Quantum algorithms for testing and learning Boolean functions

Quantum algorithms for testing and learning Boolean functions

DOMINIK FLOESS, ERIKA ANDERSSON and MARK HILLERY

Mathematical Structures in Computer Science / Volume 23 / Special Issue 02 / April 2013, pp 386 - 398
DOI: 10.1017/S0960129512000151, Published online: 28 February 2013

Math. Struct. in Comp. Science (2013), *vol.* 23, *pp.* 386–398.
doi:10.1017/S0960129512000151

Quantum algorithms for testing and learning Boolean functions[†]

DOMINIK FLOESS[†], ERIKA ANDERSSON[†]

and MARK HILLERY[‡]

[†]*SUPA, School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh EH14 4AS, United Kingdom*
Email: d.floess@pi4.uni-stuttgart.de; E.Andersson@hw.ac.uk
[‡]*Department of Physics, Hunter College of CUNY, Park Avenue, New York, NY 10061, U.S.A.*
Email: mhillery@hunter.cuny.edu

Received 2 November 2010; revised 15 August 2011

We discuss quantum algorithms based on the Bernstein–Vazirani algorithm for finding which input variables a Boolean function depends on. There are 2^n possible linear Boolean functions of n input variables; given a linear Boolean function, the Bernstein–Vazirani quantum algorithm can deterministically identify which one of these Boolean functions we are given using just one single function query. We show how the same quantum algorithm can also be used to learn which input variables any other type of Boolean function depends on. The success probability of learning that the function depends on a particular input variable depends on the form of the Boolean function that is tested, but does not depend on the total number of input variables. We also outline a procedure based on another quantum algorithm, the Grover search, to amplify further the success probability. Finally, we discuss quantum algorithms for learning the exact form of certain quadratic and cubic Boolean functions.

1. Introduction

In the oracle identification problem, we are given an oracle from a set of possible Boolean oracles, and our task is to determine which one we have (Ambainis 2002; Iwama *et al.* 2003). The complexity of the problem is measured by the number of times we must query the oracle to identify it. The time it takes to run the algorithm is determined by, and is in general proportional to, the number of oracle queries. Both the Bernstein–Vazirani (Bernstein and Vazirani 1993; Cleve *et al.* 1998) and Grover quantum algorithms (Grover 1997; Brassard *et al.* 1998) solve this type of problem. The Bernstein–Vazirani algorithm identifies linear Boolean functions with a single function query, and Grover's search algorithm finds marked elements in a database with N elements using $\mathcal{O}(\sqrt{N})$ queries.

In this paper, we will discuss quantum algorithms for testing and learning about Boolean functions. Consider the following task. We are given a black box that evaluates

[†] This work was partially supported by the National Science Foundation under grant PHY-0903660 and by EPSRC grant EP/G009821/1.

a Boolean function $f(x_1, x_2, \ldots x_n)$ that maps $\{0,1\}^n$ to $\{0,1\}$. The function depends on the values of at most m of the variables and is independent of the other $n - m$. Such a Boolean function is called a junta, and, if it depends on only one of the variables, it is called a dictatorship. Our first task is to find which of the variables the function depends on. We shall show how a variant of the Bernstein–Vazirani algorithm can solve this problem. Recently, Rötteler presented a quantum algorithm for identifying quadratic Boolean functions (Rötteler 2009). Atici and Serviedo discuss a quantum algorithm for identifying k-juntas, which is essentially based on the Bernstein–Vazirani oracle (Atici and Serviedo 2007). The quantum algorithm we outline is simpler; moreover, we also present a method based on Grover's quantum search algorithm to increase further the success probability. Following this, we will show how variants of the Bernstein–Vazirani quantum algorithm can be used to learn the form of quadratic and cubic Boolean functions where each input variable occurs only once.

Organisation of the paper

In Section 2, we review the Bernstein–Vazirani algorithm. In Section 3, we show that this quantum algorithm can also be used for the more general task of finding variables that other types of Boolean functions depend on. In Section 4, we show how a method based on the Grover search can be used to improve the success probability of finding variables that the Boolean function depends on. In Section 5, we treat the case of higher-order Boolean functions. Finally, we present our conclusions in Section 6.

