On exact quantum query complexity

Ashley Montanaro*, Richard Jozsa and Graeme Mitchison

Centre for Quantum Information and Foundations, DAMTP, University of Cambridge, UK.

November 2, 2011

Abstract

We present several families of total boolean functions which have exact quantum query complexity which is a constant multiple (between 1/2 and 2/3) of their classical query complexity, and show that optimal quantum algorithms for these functions cannot be obtained by simply computing parities of pairs of bits. We also characterise the model of nonadaptive exact quantum query complexity in terms of coding theory and completely characterise the query complexity of symmetric boolean functions in this context. These results were originally inspired by numerically solving the semidefinite programs characterising quantum query complexity for small problem sizes. We include numerical results giving the optimal success probabilities achievable by quantum algorithms computing all boolean functions on up to 4 bits, and all symmetric boolean functions on up to 6 bits.

1 Introduction

Many important quantum algorithms operate in the query complexity model. In this model, the quantity of interest is the number of oracle queries to the input $x \in \{0,1\}^n$ required to compute some (possibly partial) function $f(x)$. Many functions f are now known to admit quantum speed-up in the case where the algorithm is allowed a constant probability of error, and the model of bounded-error quantum query complexity is now relatively well understood. Intriguingly, the model of exact quantum query complexity, where the quantum algorithm must succeed with certainty, seems to be more mysterious.

If f is allowed to be a partial function, it is known that there can be an exponential separation between exact quantum and exact classical query complexity [8], and even between exact quantum and bounded-error classical query complexity [4]. There are also examples of total functions where exact quantum algorithms allow a speed-up over classical algorithms. In particular, it is known that the parity of n bits can be computed exactly using only $\lceil n/2 \rceil$ quantum queries [5, 10], while any classical algorithm (exact or randomised) must make exactly n queries. However, this is the best quantum speed-up known for the exact computation of total boolean functions. Even worse, to the authors' knowledge this is essentially the *only* exact quantum speed-up known. Some years ago, van Melkebeek, Hayes and Kutin [12] gave a quantum algorithm that computes the majority function on n bits exactly using only $n + 1 - w(n)$ queries, where $w(n)$ is the number of 1s in the binary expansion of n, but the only quantum ingredient in this algorithm is computing the parity of 2 bits using 1 query. The same applies to quantum algorithms presented by Dubrovska

*am994@cam.ac.uk

and Mischenko-Slatenkova [9], and also algorithms by Vasilieva [19, 20]. In the other direction, it is known that the separation between quantum and classical exact query complexity can be at most cubic [15], whereas the best known relationship between bounded-error quantum and classical query complexity is a 6th power [3].

1.1 Our results

We show that the model of exact quantum query complexity is richer than it may have hitherto appeared. Our results can be divided into analytical and numerical parts. On the analytical side, the main results are as follows.

- We present several new families of boolean functions f for which the exact quantum query complexity of f is a constant multiple (between $1/2$ and $2/3$) of its classical query complexity, and we show that optimal quantum algorithms for these functions cannot be obtained by simply computing parities of pairs of bits. These separations are based on concatenating small separations obtained for functions on small numbers of bits; indeed, we give optimal exact quantum query algorithms for every boolean function on 3 bits.

- We give a simple and explicit quantum algorithm for determining whether a 4-bit string has Hamming weight 2, using only 2 queries. Again, this cannot be done merely by computing parities of pairs of input bits. More generally, we give an exact algorithm which distinguishes between inputs of Hamming weight $n/2$ and Hamming weight in the set $\{0, 1, n-1, n\}$, for all even n, using 2 queries. This generalises the well-known Deutsch-Jozsa problem [8] of distinguishing Hamming weight $n/2$ from Hamming weight in $\{0, n\}$.

- We characterise the model of *nonadaptive* exact quantum query complexity in terms of a coding-theoretic quantity. In this setting, all queries to the input must be made up front, in parallel. In contrast to the classical case, there exist non-trivial quantum algorithms in this model. Using our characterisation result, we completely determine the nonadaptive exact quantum query complexity of symmetric boolean functions.

These analytical results were inspired by numerical investigations in which we evaluated the best success probability achievable by quantum algorithms computing all boolean functions on up to 4 bits, and all symmetric boolean functions on up to 6 bits. This was achieved using the semidefinite programming approach of Barnum, Saks and Szegedy [2] to formulate semidefinite programs (SDPs) giving the precise optimal success probability for quantum algorithms using up to k queries, for all $k < n$. We then solved these SDPs numerically using the CVX package [11] for Matlab; the results are given in Section 6 and Appendix A. Given an SDP solution, one can then write down an *explicit* quantum query algorithm with the same parameters; we discuss how this can be done in Section 4. Our analytical results were based on inspecting these algorithms.

Some further highlights from the numerical results are as follows.

- We conjecture that the exact quantum query complexity of the two problems of determining whether an n-bit string has Hamming weight exactly k is precisely $\max\{k, n-k\}$. We also conjecture that determining whether an n-bit string has Hamming weight at least k has exact quantum query complexity $\max\{k, n-k+1\}$ for $k \geq 1$. Both conjectures hold numerically for all $n \leq 6$.

- In particular, we present strong numerical evidence that the algorithm of van Melkebeek, Hayes and Kutin [12] is not always optimal, by showing that there exists an exact quantum algorithm for computing the majority function on 5 bits using only 3 queries, while their algorithm would require 4 queries.

- We show numerically that all boolean functions on up to 4 bits, with the exception of functions equivalent to the AND function, have an exact quantum query algorithm using at most 3 queries.

Note that Reichardt and Špalek have previously solved related SDPs (known as the adversary and generalised adversary bounds) for all boolean functions on up to 4 bits [18]. These SDPs give lower bounds on bounded-error quantum query complexity but do not characterise it exactly, although the generalised adversary bound does so up to a constant factor [17].

1.2 Organisation

The remainder of this paper is organised as follows. After formalising some definitions in Section 2, in Section 3 we move on to techniques for finding separations between exact quantum query algorithms, classical algorithms, and quantum algorithms computing parities of input bits. We then discuss in Section 4 how the Barnum-Saks-Szegedy SDP can be solved for small problems to give explicit quantum query algorithms. Section 5 gives our algorithm for for determining whether a 4-bit input has Hamming weight 2. In Section 6 we give optimal exact quantum query algorithms, found by semidefinite programming, for every boolean function on 3 bits. Our results characterising nonadaptive exact quantum query complexity are in Section 7, after which we conclude with a discussion of open problems. Two appendices detail our numerical results for all 4-bit boolean functions, and all symmetric functions on up to 6 bits, and also give example CVX source code.

2 Definitions

2.1 Generalities and boolean functions

For any bit-string x, $|x|$ will denote the Hamming weight of x, and $\bar{x}$ will denote $\mathrm{NOT}(x)$. e_i will denote the bit-string with 1 in the i'th position, and 0 elsewhere. We use the convention [X] for a quantity which evaluates to 1 if the statement X is true, and 0 otherwise. We will be interested in boolean functions $f : \{0,1\}^n \to \{0,1\}$, and particularly interested in the following families of functions, all on n bits. PARITY is the function $\mathrm{PARITY}(x) = [|x| \text{ is odd}]$. MAJ is the majority function where $\mathrm{MAJ}(x) = [|x| \geq n/2]$. The EXACT_k function is defined by $\mathrm{EXACT}_k(x) = [|x| = k]$. NAE ("not-all-equal") evaluates to 0 if all the input bits are equal, otherwise evaluates to 1. Finally, the threshold function $\mathrm{Th}_k(x)$ evaluates to 1 if and only if $|x| \geq k$. All of these are examples of *symmetric* boolean functions; a boolean function f is said to be symmetric if $f(x)$ only depends on $|x|$. A non-symmetric function on 3 bits which we will consider is SEL, where $\mathrm{SEL}(x_1, x_2, x_3)$ is defined to be equal to x_2 if $x_1 = 0$, and equal to x_3 if $x_1 = 1$.

We say that two boolean functions f and g are *isomorphic* if they are equal up to negations and permutations of the input variables, and negation of the output variable. This relationship is sometimes known as NPN-equivalence.

2.2 Query complexity model

An exact classical query algorithm to compute a boolean function f is given by a decision tree, or in other words a rooted binary tree whose vertices represent variables x_i to be queried, and whose edges represent branching depending on whether $x_i = 0$ or $x_i = 1$. The number of queries used is the depth of the decision tree; the minimal depth over all decision trees is the exact classical query complexity (aka decision tree complexity) $D(f)$.

We follow what is essentially the standard quantum query complexity model (see e.g. [2, 14]). A quantum query algorithm to compute a boolean function $f : \{0,1\}^n \to \{0,1\}$ is specified by a sequence of unitary operators $U_0, \ldots, U_t$ which do not depend on the (initially unknown) input x. These unitaries are interspersed with oracle calls O_x (which do depend on the input x). The final state of an algorithm that makes t queries, before the final measurement, is given by $U_t O_x U_{t-1} O_x \ldots O_x U_0 |0\rangle$. The overall Hilbert space $\mathcal{H}$ used by the quantum query algorithm is split into three subspaces $\mathcal{H}_{\text{in}} \otimes \mathcal{H}_{\text{work}} \otimes \mathcal{H}_{\text{out}}$. For boolean functions, $\mathcal{H}_{\text{out}}$ is a single qubit, whereas the workspace $\mathcal{H}_{\text{work}}$ is of arbitrary size. The oracle O_x acts only on $\mathcal{H}_{\text{in}}$ by mapping $|i\rangle \mapsto (-1)^{x_i}|i\rangle$ for some hidden bit string x. $\mathcal{H}_{\text{in}}$ is $n+1$ dimensional and indexed by basis vectors $|0\rangle, \ldots, |n\rangle$; a query to x_0 always returns 0. The final step of the algorithm is always simply to measure the $\mathcal{H}_{\text{out}}$ register and return the outcome. We say that the algorithm computes f within error ϵ if, on input x, the algorithm returns $f(x)$ with probability at least $1 - \epsilon$ for all x. The exact quantum query complexity $Q_E(f)$ is the minimum number of queries used by any quantum algorithm which computes $f(x)$ exactly for all x.

