: 261–278.

[5] Ambainis A, Nahimovs N. Improved constructions of quantum automata. In Kawano Y, Mosca M, eds. Theory of Quantum Computation, Communication, and Cryptography, LNCS 5106. Springer Berlin / Heidelberg. 2008. 47–56.

[6] Ablayev F, Vasiliev A. On the computation of boolean functions by quantum branching programs via fingerprinting. Electronic Colloquium on Computational Complexity (ECCC), 2008, 15(059).

[7] Barrington DA. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. Proc. of the eighteenth annual ACM symposium on Theory of computing. Berkeley, California, United States. Annual ACM Symposium on Theory of Computing. ACM Press. 1985. 1–5.

[8] Buhrman H, Cleve R, Watrous J, de Wolf R. Quantum fingerprinting. Phys. Rev. Lett., Sep 2001, 87(16):167902.

[9] Deutsch D. Quantum computational networks. Royal Society of London Proceedings Series A, Sep. 1989, 425: 73–90.

[10] Jain J, Abraham JA, Bitner J, Fussell DS. Probabilistic verification of boolean functions. Formal Methods in System Design, 1992, 1: 61–115.

[11] Khasianov A. Complexity bounds on some fundamental computational problems for quantum branching programs [PhD thesis], University of Bonn, 2005.

[12] Motwani R, Raghavan P. Randomized Algorithms. Cambridge University Press. 1995.

[13] Nielsen MA, Chuang IL. Quantum Computation and Quantum Information. Cambridge University Press, October 2000.

[14] 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 DZ, Eades P, Estivill-Castro V, Lin XM, Sharma A, eds. Computing and Combinatorics, LNCS 1858. Springer Berlin, Heidelberg. 2000. 467–476.

[15] Pudlak P, Zak S. Space complexity of computations [Technical Report]. Mathematical Institute of CSAV. Prague. 1983.

[16] Sauerhoff M, Sieling D. Quantum branching programs and space-bounded nonuniform quantum complexity. Theoretical Computer Science, 2005, 334(1–3): 177–225.

[17] Wegener I. Branching programs and binary decision diagrams. SIAM Monographs on Discrete Mathematics and Applications. SIAM Press. 2000.

[18] Yao ACC. Quantum circuit complexity. Proc. of Thirty-fourth IEEE Symposium on Foundations of Computer Science. Palo Alto, California, USA. IEEE Computer Society. 1993. 352–361.

[19] Yao ACC. On the power of quantum fingerprinting. Proc. of the thirty-fifth annual ACM symposium on Theory of computing, STOC '03. New York, ACM. 2003. 77–81.