Quantum Information Processing (2019) 18:177
https://doi.org/10.1007/s11128-019-2297-3

Evaluation of exact quantum query complexities by semidefinite programming

Kıvanç Uyanık[1]

Received: 22 December 2018 / Accepted: 20 April 2019 / Published online: 26 April 2019

Abstract
One of the difficult tasks in quantum computation is inventing efficient exact quantum algorithms, which are the quantum algorithms that output the correct answer with certainty on any input. We improve and generalize the semidefinite programming (SDP) method of Montanaro et al. (Algorithmica 71:775–796, 2015) in order to evaluate exact quantum query complexities of partial functions. We present a more systematical approach to achieve the "inspired" result by Montanaro et al. for the function EXACT_2^4, which is the Boolean function of 4 bits that output only when 2 of the input bits are equal to 1. The same approach also allows us to reduce the size of the ancilla space used by the algorithms that evaluate symmetric functions like EXACT_3^6. We employ the generalized SDP to verify the complexities of the earliest and best known quantum algorithms in the literature, namely, Deutsch–Jozsa and Grover algorithms for a small number of input bits. We utilized the method to solve the weight decision problem of bit strings with lengths up to 10 bits and observed that the generalized SDP gives better exact quantum query complexities than the known methods. Finally, we test the method on some selected functions and demonstrate that they all exhibit quantum speedup.

Keywords Quantum algorithms · Semidefinite programming · Exact query complexity

1 Introduction

Query model of quantum computation is the quantum version of the classical decision tree model. Unlike the Turing machine model where we can only make conjectures about the resources needed for the computation, simplicity of this model allows us to go beyond that and prove such conjectures. In this model, the complexity is given as

✉ Kıvanç Uyanık
kivancuyanik@iyte.edu.tr

[1] Department of Physics, İzmir Institute of Technology, 35430 İzmir, Turkey

the minimum number of queries to a black box implementation of a finite function $f : S \subseteq \{0,1\}^N \to \{0,1\}^M$. In most of the cases we have $M = 1$, and in that case f is called a decision function (see [1] for more details). We can view f as a map that reveals a property of its input $x = x_1 x_2 \dots x_N$ which can be thought of a string of N independent bits. Most of the well-known quantum algorithms can be naturally represented in the query complexity model, such as Deutsch–Jozsa algorithm [2], Grover's algorithm [3], quantum walks [4].

The quantum query model is either in the bounded-error setting, in which, the algorithm is allowed to have a limited probability of error or in the exact setting, meaning that the algorithm is always expected to give the correct output. Note that, the term "exact quantum algorithm" refers to the fact that quantum algorithm itself runs with zero probability of error, not to be confused with the task of computing the query complexity of the quantum algorithm. The more general bounded-error version is relatively well established. For total functions, where the domain of the function f is equal to the whole set $S = \{0,1\}^N$, the quantum lower bound was proved to be $O(\sqrt{N})$ [5] for the unstructured search algorithm [3]. Allowing for partial functions (i.e., choosing a domain which is a strict subset $S \subsetneq \{0,1\}^N$) can even lead to exponential separations [6,7]. In contrast, the development of the techniques for designing exact quantum algorithms fell behind in comparison with the bounded-error scheme. For example, for quite a long time, the best separation between classical and exact quantum algorithms for total Boolean functions was known to be by a factor of 2 [8], until Ambainis showed that there exists a superlinear advantage of exact quantum algorithms [9,10].

