Quantum algorithms for testing Boolean functions

Dominik F. Floess Erika Andersson
SUPA, School of Engineering and Physical Sciences
Heriot-Watt University, Edinburgh EH14 4AS, United Kingdom
dominikfloess@gmx.de E.Andersson@hw.ac.uk

Mark Hillery
Department of Physics, Hunter College of CUNY
695 Park Avenue, New York, NY 10061, USA
mhillery@hunter.cuny.edu

We discuss quantum algorithms, based on the Bernstein-Vazirani algorithm, for finding which variables a Boolean function depends on. There are 2^n possible linear Boolean functions of n 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. The same quantum algorithm can also be used to learn which input variables other types of Boolean functions depend on, with a success probability that 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 to futher amplify the success probability, based on another quantum algorithm, the Grover search.

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 [1]-[3]. The complexity of the problem is measured by the number of times we must query the oracle in order to identify it. Both the Bernstein-Vazirani [4, 5] and Grover quantum algorithms [6, 7] 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.

Consider the following task. We are given a black box that evaluates 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 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 [8]. Atici and Serviedo discuss a quantum algorithm for identifying k-juntas, essentially based on the Bernstein-Vazirani oracle [9]. The quantum algorithm we outline is simpler; moreover, we also present a method to further increase the success probability, based on Grover's quantum search algorithm.

The paper is arranged in the following way. 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 other types of Boolean functions depend on. In section 4, we show how a method based on the Grover search can be used to further increase the success probability of finding variables the Boolean function depends on. We finish with Conclusions.

S. B. Cooper, E. Kashefi, P. Panangaden (Eds.): Developments in Computational Models (DCM 2010)
EPTCS 26, 2010, pp. 101–108, doi:10.4204/EPTCS.26.9

2 The Bernstein-Vazirani algorithm

The Bernstein-Vazirani algorithm is a one-shot quantum algorithm [4, 5]. It solves the following problem. One has 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 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 $\mathscr{H}$ are labeled by n-bit strings $|x\rangle = |x_1\rangle \otimes |x_2\rangle \ldots \otimes |x_n\rangle$. The function $y \cdot x$ is mapped to the vector v_y, where

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

These vectors are orthonormal, i.e. $\langle v_y|v_{y'}\rangle = \delta_{y,y'}$, and they constitute an orthonormal basis of $\mathscr{H}$ known as the parity basis [4]. 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

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

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 in the following way

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

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 $\langle x|v_f\rangle = (1/\sqrt{N})(-1)^{f(x)}$. Now, if we know that $f(x)$ is of the form $f(x) = y\cdot x$, then 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 Learning 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 nonzero overlap. The key to using this to solve the problem stated in the Introduction is the following fact: 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 $\langle v_y|v_f\rangle = 0$. In order to prove this we start by noting

$$\begin{aligned} \langle v_y|v_f\rangle &= \frac{1}{\sqrt{N}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)+x\cdot y} \\ &= \frac{1}{\sqrt{N}} \sum_{x_1=0}^{1} \cdots \sum_{x_n=0}^{1} (-1)^{f(x)+x\cdot y}. \end{aligned} \tag{11}$$

Now look at the x_j sum,

$$\sum_{x_1=0}^{1} (-1)^{f(x)+x\cdot y} = (-1)^{f(x)} \prod_{k=1,k\neq j}^{n} (-1)^{x_k y_k} \sum_{x_j=0}^{1} (-1)^{x_j} = 0. \tag{12}$$

This proves our result. What it implies is that 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 does depend 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.

It is important to note that the success probability to find the variables the function depends on is *independent* of the total number n of input variables. 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.

3.1 Boolean functions depending on only two input variables

In order to illustrate this, let us consider a simple example. Suppose that 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, i.e. 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 nonzero 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 $y_k = 1$, etc. We find that the output of the Bernstein-Vazirani circuit in this case is

$$|\psi_{out}\rangle = \frac{1}{2}(|y_{00}\rangle + |y_{01}\rangle + |y_{10}\rangle - |y_{11}\rangle). \tag{13}$$

If we measure in the computational basis, then 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 $|y_{10}\rangle$, then 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 that 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 initially set all of the variables equal to 1, which would set the value of the function equal to 1. One then changes 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 on which variables the function depends, one would have to evaluate the function $\mathcal{O}(n)$ times. If n is large, the quantum procedure, though probabilistic, is more efficient.

Let us 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, but we will not assume the specific form of the function. We can express $f(x_1,x_2,\ldots x_n)$ as

$$f(x_1,x_2,\ldots x_n) = g(x_j,x_k), \tag{14}$$

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 that

$$\langle v_f|v_y\rangle = \frac{1}{4}\sum_{x_j,x_k=0}^{1}(-1)^{g(x_j,x_k)+y_j x_j + y_k x_k}. \tag{15}$$

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

$$|\psi_{out}\rangle = \frac{1}{2}(\pm|y_{00}\rangle \pm |y_{01}\rangle \pm |y_{10}\rangle \pm |y_{11}\rangle). \tag{16}$$

If $f(x_1,x_2,\ldots x_n)$ is one of the basis functions, 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. In all other cases, we will fail, that is get no information about which variables the function depends on, with a probability of $1/4$, so that 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

Now let us see what happens if the function depends on more than two variables. We 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

$$f(x_1, x_2, \ldots x_n) = \prod_{j=1}^{m} x_j. \tag{17}$$

The probability to identify which variables this function depends on would be the same also for other Boolean functions which 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

$$\langle v_f | v_y \rangle = \frac{1}{2^m} \sum_{x_1=0}^{1} \cdots \sum_{x_m=0}^{1} (-1)^{h(x_1,\ldots x_m;y)}, \tag{18}$$

where

$$h(x_1, \ldots x_m; y) = \prod_{j=1}^{m} x_j + \sum_{j=1}^{m} x_j y_j. \tag{19}$$