Note that if boolean functions f and g are isomorphic, $D(f) = D(g)$ and $Q_E(f) = Q_E(g)$.

3 Separating exact quantum and classical query complexity

We observe that any fixed function demonstrating a separation between exact quantum and classical query complexity, even a small additive constant, gives rise to a constant factor multiplicative separation for an asymptotically growing family of functions as follows.

Proposition 1. *Let $f : \{0,1\}^k \to \{0,1\}$ be a boolean function on k bits such that $Q_E(f) = m_q$ and $D(f) = m_c$ for some $m_q \leq m_c$. Also let $g : \{0,1\}^n \to \{0,1\}$ be a boolean function such that $D(g) = n$. Define the family of functions $f_n : \{0,1\}^{nk} \to \{0,1\}$ as follows: divide the nk input bits into blocks $b_1, \ldots, b_n$ of k bits each, and set $f_n(x_1, \ldots, x_{nk}) = g(f(b_1), f(b_2), \ldots, f(b_n))$. Then $Q_E(f_n) \leq m_q n$ and $D(f_n) = m_c n$.*

Proof. An exact quantum query algorithm for f_n using $m_q n$ queries can be obtained simply by computing $f(b_i)$ for each block b_i individually, and then classically computing $g(f(b_1), \ldots, f(b_n))$, so $Q_E(f_n) \leq m_q n$. On the other hand, any classical algorithm for f_n must know the value of $f(b_i)$ for all i with certainty. Otherwise, this would imply an algorithm that computes g using fewer than n queries, contradicting $D(g) = n$. Hence $D(f_n) = m_c n$. □

Note that, in contrast to the classical case, it is not necessarily true that the reverse inequality $Q_E(f_n) \geq m_q n$ holds; a counterexample is provided by the PARITY function $x_1 \oplus x_2$ on 2 bits. Natural examples of functions g to which Proposition 1 can be applied are AND and OR. Thus an example of an exact query complexity separation we obtain from our results on small boolean functions is the following (see Section 5).

Theorem 2. *Let $EXACT_2^\ell$ be the boolean function on 4ℓ bits defined as follows. Split the input x into consecutive blocks $b_1, \ldots, b_\ell$ containing 4 bits each, and set $EXACT_2^\ell(x_1, \ldots, x_{4\ell}) = 1$ if each block b_i contains exactly two 1s. Then $Q_E(EXACT_2^\ell) = 2\ell$ and $D(EXACT_2^\ell) = 4\ell$.*

Note that the lower bound $Q_E(\mathrm{EXACT}_2^\ell) \geq 2\ell$ in this theorem does not follow immediately from Proposition 1, but can be shown using polynomial degree arguments [3].

3.1 Quantum algorithms based on parity queries

It is well-known that quantum computers can evaluate the parity of two input bits exactly using only one query [5]. Thus a non-trivial class of exact quantum query algorithms consists of classical decision trees, each of whose nodes is a query either to an individual bit of the input, or to the parity of two input bits. One can put a lower bound on the number of queries used by such algorithms (indeed, a more general class of algorithms) as follows.

Proposition 3. *Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function, and let d be the degree of f as an n-variate polynomial over $\mathbb{F}_2$. Then any decision tree which can query the parity of any subset of the input variables at unit cost must make at least d queries to the input to compute f with certainty.*

Proof. We will show by induction that any decision tree on parities which has depth D gives rise to a degree D polynomial over $\mathbb{F}_2$. The function computed by any such tree can be written as $pT_0 + (1+p)T_1$ for some degree 1 polynomial p over $\mathbb{F}_2$ and decision trees T_0, T_1 of depth at most $D-1$. Therefore, the degree of the polynomial obtained by repeating this procedure recursively is at most D. If the tree computes f on every input, this polynomial must be equal to f, and hence be degree d. Thus $D \geq d$. □

4 Quantum query algorithms from semidefinite programming

In this section we discuss, following [2], how quantum query complexity can be evaluated as a semidefinite programming (SDP) problem, given in Definition 1 below. In this definition, $\circ$ is the Hadamard (entrywise) product of matrices. The following important characterisation will be the key to many of our results.

Theorem 4 (Barnum, Saks and Szegedy [2])**.** *There is a quantum query algorithm that uses t queries to compute a function $f : \{0,1\}^n \to \{0,1\}$ within error ϵ if and only if the SDP of Definition 1 is feasible.*

Therefore, if one minimises ϵ subject to these semidefinite constraints, one obtains the lowest possible error that can be achieved by a quantum algorithm which computes f using t queries.

4.1 A prescription for quantum algorithms

It is perhaps not immediately obvious how, given a solution to the semidefinite program of Definition 1, to produce a quantum query algorithm with the same parameters. This was implicit in [2]; we now spell out explicitly how it can be done. We will use the following standard lemma from linear algebra (see e.g. [13]).

Definition 1 (Quantum query complexity primal SDP)**.**

For a given boolean function $f : \{0,1\}^n \to \{0,1\}$ and a given integer t, find a sequence of 2^n-dimensional real symmetric matrices $(M_i^{(j)})$, where $0 \leq i \leq n$ and $0 \leq j \leq t-1$, and 2^n-dimensional real symmetric matrices Γ_0, Γ_1, satisfying the following constraints.

$$\sum_{i=0}^{n} M_i^{(0)} = E_0 \tag{1}$$

$$\sum_{i=0}^{n} M_i^{(j)} = \sum_{i=0}^{n} E_i \circ M_i^{(j-1)} \text{ (for } 1 \leq j \leq t-1) \tag{2}$$

$$\Gamma_0 + \Gamma_1 = \sum_{i=0}^{n} E_i \circ M_i^{(t-1)} \tag{3}$$

$$F_0 \circ \Gamma_0 \geq (1-\epsilon)F_0 \tag{4}$$

$$F_1 \circ \Gamma_1 \geq (1-\epsilon)F_1 \tag{5}$$

where E_i is the matrix $\langle x|E_i|y\rangle = (-1)^{x_i+y_i}$, and F_0 and F_1 are diagonal matrices defined by $\langle x|F_z|x\rangle = 1$ if and only if $f(x) = z$, and $\langle x|F_z|x\rangle = 0$ otherwise.

Lemma 5. *Let $S = (|\psi_i\rangle)$ and $T = (|\phi_j\rangle)$ be two sequences of m vectors of the same dimension. Define $\Psi = \sum_i |\psi_i\rangle\langle i|$, $\Phi = \sum_j |\phi_j\rangle\langle j|$. Then there is a unitary U such that $U|\phi_i\rangle = |\psi_i\rangle$ for all i if and only if $\Psi^\dagger\Psi = \Phi^\dagger\Phi$. If such a U exists, it can be written down as follows. Let V and W be any isometries satisfying $\Psi = V\sqrt{\Psi^\dagger\Psi}$, $\Phi = W\sqrt{\Phi^\dagger\Phi}$ (i.e. isometries occurring in polar decompositions of Ψ, Φ), and complete V and W to unitary matrices V' and W'. Then $U = V'(W')^\dagger$.*

Given a set of matrices $M_i^{(j)}$ as in Definition 1, one can derive an explicit quantum query algorithm completely mechanically, as follows. Use a workspace $\mathcal{H}_{\text{work}}$ of n qubits, and ignore the output qubit for the time being. Let the initial state be $|0\rangle|0\rangle$, and let the state of the system at time j (i.e. after j queries have been made, and just before the $(j+1)$'st query is made) be $|\psi_x^{(j)}\rangle = \sum_{i=0}^{n} |i\rangle|\psi_{x,i}^{(j)}\rangle$, where $|\psi_{x,i}^{(j)}\rangle$ is a subnormalised state in the Hilbert space $\mathcal{H}_{\text{work}}$.

We will *define* the state at time $0 \leq j \leq t-1$ in terms of the matrices $M_i^{(j)}$ occurring in a solution to the SDP by setting

$$|\psi_{x,i}^{(j)}\rangle = \sqrt{M_i^{(j)}}|x\rangle, \tag{6}$$

where $\sqrt{M_i^{(j)}}$ is the unique positive semidefinite square root of $M_i^{(j)}$. It is perhaps not immediately clear that following this prescription leads to $|\psi_x^{(j)}\rangle$ being normalised, let alone the sequence $|\psi_x^{(0)}\rangle, |\psi_x^{(1)}\rangle, \ldots, |\psi_x^{(t-1)}\rangle$ corresponding to a valid quantum query algorithm for all x; however, we will now see that this is indeed the case.