There are two main methods that proved to be useful for finding lower bounds on quantum query complexities; the polynomial method by Beals et al. [11] and the adversarial method by Ambainis [12] originated from the hybrid method by Bennett et al. [5] and Vazirani [13]. The polynomial method is based on the idea of showing that an algorithm A computing some Boolean function f can be represented or approximated by some real-valued polynomial p_A. Adversarial methods are based on the observation that the calls to the oracle that represents the function f can give us only limited amount of information about that function. We refer to the survey of Høyer and Spalek for more on these methods [14]. The method of Barnum et al. [15], which exactly characterizes the quantum query complexity can be considered as one of the variants of the adversary method. Although it is an effective method, it has its own disadvantages: since all adversary methods are essentially equivalent [16], it would not give lower bounds better than the ones that has been found from other methods. Also, it is hard to utilize the method for analytical proofs. Nevertheless, Montanaro et al. showed that the method can be quite useful. Being inspired by the numerical results, they have obtained by implementing Barnum et al.'s semidefinite programming method [15], they were able to design a slightly more general form of the Deutsch–Jozsa algorithm [17]. Thanks to their (numerical) work, we also know that AND_3 and AND_4 are the only total Boolean functions that show no quantum speedups, up to isomorphisms.

There is a growing interest in the exact computation of Boolean functions. Exact quantum algorithms have potential applications in communication complexity [9,18], finite automata [19] and cryptography [20–22]. In Montanaro et al.'s paper, they only consider the exact query complexities of the total Boolean functions up to a few

number of input bits. In this work, we shift our focus to the promise problems (partial functions) and apply the same method to some of the well-known problems in quantum computation, such as Deutsch–Jozsa, and Grover's problems. Our primary intent in doing so is to show the power and generalizability of the SDP method. For this reason, we start with well-established algorithms but later on we extend our work to the problems that are relatively less studied. In this category, first, we apply the method to the weight decision problem of Boolean functions of several bits and compare our results with previous works and we continue with numerically analyzing the exact query complexity of some selected functions. We also show that with some minor improvements to the SDP method, it can output the algorithm for the evaluation of EXACT_2^4 problem without guessing any solutions. That enabled us to significantly reduce the number of ancilla qubits needed to evaluate some functions that exhibit certain symmetries.

The paper is organized as follows. In Sect. 2, we give the necessary definitions and establish the formal structure needed for the rest of the paper. In Sect. 3, we briefly review the SDP method of Barnum et al. and continue in Sect. 4 with mainly Montanaro et al.'s implementation of the method. In Sect. 5, we present our first modification to the SDP method which enabled us to numerically evaluate the compact quantum algorithm in [17] for the function EXACT_2^4. We continue in Sect. 6, by presenting our generalization to the SDP implementation which allowed us to easily verify the query complexities of some well-known problems. In Sect. 6, we also apply the generalized SDP to the weight decision problem and evaluate exact query complexities of some selected functions such as And–Or trees and maximum deviation. In Sect. 7, we briefly conclude our results and discuss some open problems.

2 Definitions and notation

We start with following a similar notation to Barnum et al.'s [15]. Consider the finite function $f : S \longrightarrow T$, where the input domain of the function is $S \subseteq \chi^n$ and χ and T are finite sets with n, a positive integer. Throughout this paper, all the functions f will be Boolean functions, that is, the variable domain is limited to $\chi = \{0, 1\}$. f is called a total function if $S = \chi^n$, otherwise it is called a partial function(or evaluation of f is called a promise problem). f is called a decision function when $T = \{0, 1\}$. An arbitrary element in the domain of a Boolean function can be described by an N-bit string $x = x_1 x_2 \ldots x_N$ (the ith bit of x is denoted by x_i), therefore evaluation of a Boolean function f can be thought as determining some property of the bit string x. A very common example is the $\text{PARITY}(x) \triangleq |x| \,(\text{mod } 2)$ function, where $|x|$ denotes the Hamming weight of the bit string x. Another Boolean function we will frequently use is $\text{EXACT}_k^n(x)$, which outputs 1 if and only if $|x| = k$ and n is the length of the input bit string x.