2. The Bernstein–Vazirani algorithm

The Bernstein–Vazirani algorithm is a one-shot quantum algorithm (Bernstein and Vazirani 1993; Cleve *et al.* 1998) solving the following problem. We are given a black box that evaluates a linear Boolean function, given by

$$f(x) = y \cdot x = \sum_{j=1}^{n} y_j x_j, \tag{1}$$

where the addition is modulo 2 and y is a fixed, but unknown, n-bit string. We want to find y. The Bernstein–Vazirani algorithm does this with one evaluation of the function. It does so by mapping the functions to vectors in an N-dimensional Hilbert space $\mathscr{H} = \otimes^n \mathscr{H}_2$, where $N = 2^n$ and $\mathscr{H}_2$ is a two-dimensional Hilbert space. The computational basis vectors of $\mathscr{H}_2$ are $|0\rangle$ and $|1\rangle$, and the basis vectors of $\mathscr{H}$, corresponding to n-bit strings, are $|x\rangle = |x_1\rangle \otimes |x_2\rangle \ldots \otimes |x_n\rangle$. The function $y \cdot x$ is mapped to the vector $|v_y\rangle$, where

$$\langle x|v_y\rangle = \frac{1}{\sqrt{N}}(-1)^{y \cdot x}. \tag{2}$$

These vectors are orthonormal, that is, $\langle v_y|v_{y'}\rangle = \delta_{y,y'}$, and they constitute an orthonormal basis of $\mathscr{H}$, which is known as the parity basis (Bernstein and Vazirani 1993). This follows

from the identity

$$\sum_{x\in\{0,1\}^n} (-1)^{x\cdot y} = \delta_{y,0}. \tag{3}$$

Because the vectors are orthonormal, they are perfectly distinguishable, and so with one measurement we can perfectly determine which function the black box is evaluating.

This is actually accomplished by using a circuit consisting of Hadamard gates and an f-controlled-NOT gate. The Hadamard gate is the unitary transform

$$\begin{aligned} |0\rangle &\to \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle &\to \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned} \tag{4}$$

If we apply n Hadamard gates, one to each qubit in the state $|x\rangle$, we obtain

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{N}} \sum_{z\in\{0,1\}^n} (-1)^{x\cdot z}|z\rangle, \tag{5}$$

where, as before, we have set $N = 2^n$. The f-controlled-NOT gate, where f is a Boolean function, acts on $n+1$ qubits as follows:

$$U_f|x\rangle|z\rangle = |x\rangle|z + f(x)\rangle, \tag{6}$$

where $|x\rangle$ is an n-qubit computational basis sate, $|z\rangle$ is a one qubit state ($z = 0, 1$) and the addition is modulo 2. Now, the input state to the Bernstein–Vazirani circuit is the $(n+1)$-qubit state

$$|\Psi_{in}\rangle = \frac{1}{\sqrt{2}}|00\ldots0\rangle(|0\rangle - |1\rangle). \tag{7}$$

We first apply n Hadamard gates, one to each of the first n qubits, and then the f-controlled-NOT gate, giving us

$$|\Psi_{in}\rangle \to \frac{1}{\sqrt{2N}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle). \tag{8}$$

Next, we again apply n Hadamard gates to the first n qubits yielding

$$|\Psi_{out}\rangle = \frac{1}{N\sqrt{2}} \sum_{x\in\{0,1\}^n} \sum_{z\in\{0,1\}^n} (-1)^{f(x)+x\cdot z}|z\rangle(|0\rangle - |1\rangle). \tag{9}$$

Discarding the last qubit (it is not entangled with the others, so this has no effect) and expressing this result in terms of the vectors $|v_y\rangle$, we find the n-qubit output state

$$|\psi_{out}\rangle = \sum_{z\in\{0,1\}^n} \langle v_z|v_f\rangle|z\rangle, \tag{10}$$

where we have defined the vector v_f to have the components

$$\langle x|v_f\rangle = (1/\sqrt{N})(-1)^{f(x)}. \tag{11}$$

Now, if we know that $f(x)$ is of the form $f(x) = y \cdot x$, we just get the vector $|y\rangle$ as our output, and when we measure $|\psi_{out}\rangle$ in the computational basis, we find the n-bit

string y. Therefore, we find out what the function is with only one application of the f-controlled-NOT gate. Classically, we would need to evaluate the function n times to find y.

3. Testing which variables a general Boolean function depends on

If $f(x)$ is a general Boolean function, then when we measure $|\psi_{out}\rangle$ in the computational basis, we will obtain the label of one of the basis vectors v_y, with which v_f has a non-zero overlap. The key to using this to solve the problem stated in the Introduction is the following Theorem.

Theorem 3.1. If $f(x_1, x_2, \ldots x_n)$ is independent of the variable x_j and $y \in \{0,1\}^n$ has the property that $y_j = 1$, then $\langle v_y | v_f \rangle = 0$.