For any $1 \leq j \leq t$, define

$$|\phi_x^{(j)}\rangle = \sum_{i=0}^{n} (-1)^{x_i}|i\rangle|\psi_{x,i}^{(j-1)}\rangle,$$

and set $|\phi_x^{(0)}\rangle = |0\rangle|0\rangle$. Also define the matrices obtained by concatenating the vectors as columns,

$$\Psi^{(j)} = \sum_{x\in\{0,1\}^n} |\psi_x^{(j)}\rangle\langle x|, \ \Phi^{(j)} = \sum_{x\in\{0,1\}^n} |\phi_x^{(j)}\rangle\langle x|.$$

The vectors $|\phi_x^{(j)}\rangle$ represent the state of the system immediately *after* the j'th oracle call, and $|\phi_x^{(0)}\rangle$ is the initial state of the system. We would like to find unitaries $U_0, \ldots, U_{t-1}$ mapping $|\phi_x^{(j)}\rangle \mapsto |\psi_x^{(j)}\rangle$ for all x. By Lemma 5, such a sequence of unitaries will exist if

$$(\Psi^{(j)})^\dagger \Psi^{(j)} = (\Phi^{(j)})^\dagger \Phi^{(j)}.$$

But observe that for any $0 \leq j \leq t-1$,

$$\langle x|(\Psi^{(j)})^\dagger \Psi^{(j)}|y\rangle = \langle \psi_x^{(j)}|\psi_y^{(j)}\rangle = \sum_{i=0}^{n} \langle \psi_{x,i}^{(j)}|\psi_{y,i}^{(j)}\rangle = \sum_{i=0}^{n} \langle x|M_i^{(j)}|y\rangle,$$

so $(\Psi^{(j)})^\dagger \Psi^{(j)} = \sum_{i=0}^n M_i^{(j)}$. Similarly, for any $1 \leq j \leq t$,

$$\begin{aligned}\langle x|(\Phi^{(j)})^\dagger \Phi^{(j)}|y\rangle &= \langle \phi_x^{(j)}|\phi_y^{(j)}\rangle = \sum_{i=0}^{n} (-1)^{x_i+y_i} \langle \psi_{x,i}^{(j-1)}|\psi_{y,i}^{(j-1)}\rangle = \sum_{i=0}^{n} (-1)^{x_i+y_i} \langle x|M_i^{(j-1)}|y\rangle \\ &= \langle x| \left(\sum_{i=0}^{n} E_i \circ M_i^{(j-1)} \right) |y\rangle.\end{aligned}$$

As, by constraint (2) in the SDP, $\sum_{i=0}^n M_i^{(j)} = \sum_{i=0}^n E_i \circ M_i^{(j-1)}$ for all $1 \leq j \leq t-1$, this implies by Lemma 5 that for each $j \geq 1$ there exists a unitary U_j such that $U_j|\phi_x^{(j)}\rangle = |\psi_x^{(j)}\rangle$ for all x, and this U_j can be determined explicitly from Lemma 5. In the case $j = 0$, $(\Phi^{(j)})^\dagger \Phi^{(j)} = E_0$, and hence SDP constraint (1) implies the existence of a U_0 such that $U_0|\phi_x^{(0)}\rangle = |\psi_x^{(0)}\rangle$ for all x.

The final constraint we need to satisfy is that the algorithm outputs the correct result. Define the final state of the system on input x (just before the output qubit is measured) to be

$$|\gamma_x\rangle = |0\rangle_{\text{in}}(\sqrt{\Gamma_0}|x\rangle)_{\text{work}}|0\rangle_{\text{out}} + |0\rangle_{\text{in}}(\sqrt{\Gamma_1}|x\rangle)_{\text{work}}|1\rangle_{\text{out}}.$$

Then

$$\langle \gamma_x|\gamma_y\rangle = \langle x|\Gamma_0|y\rangle + \langle x|\Gamma_1|y\rangle = \langle x| \left(\sum_{i=0}^{n} E_i \circ M_i^{(t-1)} \right) |y\rangle$$

by SDP constraint (3), so by a similar argument there exists a U_t such that $U_t|\phi_x^{(t)}\rangle = |\gamma_x\rangle$ for all x. Measuring the output qubit gives the answer 0 with probability $\langle x|\Gamma_0|x\rangle$, which by constraint (4) is at least $1-\epsilon$ when $f(x) = 0$. Similarly, by constraint (5) we obtain the answer 1 with probability at least $1-\epsilon$ when $f(x) = 1$.

Observe that we have some freedom in our choice of states $|\psi_{x,i}^{(j)}\rangle$; while eqn. (6) gives one choice which always works, it would suffice to pick any states such that $\langle \psi_{x,i}^{(j)}|\psi_{y,i}^{(j)}\rangle = \langle x|M_i^{(j)}|y\rangle$. In particular, if the rank of $M_i^{(j)}$ is upper bounded by r for all i, j, one can choose states of dimension r throughout. This would reduce the size of the $\mathcal{H}_{\text{work}}$ register from n qubits to $\lceil \log_2 r \rceil$ qubits. Also observe that without loss of generality all states and unitaries occurring in a quantum query algorithm can be taken to be real.

5 EXACT$_2$

We now give a simple and explicit quantum algorithm for evaluating the EXACT$_2$ function on 4 bits using only 2 quantum queries. This algorithm was originally inspired by numerically solving the SDP discussed in the previous section. The algorithm does not use any workspace (or even an output register), and hence operates solely on the 5-dimensional input register indexed by basis states $\{|0\rangle, \ldots, |4\rangle\}$. Define a unitary matrix U by

$$U = \frac{1}{2}\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & \omega & \omega^2 \\ 1 & 1 & 0 & \omega^2 & \omega \\ 1 & \omega & \omega^2 & 0 & 1 \\ 1 & \omega^2 & \omega & 1 & 0 \end{pmatrix},$$

where $\omega = e^{2\pi i/3}$ is a complex cube root of 1. We begin in the state

$$|\psi\rangle = \frac{1}{2}\sum_{i=1}^{4} |i\rangle$$

and then apply O_x, then U, then O_x again. Finally, we perform the measurement consisting of a projection onto the state $|\psi\rangle$ and its orthogonal complement. If the outcome is $|\psi\rangle$, we output 1, and otherwise 0.

The claim is that $V_x := O_x U O_x$ leaves $|\psi\rangle$ unchanged, up to a phase factor, when x has Hamming weight 2, and otherwise maps $|\psi\rangle$ into a subspace orthogonal to $|\psi\rangle$. To see that the claim is correct, note first that $U|\psi\rangle = |0\rangle$, since $1 + \omega + \omega^2 = 0$. But for $x = 0000$, O_x is the identity. Thus

$$V_{0000}|\psi\rangle = |0\rangle.$$

For $x = 1000$, $O_x|\psi\rangle = |\psi\rangle - |1\rangle$. So $V_x|\psi\rangle = |0\rangle - O_x U|1\rangle$. But the coefficient of $|1\rangle$ in $U|1\rangle$ is zero, so O_x leaves $U|1\rangle$ unchanged and we have

$$V_{1000}|\psi\rangle = |0\rangle - U|1\rangle.$$

Similar results hold for the other weight 1 strings x.

For $x = 1100$, $O_x|\psi\rangle = |\psi\rangle - |1\rangle - |2\rangle$, and

$$\begin{aligned} U(|\psi\rangle - |1\rangle - |2\rangle) &= |0\rangle - \frac{1}{2}\left(2|0\rangle + |1\rangle + |2\rangle + (\omega + \omega^2)(|3\rangle + |4\rangle)\right) \\ &= \frac{1}{2}(-|1\rangle - |2\rangle + |3\rangle + |4\rangle). \end{aligned}$$

Applying O_x once more we get

$$V_{1100}|\psi\rangle = |\psi\rangle.$$

We get the same result for other strings of weight 2, possibly with a phase factor. For instance

$$V_{1001}|\psi\rangle = \omega^2|\psi\rangle.$$

Given a string x of weight 3, we can flip all the bits and the oracle acts identically but with a change of sign. Thus such a string behaves like one of weight 1, and similarly a string of weight 4 behaves like one of weight 0. Thus, for x such that $|x| \neq 2$, $V_x|\psi\rangle$ lies in the span of $|0\rangle$ and $U|i\rangle$ for $i = 1, 2, 3, 4$. However $\langle\psi|0\rangle = 0$ and $\langle\psi|U|i\rangle = \frac{1}{4}(1 + \omega + \omega^2) = 0$ for $i = 1, 2, 3, 4$. So this subspace is orthogonal to $|\psi\rangle$, proving our claim.

5.1 Distinguishing weights 0 and 1 from balanced strings

We would ideally like to understand the number of queries required to solve the EXACT_k function on n input bits, for all n and k. As an intermediate goal generalising our solution for $n = 4$, one can ask for a two-query algorithm, for any even n, that distinguishes strings of weight 0 and 1 from strings of weight $n/2$. We take the previous space, with basis vectors $|i\rangle$, $i = 0, 1, \ldots, n$, and tensor it with an ancilla space. In the ancilla space we select vectors $|a_i\rangle$, $i = 1, \ldots, n$, of unit length with inner products $\langle a_i|a_j\rangle = c$, for $i \neq j$, where c is arbitrary such that $|c| \leq 1$. Such vectors can always be found. We also select some vector $|0\rangle$ in the ancilla space.

The oracle acts on the first space, so $O_x|i\rangle|j\rangle = (-1)^{x_i}|i\rangle|j\rangle$ for any i, j. Recall that e_i is the string with a 1 at position i and 0's elsewhere, and let b be the "balanced" string with 1's in the first n/2 places. The algorithm starts with the state $|\phi\rangle = \sum_{i=1}^{n} |i\rangle|0\rangle$ (we keep $|\phi\rangle$ unnormalised for simplicity). As before, $V_x := O_x U O_x$, with U to be defined shortly. The aim is to show that $V_b|\phi\rangle$ is orthogonal to the subspace spanned by $V_{e_i}|\phi\rangle$ for all i, which thus discriminates between balanced and weight 1 strings (discriminating weight 0 strings will follow automatically from this).

Now define U by its action on the states $|\tau_i\rangle := O_{e_i}|\phi\rangle = |\phi\rangle - 2|i\rangle|0\rangle$:

$$U|\tau_i\rangle = \alpha|00\rangle + \beta\left(\sum_{j=1}^{n} |j\rangle|a_{j+i-1}\rangle - |i\rangle|a_{2i-1}\rangle\right) \tag{7}$$

where the subscript of the a's is taken mod n. This will be an isometry (which can be extended to a unitary on the whole tensor product space) if

$$\alpha^2 + (n-1)\beta^2 = n, \tag{8}$$

$$\alpha^2 + (n-2)\beta^2 c = n - 4. \tag{9}$$

Note that $U|\tau_i\rangle = V_{e_i}|\phi\rangle$, since $U|\tau_i\rangle$ is invariant under O_{e_i}.