In the conventional quantum computational models, the quantum information is stored in the memory registers. Like the classical registers, each quantum register can have a finite number of values. We associate a d-dimensional complex vector space $\mathcal{H}_R$ to a register R that has d possible values. Most of the time we work in the computational basis, which means that we label the basis vectors as kets $|i\rangle$,

where $i \in \{0, 1, 2, \ldots, d-1\}$. Actually, this is also a short notation: we write an arbitrary vector $|v\rangle$ as the short for the tensor product of two-dimensional vectors $|v\rangle = |v_{N-1}\rangle \otimes |v_{N-2}\rangle \otimes \ldots \otimes |v_0\rangle$ where each vector $|v_i\rangle$ holds the value of the i'th bit of $v = v_{N-1}v_{N-2}\ldots v_0$. We can group as many registers as we need to form a virtual register. Even the entire memory is formed in this way in most of the cases. Let a virtual register R be composed of m independent registers $R = R_1, \ldots, R_m$. Naturally, the associated Hilbert space is the tensor product space $\mathcal{H}_R = \mathcal{H}_{R_1} \otimes \ldots \otimes \mathcal{H}_{R_m}$ which has the tensor product basis $|i_1 i_2 \ldots i_m\rangle$, which is the short notation of $|i_1\rangle\,|i_2\rangle \ldots |i_m\rangle = |i_1\rangle \otimes |i_2\rangle \otimes \ldots \otimes |i_m\rangle$.

In the quantum query model, we need three registers. We start with the input register. It holds the value of the input bit string $x = x_1, \ldots, x_n$. Next, we have the query register. It can have values between 0 and n for a bit string of length n. Finally, we may need an ancillary work register whose dimension has no upper limit. For the input x, the query i and the value of the work register w, we denote the standard basis vectors of associated vector space $\mathcal{H}$ with $|x, i, w\rangle = |x\rangle \otimes |i\rangle \otimes |w\rangle$. During the computation, the memory can be in the arbitrary superposition state

$$|\Psi\rangle = \sum_{x,i,w} \alpha_{x,i,w} |x, i, w\rangle \quad ,$$

where the amplitude squares of the complex coefficients $\alpha_{x,i,w}$ add up to 1. It is customary to define the query and working memory registers as accessible memory as its reasons will shortly become clear.

The other crucial part of the computational model are the unitary operators that act on $\mathcal{H}$. An algorithm with t queries to the individual bits of the input x is defined with an oracle operator O and a sequence of $t+1$ unitary operators U_t. The intermediary unitaries U_t do not depend on the input and act only on the query and work registers. Therefore, our notation is in fact a short notation for $U_t = \mathbb{1}_X \otimes \left(\tilde{U}_t\right)_{IW}$ where $\mathbb{1}_X$ denotes the identity operator on the inaccessible input space. The last part to the algorithm are the orthogonal projection operators P_z, which correspond to the final measurement step. The execution of the algorithm can be divided into three parts:

- *initialization*, where the state of the computer is initialized to $|\Psi_i\rangle = |x\rangle_X\,|0\rangle_I\,|0\rangle_W = |x\rangle\,|0\rangle\,|0\rangle$,
- *evolution*, where the unitary sequence $U_0, U_1, \ldots, U_t$ and oracle calls O are alternatingly applied to the current state of the computer and
- *measurement*, where a single measurement specified by the complete set of orthogonal projectors P_z is made on the final state of the computer $\left|\Psi_f\right\rangle = U_t O U_{t-1} O \cdots U_1 O U_0\,|0\rangle\,|0, 0\rangle$ and output the result z.

Before we proceed, let us take a closer look at the oracle operator O. We model the query to any input x with the operator O whose action on the standard basis is given as

$$O\,|x, i, w\rangle = (-1)^{x_i}\,|x, i, w\rangle \quad . \tag{1}$$

Let us consider the $i > 0$ and $i = 0$ cases separately. When $i > 0$, depending on the value of i, we access the ith bit x_i of the input x and only the state $|x, i, w\rangle$ pick up a phase accordingly. On the other hand, when $i = 0$, no phase is introduced regardless of x. Another way of thinking the $i = 0$ case is attaching an $x_0 = 0$ as a zeroth element to any given string x. This way, the oracle operator can be switched off and become a "null query" when necessary. Null queries are needed to preserve compatibility to the alternative models of query complexity [23]. Notice that neither the oracle O, nor the unitaries U_j change the state of the input register. Therefore, the state of the memory can be given in the form $|x\rangle\,|\psi_x\rangle$ throughout the computation. Alternatively, instead of defining the oracle as in (1), for each input x, we can specify the algorithm with a corresponding oracle O_x, whose action can now be written as $O_x\,|i, w\rangle = (-1)^{x_i}\,|i, w\rangle$. This notation is sometimes more useful. The completeness and orthogonality of projectors P_z and normalization of the total state of the memory at all times ensures that the total probability of obtaining possible outputs is $\sum_{z\in T}\left|\langle\Psi_f P_z | P_z\Psi_f\rangle\right|^2 = 1$.