Proof. In order to prove the theorem, we start by noting

$$\begin{aligned} \langle v_y | v_f \rangle &= \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)+x\cdot y} \\ &= \frac{1}{\sqrt{N}} \sum_{x_1=0}^{1} \cdots \sum_{x_n=0}^{1} (-1)^{f(x)+x\cdot y}. \end{aligned} \tag{12}$$

Now, looking at the x_j sum, we have

$$\sum_{x_j=0}^{1} (-1)^{f(x)+x\cdot y} = (-1)^{f(x)} \prod_{k=1, k\neq j}^{n} (-1)^{x_k y_k} \sum_{x_j=0}^{1} (-1)^{x_j} = 0. \tag{13}$$

This means that if f does not depend on x_j and y is such that $y_j = 1$, then $\langle v_y | v_f \rangle = 0$, thus proving the theorem. □

Theorem 3.1 immediately and trivially implies the following theorem.

Theorem 3.2. If we use the Bernstein–Vazirani circuit with a Boolean function that is a junta and find an output vector $|y\rangle$ that has ones in a number of places, then the function depends on the variables corresponding to those places. If the function does not depend on a particular input variable, then the n-qubit state $|\psi_{out}\rangle$ will always have a 0 in that position.

Also, when used to identify which input variables a function depends on, the Bernstein–Vazirani algorithm requires only one application of the f-controlled-NOT gate defined in equation (6). It is important to note that the probability of successfully finding the variables the function depends on is *independent* of the total number n of input variables. This follows immediately, since if we add more input variables that the function f does not depend on, then the output y will always have a 0 in the corresponding positions. The probabilities of obtaining a 1 in the other positions, that is, the probability of identifying the variables the function does depend on, do not change. In general, the success probability for the quantum algorithm depends only on the form of the Boolean function that is being tested, that is, it depends on the number of significant variables,

and the functional form of the Boolean function involving these significant variables. In the following section, we will investigate the success probability for some particular forms of Boolean functions.

3.1. *Boolean functions depending on only two input variables*

In order to illustrate the fact that the success probability does not depend on the total number of input variables, we will consider a simple example. Suppose we know that our function is given by $f(x_1, x_2, \ldots x_n) = x_j x_k$, but we do not know j and k, that is, we know that the Boolean function is the product of two of the variables, but we do not know which two. Our task is to find out which two. The vector $|v_f\rangle$ corresponding to this function has a non-zero inner product with only four of the basis vectors $|v_y\rangle$. We must have $y_l = 0$ for $l \neq j,k$, which leaves four possibilities, which we shall denote by $|y_{00}\rangle$, corresponding to $y_j = y_k = 0$, $|y_{01}\rangle$, corresponding to $y_j = 0$ and $y_k = 1$, and so on. We find that the output of the Bernstein–Vazirani circuit in this case is

$$|\psi_{out}\rangle = \frac{1}{2}(|y_{00}\rangle + |y_{01}\rangle + |y_{10}\rangle - |y_{11}\rangle). \tag{14}$$

If we measure in the computational basis, we will obtain one of these basis vectors. If we obtain $|y_{00}\rangle$, we learn nothing, and the procedure has failed. This happens with a probability of 1/4. If we obtain either $|y_{01}\rangle$ or $|y_{10}\rangle$, we learn one of the variables, and if we obtain $|y_{11}\rangle$, we obtain both. All of these outcomes have a probability of 1/4, so we learn at least one of the variables on which the function depends with a probability of 3/4. This probability is independent of how many input variables n there are in total. Classically, a possible procedure would be to set all of the variables equal to 1 initially, which would set the value of the function equal to 1. We would then change the value of the variables, one at a time, to see which ones cause the value of the function to change. In order to learn which variables the function depends on, we would have to evaluate the function $\mathcal{O}(n)$ times. If n is large, the quantum procedure, though probabilistic, is more efficient. As far as we are aware, there is no classical procedure for which, with a constant number of function applications, the success probability does not decrease when the number of input variables increases. Neither are we aware of any classical procedure for which, in order to achieve a certain threshold success probability, the number of necessary function applications would not increase as a function of the number of input variables.

We will now consider a somewhat more general example. We will still assume that our function only depends on two out of the n variables, x_j and x_k say, but we will not assume the specific form of the function. We can express $f(x_1, x_2, \ldots x_n)$ as

$$f(x_1, x_2, \ldots x_n) = g(x_j, x_k), \tag{15}$$

where $g(x_j, x_k)$ is some Boolean function of two variables. Now, assuming that $y_l = 0$ for $l \neq j,k$, we have

$$\langle v_f | v_y \rangle = \frac{1}{4} \sum_{x_j, x_k = 0}^{1} (-1)^{g(x_j, x_k) + y_j x_j + y_k x_k}. \tag{16}$$