U can be defined on $O_b|\phi\rangle$ by using

$$O_b|\phi\rangle = \frac{1}{2}\left(\sum_{i=1}^{n/2} |\tau_i\rangle - \sum_{i=n/2+1}^{n} |\tau_i\rangle\right),$$

giving, up to an overall factor of β,

$$V_b|\phi\rangle = \frac{1}{2}\left(\sum_{i=1}^{n/2} O_b U|\tau_i\rangle - \sum_{i=n/2+1}^{n} O_b U|\tau_i\rangle\right) = \sum_{j=1}^{n}\sum_{k=1}^{n} \pi_{jk}|j\rangle|a_{k+j-1}\rangle + \sum_{j=1}^{n} |j\rangle|a_{2j-1}\rangle, \tag{10}$$

where

$$\pi_{jk} = -1 \text{ if } i \in L, j \in L, \qquad \pi_{jk} = 1 \text{ if } i \in L, j \in H,$$
$$\pi_{jk} = 1 \text{ if } i \in H, j \in L, \qquad \pi_{jk} = -1 \text{ if } i \in H, j \in H,$$

and $L = [1, n/2], H = [n/2 + 1, n]$. Combining (10) with (7) gives

$$\begin{aligned}\langle\phi|V_b V_{e_i}|\phi\rangle &= \sum_j \sum_k \pi_{jk}\langle a_{j+k-1}|a_{j+i-1}\rangle - \sum_j \langle a_{i+j-1}|a_{2i-1}\rangle + \sum_j \langle a_{2j-1}|a_{i+j-1}\rangle - \langle a_{2i-1}|a_{2i-1}\rangle \\ &= 0 - (c-1) + (1 + (n-1)c) - 1 \\ &= 1 + (n-2)c.\end{aligned}$$

Thus for orthogonality we require $n = -1/(n-2)$, which, with (8) and (9), gives $\alpha^2 = (n-2)^2/n$ and $\beta^2 = 4/n$.

Finally, we would like to show orthogonality for the weight zero sequence, i.e. orthogonality of $V_b|\phi\rangle$ and $U|\phi\rangle$. However, this follows from what we proved above, using

$$|\phi\rangle = \frac{1}{n-2}\sum_{i=1}^{n}|\tau_i\rangle,$$

together with the fact that $U|\tau_i\rangle$ is invariant under O_{e_i}.

6 Exact quantum query algorithms for small functions

Having whetted our appetite with the EXACT_2 problem, we now turn to quantum algorithms for other small boolean functions. For each function on n bits we considered, we calculated, via the SDP of Definition 1, the best possible success probability achievable by quantum algorithms making t queries, for $t = 1, \ldots, n-1$ (any function can clearly be computed exactly using n queries). We did this for all functions on up to 4 input bits, and for all symmetric functions on up to 6 bits. We used the convex optimisation package CVX [11] for Matlab, which allows optimisation problems to be specified in a simple and intuitive way; see Appendix B for source code. The CVX package allows the choice of underlying solvers SeDuMi and SDPT3. We used SeDuMi for the results given below, and also checked the results with SDPT3, which gave the same values up to a difference of at most 0.001. The numerical results for functions on 4 bits, and symmetric functions on up to 6 bits, are deferred to Appendix A.

Note that there is a basic issue with calculating *exact* quantum query complexity numerically, which is that one receives a numerical solution from the SDP solver, which is not exact. If the SDP solver claims that there exists a quantum query algorithm that computes some function f using k queries with success probability at least 0.999, for example, one cannot be sure that this algorithm is actually exact. In the case of all functions on up to 3 bits, we therefore give explicit optimal *exact* quantum query algorithms. These algorithms were obtained by a somewhat laborious process of taking the numerically obtained (real-valued, approximate) solutions to the SDP and using these as a guide to find exact solutions.

For completeness, we begin by giving optimal exact quantum query algorithms for all boolean functions of 1 and 2 bits. In what follows, the tables are indexed by function ID; the ID of each function is the integer obtained by converting its truth table from binary. Columns give the optimal success probability that can be achieved by quantum algorithms making $1, \ldots, n-1$ queries. Entries are starred when there is a *nonadaptive* exact quantum algorithm using that number of queries (see Section 7).

6.1 Functions of up to 2 bits

Up to isomorphism, the only non-constant function of 1 bit is $f(x_1) = x_1$, which clearly requires exactly one query. In the case of 2 bits, there are two classes of functions.

ID	Function	1 query
1	$x_1 \wedge x_2$	0.900
6	$x_1 \oplus x_2$	1*

An optimal quantum algorithm for the function $x_1 \oplus x_2$ proceeds as follows [5]. Input the state $\frac{1}{\sqrt{2}}(|1\rangle + |2\rangle)$ into the oracle to produce $\frac{1}{\sqrt{2}}((-1)^{x_1}|1\rangle + (-1)^{x_2}|x_2\rangle)$. Perform a Hadamard gate (with respect to the basis $\{|1\rangle, |2\rangle\}$), measure in the basis $\{|1\rangle, |2\rangle\}$, and output 0 if "1" is measured, and 1 if "2" is measured. It is easy to see that this algorithm succeeds with certainty.

6.2 Functions of 3 bits

The following table lists all boolean functions depending on 3 bits, up to isomorphism.

ID	Function	1 query	2 queries	$\mathbb{F}_2$ degree	D(f)
1	$x_1 \wedge x_2 \wedge x_3$	0.800	0.980	3	3
6	$x_1 \wedge (x_2 \oplus x_3)$	0.667	1*	2	3
7	$x_1 \wedge (x_2 \vee x_3)$	0.773	1	3	3
22	EXACT_2	0.571	1	3	3
23	MAJ	0.667	1	2	3
30	$x_1 \oplus (x_2 \vee x_3)$	0.667	1	2	3
53	$\mathrm{SEL}(x_1, x_2, x_3)$	0.854	1	2	2
67	$(x_1 \wedge x_2) \vee (\bar{x_1} \wedge \bar{x_2} \wedge x_3)$	0.773	1	3	3
105	PARITY	0.500	1*	1	3
126	NAE	0.900	1*	2	3

Observe that the AND function requires 3 queries to be computed exactly; in fact, it has been known for some time that AND on n bits has $Q_E(\mathrm{AND}) = n$ [3]. For most of the other functions, an optimal exact quantum algorithm is easy to determine, based only on classical queries and computing the parity of two bits using one query.

- $x_1 \wedge (x_2 \oplus x_3)$: Query x_1 and evaluate $x_2 \oplus x_3$ using one query.
- MAJ: First evaluate $x_1 \oplus x_2$. If the answer is 1, then output x_3, otherwise output x_1. This works because if x_1 and x_2 are different, then there will be at least two 1's in total if and only if x_3 is 1. If x_1 and x_2 are the same, there will be at least two 1's if and only if x_1 is 1.
- $x_1 \oplus (x_2 \vee x_3)$: This function is equivalent to $(\bar{x_3} \wedge x_2) \vee (x_3 \wedge (x_1 \oplus x_2))$. So query x_3 first, then either query x_2 or $x_1 \oplus x_2$.
- $\mathrm{SEL}(x_1, x_2, x_3)$: Query x_1 first, then either output x_2 or x_3.
- PARITY: Evaluate $x_1 \oplus x_2$, query x_3, and take the exclusive OR of the two.
- NAE: This function is equivalent to $(x_1 \oplus x_2) \vee (x_1 \oplus x_3)$.

However, the three remaining functions (EXACT_2, $x_1 \wedge (x_2 \vee x_3)$ and $(x_1 \wedge x_2) \vee (\bar{x_1} \wedge \bar{x_2} \wedge x_3)$) do not have such straightforward optimal algorithms. Indeed, by Proposition 3, they cannot be computed using 2 queries by any algorithm which is a decision tree on parity queries.

In the case of EXACT_2, we obtain an optimal algorithm by appending an additional zero bit and computing EXACT_2 on 4 bits (see Section 5). We now give quantum query algorithms for the two remaining functions. Rather than writing out the unitary operators arising in the algorithm explicitly, we simply give expressions for matrices forming a exact solution to the query complexity SDP. We stress that, given these matrices, one can follow the procedure of Section 4 to find an

explicit quantum algorithm completely mechanically. The matrices are fully specified by their non-zero eigenvalues and eigenvectors; note that, for readability, we have neither normalised nor orthogonalised the eigenvectors.

6.2.1 $x_1 \wedge (x_2 \vee x_3)$

Matrix	Eigenvalues	Eigenvectors
$M_0^{(0)}, M_1^{(0)}, M_2^{(0)}, M_3^{(0)}$	2	$(1,1,1,1,1,1,1,1)$
$M_0^{(1)}$	3/2	$(1,1,1,1,0,0,0,0)$
	1	$\{(-1,1,-1,1,0,2,0,2),(0,-1,1,0,0,-1,1,0)\}$
$M_1^{(1)}$	1	$(1,0,0,-1,2,1,1,0)$
	1/2	$(-1,0,-1,0,0,1,0,1)$
$M_2^{(1)}$	1	$(1,0,0,-1,2,1,1,0)$
	1/2	$(-1,-1,0,0,0,0,1,1)$
$M_3^{(1)}$	3/4	$\{(0,1,0,1,0,1,0,1),(0,-1,1,0,0,-1,1,0)\}$
Γ_0	5/2	$(3,2,2,3,2,0,0,0)$
	1	$\{(0,-1,0,0,1,0,0,0),(0,-1,1,0,0,0,0,0)\}$
	1/2	$(-1,0,0,1,0,0,0,0)$
Γ_1	3/2	$\{(0,0,0,0,0,1,0,1),(0,0,0,0,0,-1,1,0)\}$