Consider a quantum query algorithm A that computes a given a function $f : S \longrightarrow T$. We say that A computes f within an error ε if A outputs $f(x) = z$ with a success probability greater than $1 - \varepsilon$ for all possible inputs x:

$$\left\| P_z\Psi_f \right\|^2 \geq 1 - \varepsilon \ .$$

Among all the possible quantum algorithms, if the minimum number of calls to the oracle is t while the error is still within ε, then we say that the quantum query complexity of the function f is $QQC_\varepsilon(f) = t$. If the error is zero, i.e., $\varepsilon = 0$, the algorithm is called an exact quantum algorithm and we say that the exact query complexity of f is $QQC_0(f) = t$.

Let us have a set of vectors $\{|\phi_1\rangle, |\phi_2\rangle, \ldots, |\phi_m\rangle\}$ which are not necessarily orthogonal lying in the Hilbert space $\mathcal{H}$. We define the Gram matrix M with the following entries:

$$(M)_{ij} \triangleq \langle\phi_i|\phi_j\rangle \ ,$$

where we denote the matrix elements of a matrix B with $(B)_{ij}$. It can be shown that Gram matrices are Hermitian, positive semidefinite and $\text{rank}\,(M) = \dim\,(\text{span}\,\{|\phi_1\rangle, |\phi_2\rangle, \ldots, |\phi_m\rangle\})$. Conversely, every positive semidefinite matrix M is a Gram matrix of a set of vectors that lies in a Hilbert space $\mathcal{H}$ of dimension at least $\text{rank}\,(M)$ [24].

3 Semidefinite programs for calculating query complexities

In this section, we review the method of Barnum et al. [15], specifically how quantum query algorithms can be represented by semidefinite programs. We denote a t-step quantum query algorithm that evaluates a partial function $f : S \to T$ within error ε with $QQA_{\varepsilon,t}\,(f)$, where t is a positive integer and $0 \leq \varepsilon < 1/2$. Let us fix the input $x \in S$. The state of the memory at step j can be written as the sum

$$\left|\psi_x^{(j)}\right\rangle = \sum_{i=0}^{n} |i\rangle \left|\psi_{x,i}^{(j)}\right\rangle \ , \tag{2}$$

since the accessible space can be decomposed naturally into mutually orthogonal subspaces $(|i\rangle \otimes \mathcal{H}_w, 0 \leq i \leq n)$. The states defined in (2) generate a sequence which are interlinked by application of two successive unitaries: the oracle O_x that corresponds to the input x, and the input independent unitary operator U_j. The final step of the algorithm (measurement) is specified by the orthonormal projections P_z, each of which corresponds to a different output $z \in T$.

We define the following real symmetric matrices $M^{(j)}$, $M_i^{(j)}$ and Γ_z with the following matrix elements

$$\begin{aligned}
\left(M^{(j)}\right)_{xy} &\triangleq \left\langle \psi_x^{(j)} \middle| \psi_y^{(j)} \right\rangle \ , \\
\left(M_i^{(j)}\right)_{xy} &\triangleq \left\langle \psi_{x,i}^{(j)} \middle| \psi_{y,i}^{(j)} \right\rangle \ , \\
(\Gamma_z)_{xy} &\triangleq \left\langle \psi_x^{(j)} P_z^{\dagger} \middle| P_z \psi_y^{(j)} \right\rangle \ .
\end{aligned}$$

To express the semidefinite program in a compact manner, we continue using the notation in [15] by defining the fixed matrices

$$\begin{aligned}
(E)_{xy} &\triangleq 1 \ , \\
(E_i)_{xy} &\triangleq (-1)^{x_i + y_i} \ , \\
(\Delta_z)_{xx} &\triangleq \delta_{f(x),z} \ .
\end{aligned}$$