The right-hand side of the equation can only be 0, 1 or $\pm 1/2$, and it will only be 0 or 1 if $f(x_1, x_2, \ldots x_n)$ is one of the basis functions. Therefore, $|\psi_{out}\rangle$ is either one of the vectors $|y_{l_1 l_2}\rangle$, or of the form

$$|\psi_{out}\rangle = \frac{1}{2}(\pm|y_{00}\rangle \pm |y_{01}\rangle \pm |y_{10}\rangle \pm |y_{11}\rangle). \tag{17}$$

If $f(x_1, x_2, \ldots x_n)$ is one of the basis functions, this corresponds to the situation in the original version of the Bernstein–Vazirani algorithm, and we succeed after one trial. However, we do not know this, and several trials in which we get the same answer will be necessary to confirm that we have one of the basis functions. If $f(x_1, x_2, \ldots x_n)$ is not one of the basis functions, we will fail with a probability of 1/4, that is, we will get no information about which variables the function depends on. This happens if the measurement yields $|y_{00}\rangle$, corresponding to $y = 0$. In the remaining 3/4 of cases, we will learn at least one of the variables the function depends on. Therefore, after several trials, we will, with high probability, know x_j and x_k.

3.2. *Boolean functions depending on more than two input variables: an example*

We will now consider what the success probability is for a case where the function depends on more than two variables. We already know that the quantum algorithm will always find the variables a function depends on, but that the success probability for this will vary with the form of the Boolean function. Let us consider the case

$$f(x_1, x_2, \ldots x_n) = \prod_{j=1}^{m} x_j. \tag{18}$$

The probability to identify which variables this function depends on would also be the same for other Boolean functions that are a product of any m out of the n variables. For vectors $|v_y\rangle$ such that $y_j = 0$ for $j > m$, we have

$$\langle v_f | v_y \rangle = \frac{1}{2^m} \sum_{x_1=0}^{1} \cdots \sum_{x_m=0}^{1} (-1)^{h(x_1, \ldots x_m; y)}, \tag{19}$$

where

$$h(x_1, \ldots x_m; y) = \prod_{j=1}^{m} x_j + \sum_{j=1}^{m} x_j y_j. \tag{20}$$

Now, if the product $x_1 x_2 \ldots x_m$ were absent from the exponent in equation (19), and if at least one of the $y_j \neq 0$, then the sum would be zero. The product changes the sign of only one of the terms, so we have

$$\langle v_f | v_y \rangle = \pm \frac{1}{2^{m-1}}. \tag{21}$$

If $y_j = 0$ for $j = 1, \ldots n$ (we shall denote the vector corresponding to this y by $|v_0\rangle$), then without the product in the exponent, all of the terms in the sum in equation (19) would be 1. The presence of the product again changes only one term, so

$$\langle v_f | v_0 \rangle = 1 - \frac{1}{2^{m-1}}. \tag{22}$$

Note that since the failure probability is just $|\langle v_f|v_0\rangle|^2$, this implies that the failure probability grows with m. This is the 'worst case scenario'; this type of Boolean function belongs to the class of functions for which the Bernstein–Vazirani algorithm has the least probability of succeeding in finding the variables it depends on, since a phase factor is added only to a single term. Nevertheless, the success probability is still independent of the total number n of input variables.

4. Amplification of the success probability

The desirable outcomes of the measurement of the output state $|\psi_{out}\rangle$ are those with as many 1's as possible, since a '1' in position i indicates that the Boolean function depends on input variable x_i. To increase further the success probability of the quantum algorithm, it is possible to amplify components of $|\psi_{out}\rangle$ with a chosen number and above of 1's. This procedure is based on Grover's quantum search algorithm. Grover's algorithm uses $\mathcal{O}(\sqrt{N/M})$ queries for searching a database with N elements, where M of these are solutions to the search problem (Grover 1997; Brassard *et al.* 1998). Classically, $\mathcal{O}(N/M)$ database queries are needed.

Let us define the normalised states $|\alpha\rangle$ and $|\beta\rangle$ by

$$|\alpha\rangle = A{\sum_x}'' v_x|x\rangle; \quad A = \frac{1}{\sqrt{\sum}'' v_x^2} \tag{23}$$

$$|\beta\rangle = B{\sum_x}' v_x|x\rangle; \quad B = \frac{1}{\sqrt{\sum}' v_x^2}, \tag{24}$$

where the prime $'$ indicates a sum over all $x \in \{0,1\}^n$ that contain k or more 1's and $''$ indicates a sum over the remaining x. The state $|\psi_{out}\rangle$ in terms of $|\alpha\rangle$ and $|\beta\rangle$ is

$$|\psi_{out}\rangle = \frac{1}{A}|\alpha\rangle + \frac{1}{B}|\beta\rangle = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle, \tag{25}$$

where

$$\begin{aligned} \cos\frac{\theta}{2} &= 1/A = \sqrt{\sum}'' v_x^2 \\ \sin\frac{\theta}{2} &= 1/B = \sqrt{\sum}' v_x^2. \end{aligned} \tag{26}$$