6.2.2 $(x_1 \wedge x_2) \vee (\bar{x_1} \wedge \bar{x_2} \wedge x_3)$

Matrix	Eigenvalues	Eigenvectors
$M_0^{(0)}, M_1^{(0)}, M_2^{(0)}, M_3^{(0)}$	2	$(1,1,1,1,1,1,1,1)$
$M_0^{(1)}$	1	$(-2,-1,-1,0,-1,0,0,1)$
	3/4	$(0,0,-1,-1,2,2,1,1)$
	1/4	$(0,0,1,1,0,0,1,1)$
$M_1^{(1)}$	1	$(-2,-1,-1,0,-1,0,0,1)$
	3/4	$(0,-1,2,1,-1,-2,1,0)$
	1/4	$(0,-1,0,-1,1,0,1,0)$
$M_2^{(1)}$	1	$(0,1,1,2,1,2,2,3)$
	3/4	$(0,-1,0,-1,1,0,1,0)$
	1/4	$(0,-1,2,1,-1,-2,1,0)$
$M_3^{(1)}$	1	$(0,3,-1,2,-1,2,-2,1)$
	3/4	$(0,0,1,1,0,0,1,1)$
	1/4	$(0,0,-1,-1,2,2,1,1)$
Γ_0	$\frac{1}{4}(5+\sqrt{5})$	$(1+\sqrt{5},0,-1+\frac{1}{2}(5+\sqrt{5}),1,-1+\frac{1}{2}(5+\sqrt{5}),1,0,0)$
	3/2	$(0,0,0,-1,0,1,0,0)$
	1	$(0,0,-1,0,1,0,0,0)$
	$\frac{1}{4}(5-\sqrt{5})$	$(1-\sqrt{5},0,-1+\frac{1}{2}(5-\sqrt{5}),1,-1+\frac{1}{2}(5-\sqrt{5}),1,0,0)$
Γ_1	3/2	$\{(0,-1,0,0,0,0,0,1),(0,1,0,0,0,0,1,0)\}$

7 Nonadaptive exact quantum query complexity

We now turn to a very restricted model of exact query complexity, in which the algorithm's queries are required to be nonadaptive. In other words, the choice of which input variables to query cannot depend on the result of previous queries, so the algorithm must choose which variables to query at the start. For a boolean function f, let $D^{na}(f)$, $Q_E^{na}(f)$ be the nonadaptive quantum and classical exact query complexities of f, i.e. the minimum number of nonadaptive queries required to compute f with certainty. This model is extremely restricted classically, as we see from the following easy proposition.

Proposition 6. *For any boolean function f depending on n variables, $D^{na}(f) = n$.*

Proof. A nonadaptive exact classical query algorithm $\mathcal{A}$ making k queries is specified by a list of k fixed variables which are queried. If $k < n$, there must exist a variable i which is not queried, but on which f depends. Thus there must exist an input x such that if bit i is flipped, $\mathcal{A}$ does not notice the difference, so $\mathcal{A}$ cannot be correct on every input. □

In the case of nonadaptive *quantum* query complexity, things are more interesting. An optimal quantum algorithm for PARITY is nonadaptive and uses $\lceil n/2 \rceil$ quantum queries [5, 10], so quantum algorithms can indeed achieve an advantage over classical algorithms in this model. However, it is also known that this separation by a factor of 2 is optimal for total functions in the nonadaptive model [16].

We now introduce some additional notation. For any boolean function $f : \{0,1\}^n \to \{0,1\}$, define

$$S_f := \{z : \forall x, f(x) = f(x+z)\},$$

where addition is over the group $\mathbb{Z}_2^n$; i.e. S_f is the set of translations of the input under which f is invariant. Note that S_f is a subspace of $\{0,1\}^n$. For any subspace $S \subseteq \{0,1\}^n$, let $S^\perp$ denote the orthogonal subspace to S, i.e. $S^\perp = \{x : x \cdot s = 0, \forall\, s \in S\}$. Finally, let $d(x,S)$ denote the *maximum* Hamming distance between a bit-string $x \in \{0,1\}^n$ and a subset $S \subseteq \{0,1\}^n$: $d(x,S) = \max_{y \in S} d(x,y)$. Then we have the following theorem.

Theorem 7. *For any boolean function $f : \{0,1\}^n \to \{0,1\}$,*

$$Q_E^{na}(f) = \min_{x \in \{0,1\}^n} \max_{y \in S_f^\perp} d(x,y) = \min_{x \in \{0,1\}^n} d(x, S_f^\perp).$$

We have thus completely characterised the nonadaptive quantum query complexity of f. In the coding theory literature, the quantity $\min_{x \in \{0,1\}^n} d(x, S_f^\perp)$ is known as the *radius* of the code $S_f^\perp$ [6]. Observe that Theorem 7 implies that $Q_E^{na}(f)$ can be computed exactly in time polynomial in 2^n. We now prove this theorem and then draw some corollaries. In the proof it will be convenient to use the notation

$$\hat{f}(x) = \frac{1}{2^n} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} f(y)$$

for the Fourier transform (over $\mathbb{Z}_2^n$) of some function $f : \{0,1\}^n \to \mathbb{R}$.

Proof of Theorem 7. Any nonadaptive quantum algorithm making k queries to the input x corresponds to a choice of a state $|\psi\rangle$, which is input to k copies of the oracle (i.e. the unitary operator $O_x^{\otimes k}$), followed by a projective measurement to determine whether $f(x) = 0$ or $f(x) = 1$. Label

computational basis states by a length k string of integers $(i_1, \ldots, i_k)$ in the range $\{0, \ldots, n\}$; each such string represents a list of variables queried, with 0 representing a "null query" which does nothing. $O_x^{\otimes k}$ acts on these basis states by mapping

$$|i_1, \ldots, i_k\rangle \mapsto (-1)^{x_{i_1}+\cdots+x_{i_k}}|i_1, \ldots, i_k\rangle.$$

Now note that we can restrict ourselves to query strings in non-decreasing order, and containing at most one of each integer between 1 and n. The first of these is because querying any permutation of a string is equivalent to querying the string itself. The second is because querying the same index twice does nothing, and hence is equivalent to the null query.

These strings are now in obvious one-to-one correspondence with the set of n-bit strings of Hamming weight at most k. Thus the state we obtain from applying $O_x^{\otimes k}$ to an arbitrary input state is of the form

$$|\psi_x\rangle = \sum_{s\in\{0,1\}^n, |s|\leq k} (-1)^{s\cdot x}\alpha_s|s\rangle.$$

A nonadaptive quantum query algorithm computing f exactly using k queries exists if and only if there exists a set $\{\alpha_s\}$ such that $\langle\psi_x|\psi_y\rangle = 0$ for all x, y such that $f(x) \neq f(y)$, or in other words

$$\sum_{s\in\{0,1\}^n, |s|\leq k} (-1)^{s\cdot(x+y)}|\alpha_s|^2 = 0.$$

For brevity, write $w(s) = |\alpha_s|^2$. The above constraint says that, for all $z \notin S_f$,

$$\sum_{s\in\{0,1\}^n, |s|\leq k} (-1)^{s\cdot z} w(s) = 0.$$

Considering $w(s)$ as a function $w : \{0,1\}^n \to \mathbb{R}$ such that $w(s) = 0$ for $|s| > k$, in Fourier-analytic terminology the constraint says that

$$\widehat{w}(z) = 0 \text{ if } z \notin S_f,$$

i.e. that $\widehat{w}$ is only supported on the subspace S_f. This is equivalent to the constraint that

$$w(s) = \frac{1}{|S_f^\perp|}\sum_{t\in S_f^\perp} w(s+t) \text{ for all } s.$$

To see this, define the function $P_{S_f}(x) = [x \in S_f]$, and note that $\widehat{P_{S_f}}(t) = \frac{1}{|S_f^\perp|}[t \in S_f^\perp]$. Letting $*$ denote convolution over $\mathbb{Z}_2^n$ (i.e. $(f*g)(x) = \sum_y f(y)g(x+y)$), by Fourier duality we have

$$\begin{aligned} P_{S_f}w = w \quad &\Leftrightarrow \quad \widehat{P_{S_f}} * \widehat{w} = \widehat{w} \Leftrightarrow \sum_t w(s+t)\widehat{P_{S_f}}(t) = w(s) \text{ for all } s \\ &\Leftrightarrow \quad \frac{1}{|S_f^\perp|}\sum_{t\in S_f^\perp} w(s+t) = w(s) \text{ for all } s. \end{aligned}$$

Thus w is uniform on cosets of $S_f^\perp$. As w is not identically zero, there must be a coset $t + S_f^\perp$ such that every element $s \in t + S_f^\perp$ has Hamming weight at most k. If $S_f = 0$, then $S_f^\perp = \{0,1\}^n$ and hence has only one coset, which contains 1^n. Hence we must have $k = n$. More generally, we have that $Q_E^{na}(f)$ is the minimal k such that there exists a t satisfying $|s| \leq k$ for all $s \in t + S_f^\perp$. In other words, $Q_E^{na}(f)$ is the minimal k such that there exists a t satisfying $d(s,t) \leq k$ for all $s \in S_f^\perp$. $\square$

We observe from this proof that it is without loss of generality that any nonadaptive exact quantum algorithm can be described as picking a coset of $S_f^\perp$ and querying everything in that subset uniformly. Explicitly, we have the following algorithm.

1. Let $t \in \{0,1\}^n$ be a bit-string such that $d(t, S_f^\perp) = k$.
2. Produce the state of n qubits $\frac{1}{|S_f^\perp|^{1/2}} \sum_{s \in t+S_f^\perp} (-1)^{s \cdot x} |s\rangle$ at a cost of k queries to the oracle.
3. Perform Hadamards on every qubit of the resulting state and measure to get outcome $\tilde{x}$.
4. Output $f(\tilde{x})$.

One can easily verify that in fact $f(\tilde{x}) = f(x)$ with certainty. We note that this is reminiscent of an algorithm of van Dam [7] which learns x itself with bounded error using $n/2 + O(\sqrt{n})$ queries to the oracle. Here, we also compute a partial Fourier transform from which $f(x)$ can be determined, but our algorithm succeeds with certainty.

We now draw some corollaries from Theorem 7.