Now, if the product $x_1 x_2 \ldots x_m$ were absent from the exponent in Eq. (18), 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 that we have

$$\langle v_f | v_y \rangle = \pm \frac{1}{2^{m-1}}. \tag{20}$$

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 Eq. (18) would be 1. The presence of the product again changes only one term, so that

$$\langle v_f | v_0 \rangle = 1 - \frac{1}{2^{m-1}}. \tag{21}$$

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 least probability to succeed 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 further increase the success probability, 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 $\mathcal{O}(N/M)$ queries for searching a database with N elements, where M of these are solutions to the search problem [6, 7]. Classically, $\mathcal{O}(N/M)$ database queries are needed.

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

$$|\alpha\rangle = A \sum_{x}{}'' v_x |x\rangle; \quad A = \frac{1}{\sqrt{\sum'' v_x^2}} \tag{22}$$

$$|\beta\rangle = B \sum_{x}{}' v_x |x\rangle; \quad B = \frac{1}{\sqrt{\sum' v_x^2}}, \tag{23}$$

where the prime $'$ indicates a sum over all $x \in \{0,1\}^n$ which 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

$$|\psi_{out}\rangle = \frac{1}{A}|\alpha\rangle + \frac{1}{B}|\beta\rangle = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle, \tag{24}$$

where $\cos\theta = 1/A = \sqrt{\sum'' v_x^2}$ and $\sin\theta = 1/B = \sqrt{\sum' v_x^2}$. Repeated application of the operator

$$G = H^{\otimes n}U_f H^{\otimes n}(2|0\rangle\langle 0| - \mathbf{1})H^{\otimes n}U_f H^{\otimes n}O, \tag{25}$$

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

$$G^l|\psi_{out}\rangle = \cos\left(\frac{2l+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2l+1}{2}\theta\right)|\beta\rangle. \tag{26}$$

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

$$R(\gamma) = \frac{\arccos[\sin(\theta/2)]}{\theta} = \frac{\arccos\sqrt{\gamma}}{2\arcsin\sqrt{\gamma}}, \tag{27}$$

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

$$R < \frac{\pi}{4\sqrt{\gamma}}, \tag{28}$$

and if $\gamma \ll 1$, then

$$R \lesssim \frac{\pi}{4\sqrt{\gamma}}. \tag{29}$$

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 (e.g. that the Boolean function 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 k is 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 [10]. This will require more queries of the function to be tested. It does, however, 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, let us 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 also pointed out before, the success probability would remain the same for any Boolean function which 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 in order 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 that the total number of function queries is roughly $2R = \pi\, 2^{m-2}$, which is also $\mathcal{O}(2^m)$. We point out 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 [10] 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 variable 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.

We can compare the success probability of the amplification strategy 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). In each round, the failure probability is $(1-2^{-m+1})^2$, so that the probability to fail in all rounds, learning none of the variables the function depends on, is approximately $p_f = (1-2^{-m+1})^{\pi\ 2^{m-1}}$. The probability to obtain 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 to never learn one particular variable x_i that the function does depend on, in any of the $2R = \pi\ 2^{m-2}$ tries, is equal to

$$p(\text{not learn } x_i) = \left(\sum_{v_y : y_i = 0} |\langle v_f | v_y \rangle|^2 \right)^{\pi 2^{m-2}} = (1-2^{-m+1})^{\pi 2^{m-2}}. \tag{30}$$

This probability approaches $e^{-\pi/2} \approx 0.21$ when m becomes large. For $2R$ function queries, there is therefore an appreciable probability for 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 which 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 Conclusions

We have shown that the Bernstein-Vazirani algorithm may be used for testing which input variables an unknown Boolean function depends on. In a sense, 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 presented quantum algorithm 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 [9]. Nevertheless, a full comparison of the success probabilities of the different quantum and classical algorithms remains to be made. 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

[1] A. Ambainis, (2002): *Quantum lower bounds by quantum arguments*. *Journal of Computer and System Science* 64, pp. 750–767.

[2] A. Ambainis, K. Iwama, A. Kawachi, H. Masuda, R. H. Putra, and S. Yamashita (2004): *Quantum Identification of Boolean Oracles*. *Proceedings of STACS 2004, Lecture Notes in Computer Science* (Springer, Berlin), pp. 105–116.

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

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

[5] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca (1998): *Quantum Algorithms Revisited*. *Proc. R. Soc. London, Ser. A* 454, pp. 339–354 (1998).

[6] L. Grover (1997): *Quantum Mechanics Helps in Searching for a Needle in a Haystack*. *Phys. Rev. Lett.* 79, pp. 325–328.

[7] G. Brassard, P. Hoyer, M. Boyer and A. Tapp (1998): *Tight bounds on quantum searching*. *Fortsch. Phys. – Prog. Phys.* 46, pp. 493–505.

[8] M. Rötteler (2009): *Quantum algorithms to solve the hidden shift problem for quadratics and for functions of large Gowers norm* . In *Mathematical Foundations of Computer Science 2009, Proceedings, Lecture Notes in Computer Science* (Springer, Berlin), pp. 663–674.

[9] A. Atici and R. A. Serviedo (2007): *Quantum Algorithms for Learning and Testing Juntas*. *Quant. Inf. Proc.* 6, pp. 323–348.

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