Repeated application of the operator

$$G = H^{\otimes n} U_f H^{\otimes n}(2|0\rangle\langle 0| - \mathbf{1}) H^{\otimes n} U_f H^{\otimes n} O, \tag{27}$$

where the operator O produces phase factors -1 for components with k or more 1's, gives

$$G^l|\psi_{out}\rangle = \cos\left(\frac{2l+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2l+1}{2}\theta\right)|\beta\rangle \tag{28}$$

after l applications. The optimal number of Grover iterations is given by the integer closest to

$$R(\gamma) = \frac{\arccos[\sin(\theta/2)]}{\theta} = \frac{\arccos\sqrt{\gamma}}{2\arcsin\sqrt{\gamma}} \tag{29}$$

where $\gamma = \sum' v_x^2$. The leading term in the power series expansion of $R(\gamma)$ about $\gamma = 0$ is $\pi/(4\sqrt{\gamma})$. All higher order terms have a negative sign. Hence, we have

$$R < \frac{\pi}{4\sqrt{\gamma}}, \tag{30}$$

and if $\gamma \ll 1$, then

$$R \lesssim \frac{\pi}{4\sqrt{\gamma}}. \tag{31}$$

For this number of iterations, the final state contains the largest possible fraction of the component $|\beta\rangle$. If the form of the Boolean function is known (for example, it is known that it is of the form $x_i x_j$, but not what i, j are), then it is possible to calculate γ and the optimal number of Grover iterations for the chosen value of k. The smaller the value of k chosen, the larger γ is, and the fewer Grover iterations are needed. If the form of the function is not known, then, just as for the usual Grover search algorithm, it is possible to estimate the optimal number of Grover steps (Floess 2010). This will require more queries of the function to be tested. However, this does not necessarily mean that a significantly greater number of function queries is needed; this is the case for the example below.

4.1. *Amplification for a single term of order k*

As an example, we will consider the case where $f(x_1, x_2, \ldots x_n) = \prod_{j=1}^{m} x_j$, and suppose that we want to identify all variables this function depends on. As we pointed out earlier, the success probability would remain the same for any Boolean function that is a product of m input variables. From equation (20), we obtain $\gamma = 2^{-2m+2}$, and consequently the optimal number of Grover iterations needed to obtain a high probability of identifying all input variables the function depends on is given by the integer closest to $R = \pi\ 2^{m-3}$, which is $\mathcal{O}(2^m)$. Each iteration uses two queries of the Boolean function, so the total number of function queries is roughly $2R = \pi\ 2^{m-2}$, which is also $\mathcal{O}(2^m)$. Note that this number is independent of n, which is the total number of input variables.

If the Boolean function is a product of m of the input variables, but we do not know this, then we first need to estimate the optimal number of Grover iterations. It can be shown (Floess 2010) that for a product of m input variables, the circuit for estimating the optimal number of Grover steps requires $\mathcal{O}(2^m)$ function queries. In other words, if we are looking to amplify terms with m or more 1's, that is, to find all variables that the function depends on, then having to estimate the required number of Grover iterations does not change the order of how many function queries are needed in total. To restate this explicitly, for a function of the form $f(x_1, x_2, \ldots x_n) = \prod_{j=1}^{m} x_j$, the complexity of the quantum amplification step remains the same whether we know the value of m or not, and also does not depend on the total number of input variables. Also, the comparison with classical strategies remains the same as in the rest of this paper. That is, to reach some specified success probability, all quantum algorithms we consider require a number of function calls that does not depend on the total number of input variables, but only on the form of the function tested. As far as we are aware, classical algorithms always require a number of function calls that increases with the total number of input variables.