Corollary 8. *For any boolean function $f : \{0,1\}^n \to \{0,1\}$ such that f is not invariant under any translation, $Q_E^{na}(f) = n$.*

Proof. If f is not invariant under any translation, $S_f = \emptyset$ and hence $S_f^\perp = \{0,1\}^n$. For any bit-string $x \in \{0,1\}^n$, there exists a $y \in \{0,1\}^n$ such that $d(x,y) = n$. Hence $Q_E^{na}(f) = n$. □

One could also have observed this corollary by noting that $Q_E^{na}(f)$ depends only on S_f, and for the AND function (which has $Q_E(\text{AND}) = n$ [3]), $S_{\text{AND}} = \emptyset$. The corollary implies that only an exponentially small fraction of boolean functions f have $Q_E^{na}(f) < n$. One class of functions that do satisfy this is given by the following corollary.

Corollary 9. *For any boolean function $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = f(\bar{x})$ for all x, $Q_E^{na}(f) \leq n - 1$.*

Proof. The constraint $f(x) = f(\bar{x})$ for all x is equivalent to $f(x + 1^n) = f(x)$ for all x, so $\{0^n, 1^n\} \subseteq S_f$, implying $S_f^\perp \subseteq \{x : |x| \text{ even}\}$. If n is odd, then $d(0^n, S_f^\perp) \leq n - 1$ (as 1^n has odd Hamming weight, there is no even weight bit string distance n from 0^n). Similarly, if n is even, $d(10^{n-1}, S_f^\perp) \leq n - 1$. Hence $Q_E^{na}(f) \leq n - 1$. □

An explicit nonadaptive quantum algorithm achieving this query complexity proceeds as follows. Evaluate $y_k := x_1 \oplus x_k$, for $2 \leq k \leq n$, at a cost of $n-1$ queries in total, then output $f(0, y_2, \ldots, y_n)$. If $x_1 = 0$, this is simply $f(x)$, while if $x_1 = 1$, this is $f(\bar{x}) = f(x)$.

Corollary 10. *For any boolean function $f : \{0,1\}^n \to \{0,1\}$ that depends on all n input bits, $Q_E^{na}(f) \geq \lceil n/2 \rceil$.*

Proof. We need to show that, for any bit-string x, we can find an element $y \in S_f^\perp$ such that $d(x,y) \geq n/2$. As f depends on all its input bits, for all $i \in \{1, \ldots, n\}$, $e_i \notin S_f$. This implies that, for all $i \in \{1, \ldots, n\}$, there is at least one element of $S_f^\perp$ whose i'th bit is 1. Thus, if we pick an element $y \in S_f^\perp$ at random, each bit of y will be 0 or 1 with equal probability, so the expectation of $d(x,y)$ is exactly $n/2$. Therefore, $d(x, S_f^\perp) \geq n/2$. □

Corollary 10 was previously proven in [16] via a different method. We can also show that functions whose nonadaptive exact quantum query complexity is minimal are of very restricted form.

Corollary 11. *Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function that depends on all n input bits and such that $Q_E^{na}(f) = n/2$. Then there exists an x such that $d(x,y) = n/2$ for all $y \in S_f^\perp$.*

Proof. By the proof of Corollary 10, $\mathbb{E}_{y \in S_f^\perp} d(x,y) = n/2$. Thus, if $d(x,y) \leq n/2$ for *all* $y \in S_f^\perp$, we must have $d(x,y) = n/2$ for all $y \in S_f^\perp$. □

7.1 Symmetric boolean functions

It turns out that we can apply Theorem 7 to completely characterise the nonadaptive exact quantum query complexity of symmetric boolean functions, via the following quadrichotomy.

Theorem 12. *Let $f : \{0,1\}^n \to \{0,1\}$ be symmetric. Then exactly one of the following four possibilities is true.*

1. *f is constant and $Q_E^{na}(f) = 0$.*
2. *f is the PARITY function or its negation and $Q_E^{na}(f) = \lceil n/2 \rceil$.*
3. *f satisfies $f(x) = f(\bar{x})$ (but is not constant, the PARITY function or its negation) and $Q_E^{na}(f) = n - 1$.*
4. *f is none of the above and $Q_E^{na}(f) = n$.*

We will prove Theorem 12 using the following lemma, whose proof is given afterwards.

Lemma 13. *Let $f : \{0,1\}^n \to \{0,1\}$ be symmetric and satisfy $f(x) = f(x+a)$ for all $x \in \{0,1\}^n$, for some a with $1 \leq |a| \leq n-1$. Then, if $|a|$ is odd, f is constant. If $|a|$ is even, f is constant, PARITY or its negation.*

Proof of Theorem 12. First assume there is an a with $1 \leq |a| \leq n-1$ such that $f(x) = f(x+a)$ for all $x \in \{0,1\}^n$. Then, by Lemma 13, f is constant, PARITY or its negation. If f is constant then clearly $Q_E^{na}(f) = 0$. If f is PARITY or its negation, by the result [10] of Farhi et al. $Q_E^{na}(f) = \lceil n/2 \rceil$. On the other hand, if there is no a with $1 \leq |a| \leq n-1$ such that $f(x) = f(x+a)$ for all $x \in \{0,1\}^n$, but there is such an a with $|a| = n$, $f(x) = f(\bar{x})$ and by Corollary 9 $Q_E^{na}(f) = n - 1$. Finally, if there is no $a \neq 0^n$ such that $f(x) = f(x+a)$ for all $x \in \{0,1\}^n$, by Corollary 8 $Q_E^{na}(f) = n$. □

It will be convenient to prove Lemma 13 using Fourier analysis, based on the following well-known fact.

Fact 14. *For any $f : \{0,1\}^n \to \mathbb{R}$ and for any $a \in \{0,1\}^n$, if $f(x) = f(x+a)$ for all $x \in \{0,1\}^n$, then for all b such that $|a \wedge b|$ is odd, $\hat{f}(b) = 0$.*

Proof of Lemma 13. Note that, because f is symmetric, if $\hat{f}(s) = 0$ for some s with $|s| = k$, $\hat{f}(t) = 0$ for *all* t with $|t| = k$. Without loss of generality, assume a consists of j ones followed by $n-j$ zeroes (i.e. is of the form $1 \ldots 10 \ldots 0$). Consider the bit-string s which is 1 on the set $\{1, \ldots, 2k+1\}$ for some $0 \leq k \leq (j-1)/2$, and the set $\{n-\ell+1, \ldots, \ell\}$, for some $1 \leq \ell \leq j$. By Fact 14, for any such bit-string s, $\hat{f}(s) = 0$. By varying k and ℓ, we can vary $|s|$ arbitrarily between 1 and either n

(if $|a|$ is odd), or $n-1$ (if $|a|$ is even). Thus, if $|a|$ is odd, $\hat{f}(s) = 0$ for all $s \neq 0^n$, so f is constant. Otherwise, if $|a|$ is even, $\hat{f}(s) = 0$ for all $s \notin \{0^n, 1^n\}$. The only boolean functions satisfying this are constant functions, PARITY and its negation. □

8 Open problems

It is a very tempting conjecture that $Q_E(\mathrm{EXACT}_k) = \max\{k, n-k\}$. It is easy to see that the lower bound $Q_E(\mathrm{EXACT}_k) \geq \max\{k, n-k\}$ holds; by setting $\min\{k, n-k\}$ input bits to 0 we obtain a function equivalent to the AND function on $\ell := \max\{k, n-k\}$ bits, which has exact quantum query complexity ℓ. So it would suffice to prove the upper bound $Q_E(\mathrm{EXACT}_{n/2}) \leq n/2$ for all even n to prove this conjecture. To see this, note that for any ℓ, an algorithm for the function EXACT_ℓ on n bits gives an algorithm for EXACT_ℓ on $n' \geq n$ bits, simply by appending extra zero bits.

More generally, the problem of finding any boolean function f such that $Q_E(f) < D(f)/2$ still remains. We are hopeful that the numerical techniques used in this paper may prove helpful in resolving this question.

Acknowledgements

AM was supported by an EPSRC Postdoctoral Research Fellowship.

A Numerical results for functions on up to 6 bits

In this appendix we collate our numerical results concerning the optimal success probability achievable by quantum algorithms for all boolean functions on 4 input bits, and symmetric boolean functions on 5 and 6 input bits. We split the results into sections according to the number of bits on which the functions depend. Note that each section on functions of k bits does not include functions which only depend on fewer than k bits. In the following tables, entries are starred when there is a nonadaptive exact quantum algorithm using that number of queries (see Section 7). An entry "1" means that the SDP solver claims a solution with success probability greater than 0.999; note that this does not strictly speaking imply the existence of an *exact* algorithm using that number of queries. We use the notation $\mathrm{SYM}(c_0, \ldots, c_n)$ to mean the symmetric function f such that $f(x) = c_{|x|}$. In the tables of symmetric functions, we simply identify each function with the vector $(c_0, \ldots, c_n)$.

Note that for all non-constant symmetric f, the decision tree complexity $D(f) = n$, but this is not the case for $Q_E(f)$. For most functions f on 4 bits, $D(f)$ is easily verified to be 4 via polynomial degree arguments; we calculated $D(f)$ for the remaining functions f in an ad hoc fashion.