But how much better is it to use the Bernstein–Vazirani algorithm together with amplification, compared with using the same total number of function calls to repeat the Bernstein–Vazirani algorithm without amplification? We can compare the success probability of the quantum strategy that uses amplification to the case where we run the unmodified Bernstein–Vazirani algorithm roughly $2R = \pi\ 2^{m-2}$ times (the number of runs is given by the integer closest to this number). Without using amplification, the failure probability is $(1-2^{-m+1})^2$ in each round, so the probability of failing in all rounds, learning none of the variables the function depends on, is approximately

$$p_f = (1-2^{-m+1})^{\pi\ 2^{m-1}}. \tag{32}$$

The probability of obtaining at least one variable is therefore approximately $1-p_f$, which approaches $1-e^{-\pi} \approx 0.96$ when m becomes large. On the other hand, the probability of never learning one particular variable x_i that the function depends on in any of the $2R = \pi\ 2^{m-2}$ tries is equal to

$$p(\text{not learn } x_i) = \left(\sum_{v_y : y_i = 0} |\langle v_f | v_y \rangle|^2\right)^{\pi 2^{m-2}} = (1-2^{-m+1})^{\pi 2^{m-2}}. \tag{33}$$

This probability approaches $e^{-\pi/2} \approx 0.21$ when m becomes large. For $2R$ function queries, there is therefore an appreciable probability of not learning at least one variable the function depends on when using the Bernstein–Vazirani algorithm without amplification. The amplified procedure is very likely to obtain *all* variables that the function depends on with a similar number of function queries. Amplitude amplification for terms with m 1's has therefore improved the situation.

5. Quadratic and cubic functions

Let us now consider a particularly simple class of Boolean functions, those in which each variable appears in at most one term. Initially, we shall suppose that the function is a sum of linear and quadratic terms. Our task is to determine first, upon what variables the function depends, and second, which variables appear in quadratic terms and which appear in linear terms. We want to accomplish this with as few function queries as possible.

Quantum mechanically this can be done with three queries. We first find the variables in the quadratic terms. This can be done as follows. Let x be an n-bit input string and $\bar{x}$ be the string generated from x by flipping each bit, that is, if x_j is the value of the jth bit in x, then the value of the jth bit in $\bar{x}$ is $x_j + 1$. Now consider what happens if we take $f(x) + f(\bar{x})$. A variable that appears linearly is simply eliminated, because $x_j + \bar{x}_j = x_j + x_j + 1 = 1$. Quadratic terms, however, become linear terms

$$x_j x_k + (1 + x_j)(1 + x_k) = x_j + x_k + 1. \tag{34}$$

So if we apply the Berstein–Vazirani algorithm to the function $g(x) = f(x) + f(\bar{x})$, we will obtain all of the variables that appear in quadratic terms. Next, we can set these variables to zero and then apply the Bernstein–Vazirani algorithm to the original function, and

this will tell us the variables on which the function depends linearly. So, as stated, this procedure requires three function evaluations.

This can be accomplished easily in a quantum circuit. After the application of the f-Controlled-NOT gate in the standard Berstein–Vazirani algorithm, the state of the system is

$$\frac{1}{\sqrt{2N}}\sum_x(-1)^{f(x)}|x\rangle\otimes(|0\rangle-|1\rangle). \tag{35}$$

We will now apply NOT gates to the first n qubits, that is, all the input qubits except for the ancillary qubit. This has the effect of changing $|x\rangle$ to $|\bar{x}\rangle$. Next, we apply an f-Controlled-NOT gate to the system, and the resulting state is

$$\frac{1}{\sqrt{2N}}\sum_x(-1)^{f(x)+f(\bar{x})}|x\rangle\otimes(|0\rangle-|1\rangle). \tag{36}$$

Finally, we apply a Hadamard gate to each of the first n qubits, and, if $f(x)+f(\bar{x})$ is a linear function, we will obtain an output vector $|y\rangle$ with ones in the places corresponding to the variables on which $f(x)+f(\bar{x})$ depends.

Classically, it is possible to determine which variables the function depends on linearly and which it depends on quadratically with $2n$ function evaluations. First, we set all of the variables equal to one, and then set each to zero while keeping all of the other variables equal to one. If the function changes when we change a variable from 1 to 0, then the function depends on that variable, though at this point we do not know if the variable appears in a quadratic or linear term. We now set all of the variables equal to 0, and set each variable equal to 1 in turn. If the function changes when a particular variable is set equal to 1, then the function depends on that variable and it appears in a linear term. When this procedure is complete, we know all of the variables the function depends on and the ones on which it depends linearly. The variables in the first set but not in the second are the ones that appear in quadratic terms.

Now suppose we make the function more complicated by allowing cubic terms, but keep the restriction that each variable appears in only one term. We now want to determine on which variables a given function depends, and which of those variables appear linearly, which appear in quadratic terms and which appear in cubic terms. We will show that we can turn the cubic terms into linear terms and eliminate all of the other types of terms. We can then use the Berstein–Vazirani algorithm to find the variables in the cubic terms. Next we set these variables equal to zero, which yields a function with quadratic and linear terms. That case we already know how to handle.

Let us consider a single cubic term $f(x)=x_1x_2x_3$ in order to see what can happen. As a first step, we add this term to a term in which each x_j has been negated, that is, replaced by x_j+1. We find

$$\begin{aligned} g(x) &= f(x)+f(\bar{x}) = x_1x_2x_3+(1+x_1)(1+x_2)(1+x_3) \\ &= x_1x_2+x_1x_3+x_2x_3+x_1+x_2+x_3+1. \end{aligned} \tag{37}$$

Therefore, the cubic term has been reduced to a sum of quadratic and linear terms in the same variables as the ones making up the cubic term. It is tempting to try to take $g(x)+g(\bar{x})$ in order to reduce the order of the term still further, but since $g(x)+g(\bar{x})=0$,

that does not work. The reason for this is that in the quadratic part of $g(x)$, each x_i appears not once but twice. However, what we can do is set one of the variables, say x_1, equal to either 0 or 1, and then perform this procedure. We find that

$$\begin{aligned} h_1^{(0)}(x_2, x_3) &= g(0, x_2, x_3) + g(0, x_2 + 1, x_3 + 1) = x_2 + x_3 + 1 \\ h_1^{(1)}(x_2, x_3) &= g(1, x_2, x_3) + g(1, x_2 + 1, x_3 + 1) = x_2 + x_3 + 1. \end{aligned} \tag{38}$$

For a general Boolean function, $f(x)$, we define $g(x) = f(x) + f(\bar{x})$ and

$$\begin{aligned} h_j^{(m)}(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots x_n) = g(x_1, \ldots, x_{j-1}, m, x_{j+1}, \ldots x_n) \\ + g(x_1 + 1, \ldots, x_{j-1} + 1, m, x_{j+1} + 1, \ldots x_n + 1), \end{aligned} \tag{39}$$

where $m = 0, 1$. Returning now to our simple cubic function, what we have as a result of our procedure is a linear function that depends on two out of the three variables contained in the cubic term. If we set each of the variables equal to 0 or 1 in turn and then perform the above procedure, adding the function of the negated variables to the original function, we will obtain, with some redundancy, all of the variables on which the cubic term depends. Note that when negating all the other input variables, the variable that was fixed to 0 or 1 should stay fixed.

However, for this procedure to be useful, it has to eliminate the linear and quadratic terms, so that at the end we are only left with the variables appearing in the cubic terms. Linear terms are eliminated in the first step when we form $g(x) = f(x) + f(\bar{x})$, and a quadratic term, such as $x_1 x_2$, will have been turned into $x_1 + x_2 + 1$. In the next step, one of two things will happen. If neither x_1 nor x_2 are set equal to 0 or 1, then the contribution of $x_1 + x_2 + 1$ to $h_j^{(m)}$ will be zero. If x_1 or x_2 is one of the variables that is set equal to 0 or 1, then the contribution of $x_1 + x_2 + 1$ to $h_j^{(m)}$ is just the constant value 1. In both cases, the function $h_j^{(m)}$ does not depend on the values of x_1 or x_2. Therefore, the functions $h_j^{(m)}$ depend linearly only on the variables that appear in the cubic terms. Applying Berstein–Vazirani n times (for $j = 1, 2, \ldots n$) will yield all of the variables that appear in cubic terms. The total number of function evaluations to determine which variables appear in which terms will then be $n + 3$: that is, n to determine the variables in the cubic terms, and, after these have been set to 0, three more evaluations to determine those appearing in the quadratic and linear terms.

We now need to demonstrate how to carry out these steps quantum mechanically. We begin by showing how to create a superposition of two states, with one state corresponding to $x_1 = 0$ and the other to $x_1 = 1$. We have already shown how to create the state

$$\frac{1}{\sqrt{N}} \sum_x (-1)^{g(x)} |x\rangle = \frac{1}{\sqrt{N}} \sum_y [(-1)^{g(0,y)} |0, y\rangle + (-1)^{g(1,y)} |1, y\rangle], \tag{40}$$

where y is the $n - 1$ bit string $x_2 x_3 \ldots x_n$. We now append an ancilla qubit in the state $(|0\rangle - |1\rangle)/\sqrt{2}$ and apply the f-Controlled-NOT gate. The ancilla remains disentangled from the rest of the state, and we obtain

$$\frac{1}{\sqrt{N}} \sum_y [(-1)^{g(0,y)+f(0,y)} |0, y\rangle + (-1)^{g(1,y)+f(1,y)} |1, y\rangle]. \tag{41}$$

We now apply NOT gates to all but the first qubit in the state, which has the effect of transforming y to $\bar{y}$. We then again append an ancilla qubit and apply the f-Controlled-NOT gate. The result is

$$\frac{1}{\sqrt{N}}\sum_{y}[(-1)^{h_1^{(0)}(y)}|0,y\rangle + (-1)^{h_1^{(1)}(y)}|1,y\rangle]. \tag{42}$$