A.1 Functions of 4 bits

ID	Function	1 query	2 queries	3 queries	D(f)
1	$x_1 \wedge x_2 \wedge x_3 \wedge x_4$	0.735	0.962	0.996	4
6		0.654	0.931	1*	4
7		0.750	0.954	1	4
22		0.572	0.906	1	4
23		0.667	0.926	1	4
24	$x_1 \wedge \neg\mathrm{NAE}(\bar{x}_2, x_3, x_4)$	0.654	0.931	1*	4
25		0.640	0.961	1	4
27	$x_1 \wedge \mathrm{SEL}(x_4, x_2, x_3)$	0.667	0.965	1	3
30		0.600	0.956	1	4
31		0.718	0.970	1	4
61		0.643	0.976	1	4
105		0.500	0.900	1*	4
107		0.571	0.941	1	4
111		0.662	0.968	1*	4
126	$x_1 \wedge \mathrm{NAE}(x_2, x_3, x_4)$	0.667	0.947	1*	4
127		0.727	0.972	1	4
278	EXACT_3	0.529	0.884	1	4
279	Th_3	0.643	0.900	1	4
280		0.572	0.906	1	4
281		0.600	0.956	1	4
282		0.571	0.936	1	4
283		0.637	0.959	1	4
286		0.546	0.932	1	4
287		0.659	0.945	1	4
300		0.571	0.936	1	4
301		0.572	0.964	1	4
303	$\mathrm{SEL}(x_3, x_1 \wedge x_2, \mathrm{SEL}(x_4, x_1, x_2))$	0.644	0.966	1	3
316		0.562	0.962	1	4
317	$\mathrm{SEL}(x_3, x_1 \wedge x_2, \mathrm{SEL}(x_2, x_1, x_4))$	0.572	0.980	1	3
318		0.546	0.956	1	4
319		0.640	0.972	1	4
360		0.529	0.884	1	4
361		0.500	0.916	1	4
362		0.546	0.932	1	4
363		0.546	0.955	1	4
366		0.546	0.956	1	4
367		0.571	0.969	1	4
382		0.546	0.923	1	4
383		0.600	0.946	1	4
384	$\mathrm{NAE}(\bar{x}_1, x_2, x_3, x_4)$	0.800	0.980	1*	4
385		0.750	0.954	1	4
386		0.640	0.961	1	4
387		0.667	0.965	1	4
390		0.571	0.936	1	4
391		0.637	0.959	1	4

ID	Function	1 query	2 queries	3 queries	D(f)
393	$\mathrm{SEL}(x_3, x_1 \wedge \bar{x}_4, x_2 \wedge x_4)$	0.667	0.965	1	3
395		0.724	0.963	1	4
399		0.751	0.980	1	4
406		0.500	0.916	1	4
407		0.572	0.940	1	4
408		0.600	0.956	1	4
409		0.643	0.976	1	4
410		0.572	0.964	1	4
411		0.656	0.969	1	4
414		0.546	0.955	1	4
415		0.642	0.965	1	4
424		0.667	0.926	1	4
425		0.637	0.959	1	4
426		0.718	0.970	1	4
427	$\mathrm{SEL}(x_3, x_1 \wedge \bar{x}_4, \mathrm{SEL}(x_4, x_1, x_2))$	0.751	0.980	1	3
428		0.637	0.959	1	4
429	$\mathrm{SEL}(x_2, x_1 \wedge \bar{x}_4, \mathrm{SEL}(x_3, x_1, x_4))$	0.656	0.969	1	3
430		0.644	0.966	1	4
431		0.710	0.977	1	4
444		0.572	0.980	1	4
445		0.641	0.965	1	4
446		0.572	0.969	1	4
447		0.667	0.980	1	4
488		0.643	0.900	1	4
489		0.572	0.940	1	4
490		0.659	0.945	1	4
491		0.642	0.965	1	4
494		0.640	0.972	1	4
495	$\mathrm{SEL}(x_3, x_1, \mathrm{SEL}(x_4, x_1, x_2))$	0.667	0.980	1	3
510		0.600	0.946	1	4
829		0.563	0.975	1	4
854		0.598	0.955	1	4
855		0.714	0.969	1	4
856		0.572	0.964	1	4
857		0.579	0.961	1	4
858	$\mathrm{SEL}(x_1, x_2 \wedge x_3, x2 \oplus x_4)$	0.572	0.980	1	3
859		0.628	0.974	1	4
862		0.572	0.966	1	4
863	$\mathrm{SEL}(x_1, x_2 \wedge x_3, x2 \vee x_4)$	0.667	0.986	1	3
872		0.546	0.932	1	4
873		0.500	0.946	1	4
874		0.598	0.955	1	4
875		0.572	0.951	1	4
876		0.546	0.956	1	4
877		0.545	0.961	1	4
878		0.572	0.966	1	4
879		0.600	0.966	1	4
892		0.563	0.975	1	4

ID	Function	1 query	2 queries	3 queries	D(f)
893		0.571	0.966	1	4
894		0.572	0.947	1	4
961		0.718	0.970	1	4
965	$\mathrm{SEL}(x_2, x_1 \wedge \bar{x}_3, \mathrm{SEL}(x_1, x_3, x_4))$	0.751	0.980	1	3
966		0.644	0.966	1	4
967		0.710	0.977	1	4
980		0.659	0.945	1	4
981		0.714	0.969	1	4
982		0.572	0.951	1	4
983		0.661	0.965	1	4
984	$\mathrm{SEL}(x_1, x_2 \wedge x_3, \mathrm{SEL}(x_4, \bar{x}_3, \bar{x}_2))$	0.644	0.966	1	3
985		0.628	0.974	1	4
987	$\mathrm{SEL}(x_4, \mathrm{SEL}(x_3, x_1, x_2), \mathrm{SEL}(x_2, x_1, x_3))$	0.661	0.965	1	3
988		0.640	0.972	1	4
989	$\mathrm{SEL}(x_1, x_2 \wedge x_3, \bar{x}_3 \vee x_4)$	0.667	0.986	1	3
990		0.600	0.966	1	4
1632	$(x_1 \oplus x_2) \wedge (x_3 \oplus x_4)$	0.667	1*	1*	4
1633		0.562	0.962	1	4
1634		0.643	0.976	1	4
1635		0.572	0.980	1	4
1638		0.667	0.947	1*	4
1639		0.641	0.965	1	4
1641		0.500	0.936	1*	4
1643		0.561	0.966	1	4
1647	$\mathrm{MAJ}(x_1, x_2, x_3 \oplus x_4)$	0.667	1	1*	4
1650		0.656	0.969	1	4
1651		0.628	0.974	1	4
1654		0.641	0.965	1	4
1656		0.546	0.956	1	4
1657		0.500	0.964	1	4
1658		0.571	0.966	1	4
1659		0.571	0.962	1	4
1662		0.600	0.954	1	4
1680	$(x_1 \oplus x_2) \wedge (x_1 \oplus x_3 \oplus x_4)$	0.500	0.900	1*	4
1681		0.500	0.916	1	4
1683		0.500	0.946	1	4
1686		0.500	0.936	1*	4
1687		0.500	0.964	1	4
1695	$\mathrm{SEL}(x_3 \oplus x_4, x_1, x_2)$	0.500	1	1*	3
1712		0.571	0.941	1	4
1713		0.546	0.955	1	4
1714		0.572	0.940	1	4
1715		0.572	0.951	1	4
1716		0.546	0.955	1	4
1717		0.545	0.961	1	4
1718		0.561	0.966	1	4
1719	$\mathrm{SEL}(x_4, \mathrm{SEL}(x_2, x_1, x_3), \mathrm{SEL}(x_3, x_2, x_1))$	0.572	0.962	1	3
1721		0.500	0.964	1	4

ID	Function	1 query	2 queries	3 queries	D(f)
1725		0.529	0.955	1	4
1776		0.662	0.967	1*	4
1777		0.572	0.969	1	4
1778		0.642	0.965	1	4
1782	$\mathrm{SEL}(x_2, x_1, x_3 \oplus x_4)$	0.667	1	1*	3
1785	$x_1 \oplus (x_2 \wedge (x_3 \oplus x_4))$	0.500	1	1*	4
1910		0.600	0.954	1	4
1912		0.546	0.923	1	4
1913		0.529	0.955	1	4
1914		0.572	0.947	1	4
1918		0.572	0.922	1	4
1968		0.662	0.968	1*	4
1969		0.642	0.965	1	4
1972		0.572	0.969	1	4
1973		0.600	0.966	1	4
1974		0.572	0.962	1	4
1980	$\mathrm{SEL}(x_3, \mathrm{SEL}(x_4, x_1, x_2), x_1 \oplus x_2)$	0.571	0.966	1	3
2016	$\mathrm{SEL}(x_1, x_2 \wedge (x_3 \vee x_4), \bar{x}_2 \wedge (\bar{x}_3 \vee \bar{x}_4))$	0.773	1	1*	4
2017		0.640	0.972	1	4
2018		0.710	0.977	1	4
2019		0.667	0.986	1	4
2022	$\mathrm{SEL}(x_3, \mathrm{SEL}(x_2, x_1, x_4), \mathrm{SEL}(x_1, x_2, \bar{x}_4))$	0.661	0.965	1	3
2025		0.571	0.966	1	4
2032		0.727	0.971	1	4
2033		0.667	0.980	1	4
2034	$\mathrm{SEL}(x_2, x_1, \mathrm{SEL}(x_4, x_3, \bar{x}_1))$	0.667	0.980	1	3
2040		0.600	0.946	1	4
5736	EXACT_2	0.572	1	1*	4
5737	SYM(0,0,1,0,1)	0.500	0.962	1	4
5738		0.563	0.975	1	4
5739		0.500	0.980	1	4
5742		0.572	0.947	1	4
5758		0.572	0.922	1	4
5761		0.500	0.860	1	4
5763		0.500	0.907	1	4
5766		0.500	0.936	1*	4
5767		0.500	0.933	1	4
5769		0.500	0.907	1	4
5771		0.500	0.946	1	4
5774		0.561	0.966	1	4
5782		0.500	0.962	1	4
5783		0.500	0.954	1	4
5784		0.500	0.946	1	4
5785		0.500	0.933	1	4
5786		0.500	0.964	1	4
5787		0.500	0.955	1	4
5790	$\mathrm{SEL}(x_3, \mathrm{SEL}(x_4, x_1, x_2), x_2 \oplus x_4)$	0.500	0.980	1	3
5801		0.500	0.933	1	4