Now for functions of the type we are considering, which are at most cubic and with each variable appearing in only one term, the functions $h_j^{(0)}$ and $h_j^{(1)}$ are the same. This can be verified simply by seeing what happens to linear, quadratic and cubic terms in the progression from $f(x)$ to $h_j^{(m)}$, $m = 0, 1$. Noting this, the above state can be expressed as

$$\frac{1}{\sqrt{N}}(|0\rangle + |1\rangle)\sum_{y}(-1)^{h_1^{(0)}(y)}|y\rangle. \tag{43}$$

Applying a Hadamard gate to each of the last $n-1$ qubits will result in a state with ones in the places corresponding to the variables on which $h_1^{(0)}$ depends.

Most of this procedure can be repeated in the classical case. Once we have formed $h_j^{(0)}$, however, $n-1$ function calls are then needed to find the variables on which $h_j^{(0)}$ depends (the function is evaluated for the bit strings in which all of the bits but one are set equal to zero). Therefore, in the quantum case, $O(n)$ function evaluations are required, while in the classical case $O(n^2)$ are. Thus the quantum advantage resulting from the use of the Bernstein–Vazirani algorithm is useful for more than just linear functions.

6. Conclusions

We have shown that the Bernstein–Vazirani algorithm may be used for testing which input variables an unknown Boolean function depends on. This task is more general than distinguishing between linear Boolean functions, which is the task for which the Bernstein–Vazirani algorithm was originally devised. The success probability of finding variables a Boolean function depends on may be further enhanced by an amplification procedure based on Grover's search algorithm. The success probability for the quantum algorithm we have presented depends on the particular form of the Boolean function, but has the general property that it is independent of the total number of input variables. It shares this property with the algorithm presented in Atici and Serviedo (2007). Nevertheless, a full comparison of the success probabilities of the different quantum and classical algorithms remains to be made.

We have also outlined quantum algorithms for learning which input variables quadratic and cubic Boolean functions depend on for the case when each input variable occurs at most once in the form of the function. For quadratic functions, three queries are needed, and for cubic functions, the success probability scales linearly with the number of input variables. These algorithms are deterministic, that is, they are guaranteed to give perfect knowledge about the exact form of the Boolean function. It would also be interesting to investigate how the probabilistic quantum algorithms in Sections 3 and 4 could be combined with the probabilistic algorithms for learning cubic and quadratic functions in Section 5. Then, it should be possible to devise a probabilistic algorithm for learning

cubic and quadratic Boolean functions, where the number of function queries needed is independent of the number of input variables. Other variations of the Bernstein–Vazirani algorithm may also be tailored for investigating Boolean functions of particular forms, and this will be the subject of further investigations.

References

Ambainis, A. (2002) Quantum lower bounds by quantum arguments. *Journal of Computer and System Science* **64** 750–76.

Ambainis, A., Iwama, K., Kawachi, A., Masuda, H., Putra, R. H. and Yamashita, S. (2004) Quantum Identification of Boolean Oracles. In: Diekert, V. and Habib, M. (eds.) Proceedings of STACS 2004. *Springer-Verlag Lecture Notes in Computer Science* **2996** 105–116.

Atici, A. and Serviedo, R. A. (2007) Quantum Algorithms for Learning and Testing Juntas. *Quantum Information Processing* **6** 323–348.

Bernstein, E. and Vazirani, U. (1993) Quantum Complexity Theory. In: *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, ACM Press 11–20.

Brassard, G., Hoyer, P., Boyer, M. and Tapp, A. (1998) Tight bounds on quantum searching. *Fortschritte der Physik* **46** 493–505.

Cleve, R., Ekert, A., Macchiavello, C. and Mosca, M. (1998) Quantum Algorithms Revisited. *Proceedings of the Royal Society of London, Series A* **454** 339–354.

Grover, L. (1997) Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Physics Review Letters* **79** 325–328.

Floess, D. F. (2010) Quantum Mechanics and Boolean Functions, Master's thesis, Heriot-Watt University.

Iwama, K., Kawachi, A., Masuda, H., Putra, R. H. and Yamashita, S. (2003) Quantum Evaluation of Multi-Valued Boolean Functions. (Available at quant-ph/0304131.)

Rötteler, M. (2009) Quantum algorithms to solve the hidden shift problem for quadratics and for functions of large Gowers norm. In: Krlovic, R. and Niwinski, D. (eds.) Mathematical Foundations of Computer Science 2009, Proceedings. *Springer-Verlag Lecture Notes in Computer Science* **5734** 663–674.