ID	Function	1 query	2 queries	3 queries	D(f)
5804		0.545	0.961	1	4
5805		0.500	0.955	1	4
5820		0.529	0.955	1	4
5865		0.500	0.954	1	4
6014	SYM(0,0,1,1,0)	0.600	0.874	1	4
6030	$\mathrm{SEL}(x_3, \mathrm{SEL}(x_4, x_1, x_2), \mathrm{SEL}(x_4, x_2, \bar{x}_1))$	0.667	1	1*	3
6038		0.500	0.980	1	4
6040		0.572	0.951	1	4
6042	$\mathrm{SEL}(x_4, \mathrm{SEL}(x_3, x_1, x_2), \mathrm{SEL}(x_2, x_3, \bar{x}_1))$	0.572	0.962	1	3
6060		0.600	0.966	1	4
6120	$x_1 \oplus \mathrm{MAJ}(x_2, x_3, x_4)$	0.667	1	1*	4
6375	$x_1 \oplus \neg\mathrm{NAE}(\bar{x}_2, x_3, x_4)$	0.500	0.900	1*	4
6625		0.500	0.946	1	4
6627		0.500	0.955	1	4
6630		0.500	0.954	1	4
7128	Sorted input bits [1]	0.854	1	1*	3
7140	$x_1 \oplus \mathrm{SEL}(x_4, x_2, x_3)$	0.500	1	1*	3
7905		0.500	0.900	1*	4
27030	PARITY	0.500	1*	1*	4

A.2 Symmetric functions of 5 bits

Function	1 query	2 queries	3 queries	4 queries
(0,0,0,0,0,1)	0.693	0.925	0.988	0.999
(0,0,0,0,1,0)	0.516	0.761	0.972	1
(0,0,0,0,1,1)	0.640	0.798	0.974	1
(0,0,0,1,0,0)	0.530	0.616	1	1
(0,0,0,1,0,1)	0.500	0.593	0.995	1
(0,0,0,1,1,0)	0.546	0.758	1	1
(0,0,0,1,1,1)	0.600	0.728	1	1
(0,0,1,0,0,1)	0.500	0.640	0.988	1
(0,0,1,0,1,0)	0.500	0.517	1	1
(0,0,1,0,1,1)	0.500	0.534	1	1
(0,0,1,1,0,0)	0.600	0.874	1	1*
(0,0,1,1,0,1)	0.500	0.856	0.998	1
(0,0,1,1,1,0)	0.616	0.762	0.969	1
(0,1,0,0,0,1)	0.500	0.728	0.967	1
(0,1,0,0,1,0)	0.500	0.860	1	1*
(0,1,0,1,0,1)	0.500	0.500	1*	1*
(0,1,0,1,1,0)	0.500	0.616	0.998	1
(0,1,1,0,0,1)	0.500	0.784	0.998	1
(0,1,1,1,1,0)	0.736	0.962	0.996	1*

A.3 Symmetric functions of 6 bits

Function	1 query	2 queries	3 queries	4 queries	5 queries
(0,0,0,0,0,0,1)	0.663	0.900	0.980	0.997	0.9999
(0,0,0,0,0,1,0)	0.511	0.684	0.940	0.993	1
(0,0,0,0,0,1,1)	0.640	0.738	0.946	0.993	1
(0,0,0,0,1,0,0)	0.516	0.572	0.878	1	1
(0,0,0,0,1,0,1)	0.500	0.541	0.875	0.999	1
(0,0,0,0,1,1,0)	0.527	0.751	0.904	1	1
(0,0,0,0,1,1,1)	0.589	0.710	0.901	1	1
(0,0,0,1,0,0,0)	0.530	0.616	1	1	1*
(0,0,0,1,0,0,1)	0.500	0.614	0.980	0.997	1
(0,0,0,1,0,1,0)	0.500	0.504	0.946	1	1
(0,0,0,1,0,1,1)	0.500	0.525	0.952	1	1
(0,0,0,1,1,0,0)	0.546	0.667	0.864	1	1
(0,0,0,1,1,0,1)	0.500	0.625	0.860	1	1
(0,0,0,1,1,1,0)	0.556	0.721	0.905	1	1
(0,0,1,0,0,0,1)	0.500	0.583	0.882	0.997	1
(0,0,1,0,0,1,0)	0.500	0.527	0.839	1	1
(0,0,1,0,0,1,0)	0.500	0.541	0.843	1	1
(0,0,1,0,1,0,0)	0.500	0.517	1	1	1*
(0,0,1,0,1,0,1)	0.500	0.500	0.985	1	1
(0,0,1,0,1,1,0)	0.500	0.520	0.940	1	1
(0,0,1,1,0,0,1)	0.500	0.712	0.867	1	1
(0,0,1,1,0,1,0)	0.500	0.513	0.840	1	1
(0,0,1,1,1,0,0)	0.616	0.762	0.969	1	1*
(0,0,1,1,1,0,1)	0.500	0.722	0.965	1	1
(0,0,1,1,1,1,0)	0.625	0.702	0.939	0.992	1
(0,1,0,0,0,0,1)	0.500	0.652	0.934	0.992	1
(0,1,0,0,0,1,0)	0.500	0.728	0.967	1	1*
(0,1,0,0,1,0,1)	0.500	0.500	0.836	1	1
(0,1,0,0,1,1,0)	0.500	0.667	0.863	1	1
(0,1,0,1,0,0,1)	0.500	0.500	0.934	1	1
(0,1,0,1,0,1,0)	0.500	0.500	1*	1*	1*
(0,1,0,1,1,1,0)	0.500	0.553	0.880	0.999	1
(0,1,1,0,0,0,1)	0.500	0.758	0.908	1	1
(0,1,1,0,1,1,0)	0.500	0.640	0.988	1	1*
(0,1,1,1,1,1,0)	0.693	0.925	0.988	0.999	1*

B Source code

The following is an example of how the CVX package [11] can be used to determine quantum query complexity. In this case, we calculate the minimal error probability over all quantum algorithms using 2 queries to compute some function $f : \{0,1\}^3 \to \{0,1\}$ (given as a column vector).

```
cvx_begin
```

```
  cvx_precision best;

  % variables m_i^j : 0 <= i <= n, 0 <= j <= t-1
  variable m00(8,8) symmetric; variable m10(8,8) symmetric;
  variable m20(8,8) symmetric; variable m30(8,8) symmetric;
  variable m01(8,8) symmetric; variable m11(8,8) symmetric;
  variable m21(8,8) symmetric; variable m31(8,8) symmetric;

  variable g0(8,8) symmetric; variable g1(8,8) symmetric;

  variable epss;

  minimise( epss );

  subject to

    % Input condition.
    m00 + m10 + m20 + m30 == ones(8,8);

    % Running conditions (between 1 and t-1).
    m01 + m11 + m21 + m31 == E0 .* m00 + E1 .* m10 + E2 .* m20 + E3 .* m30;

    % Output matches last but one query.
    g0 + g1 == E0 .* m01 + E1 .* m11 + E2 .* m21 + E3 .* m31;

    % Output constraints.
    diag(g0) >= (1-epss)*(1-f);
    diag(g1) >= (1-epss)*f;

    % Semidefinite constraints.
    m00 == semidefinite(8); m10 == semidefinite(8);
    m20 == semidefinite(8); m30 == semidefinite(8);
    m01 == semidefinite(8); m11 == semidefinite(8);
    m21 == semidefinite(8); m31 == semidefinite(8);

    g0 == semidefinite(8); g1 == semidefinite(8);

cvx_end
```

References

[1] A. Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Syst. Sci.*, 72(2):220–238, 2006. quant-ph/0305028.

[2] H. Barnum, M. Saks, and M. Szegedy. Quantum query complexity and semi-definite programming. In *Proc. 18th Annual IEEE Conf. Computational Complexity*, pages 179–193, 2003.

[3] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001. quant-ph/9802049.

[4] G. Brassard and P. Høyer. An exact quantum polynomial-time algorithm for Simon's problem. In *Theory of Computing and Systems, Proceedings of the Fifth Israeli Symposium on*, pages 12–23, 1997. quant-ph/9704027.

[5] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proc. R. Soc. Lond. A*, 454(1969):339–354, 1998. quant-ph/9708016.

[6] G. Cohen, M. Karpovsky, H. Mattson, Jr., and J. Schatz. Covering radius – survey and recent results. *IEEE Trans. Inform. Theory*, 31(3):328–343, 1985.

[7] W. van Dam. Quantum oracle interrogation: Getting all information for almost half the price. In *Proc. 39th Annual Symp. Foundations of Computer Science*, pages 362–367. IEEE, 1998. quant-ph/9805006.

[8] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proc. Roy. Soc. London Ser. A*, 439(1907):553–558, 1992.

[9] Alina Dubrovska and Taisija Mischenko-Slatenkova. Computing boolean functions: Exact quantum query algorithms and low degree polynomials, 2006. quant-ph/0607022.

[10] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. A limit on the speed of quantum computation in determining parity. *Phys. Rev. Lett.*, 81:5442–5444, 1998. quant-ph/9802045.

[11] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. http://cvxr.com/cvx, April 2011.

[12] T. Hayes, S. Kutin, and D. van Melkebeek. The quantum black-box complexity of majority. *Algorithmica*, 34(4):480–501, 2002. quant-ph/0109101.

[13] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[14] P. Høyer and R. Špalek. Lower bounds on quantum query complexity. *Bulletin of the European Association for Theoretical Computer Science*, 87:78–103, 2005. quant-ph/0509153.

[15] G. Midrijānis. Exact quantum query complexity for total Boolean functions, 2004. quant-ph/0403168.

[16] A. Montanaro. Nonadaptive quantum query complexity. *Information Processing Letters*, 110(24):1110–1113, 2010. arXiv:1001.0018.

[17] B. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *Proc. 50th Annual Symp. Foundations of Computer Science*, pages 544–551, 2009. arXiv:0904.2759.

[18] B. Reichardt and R. Špalek. Span-program-based quantum algorithm for evaluating formulas. In *Proc. 40th Annual ACM Symp. Theory of Computing*, pages 103–112, 2008. arXiv:0710.2630.

[19] Alina Vasilieva. Quantum query algorithm constructions for computing AND, OR and MAJORITY boolean functions, 2007. arXiv:0710.5592.

[20] Alina Vasilieva. Exact quantum query algorithm for error detection code verification, 2009. arXiv:0904.3660.