We define the semidefinite program $SDP\,(f, t, \varepsilon)$ for a function f, an integer t and a nonnegative real number $\varepsilon : (0 \leq \varepsilon < 1/2)$, with a system of $s = \dim S$ dimensional real symmetric matrices $M^{(t)}$, $M_i^{(j)}$ and Γ_z $(0 \leq j \leq t-1$ and $0 \leq i \leq n)$ which satisfy the following conditions:

$$\sum_{i=0}^{n} M_i^{(0)} = E \ , \tag{3}$$

$$\sum_{i=0}^{n} M_i^{(j)} = \sum_{i=0}^{n} E_i * M_i^{(j-1)} \quad \text{for } 1 \leq j \leq t-1 \ , \tag{4}$$

$$M^{(t)} = \sum_{i=0}^{n} E_i * M_i^{(t-1)} \ , \tag{5}$$

$$M^{(t)} = \sum_{z \in T}^{n} \Gamma_z \ , \tag{6}$$

$$\Delta_z * \Gamma_z \geq (1 - \varepsilon)\, \Delta_z \ , \tag{7}$$

where $*$ denotes the Hadamard (entrywise) product of matrices. The main result of [15] is summarized in Theorem 1:

Theorem 1 *Let $f : S \subseteq \{0,1\}^N \to \{0,1\}^M$ be a partial function. There exists a t-query $QQA_{\varepsilon,t}(f)$ that computes function f within error ε if and only if $SDP(f,t,\varepsilon)$ with conditions (3), (4), (5), (6) and (7) is feasible. Moreover, let $r = \max\left(rank\ M_i^{(j)}\right)$ be the maximum of the ranks over the indices $(0 \le j \le t-1)$ and $(0 \le i \le n)$. Then, the number of qubits needed for the working space of the memory is at most $\lceil \log |r| \rceil$.*

4 Exact quantum query complexity

The method of Barnum et al. which is summarized in Theorem 1 provides a way of showing the existence of a quantum algorithm that makes a fixed number of queries to compute a given (mostly a decision) function. However, sometimes we actually want the algorithm itself, i.e., the unitaries in between. Montanaro et al. offers a method to find the unitary operators U_j thus a quantum algorithm for the given parameters can be explicitly constructed [17]. Also, they modified the feasibility condition of the SDP to a minimization condition on the error parameter ε. By checking whether this minimized error is below a certain threshold or not, they were able to obtain the exact quantum query complexity of the problem. Here, we review their prescription which generates query algorithms out of the solutions of SDPs.

Suppose we are given a set of matrices $M_i^{(j)}$, $M^{(t)}$, and Γ_z which satisfies the constraints (3), (4), (5), (6) and (7). Let us not restrict the dimension of $\mathcal{H}_w$ and take it as large as dim (S) for the moment. For input x, we represent the state of the memory just after the jth oracle call with the kets $\left|\phi_x^{(j)}\right\rangle$. In a sense, $\left|\phi_x^{(j)}\right\rangle$'s define half of the intermediate states of computation. We define the other half by $\left|\psi_x^{(j)}\right\rangle$, which denote the state of the computer just after the application of the unitary U_j and just before the $j+1$' st oracle call. The flow is shown in (8) for $0 \le j \le t-1$:

$$\cdots \xrightarrow{O_x} \left|\phi_x^{(j)}\right\rangle \xrightarrow{U_j} \left|\psi_x^{(j)}\right\rangle \xrightarrow{O_x} \left|\phi_x^{(j+1)}\right\rangle \xrightarrow{U_{j+1}} \cdots . \tag{8}$$

Now, we define $\left|\psi_i^{(j)}\right\rangle$ as the sum $\sum_{i=0}^{n} |i\rangle \left|\psi_{x,i}^{(j)}\right\rangle$ and taking the action of the oracle O_x into account , we can use the same states to define $\left|\phi_x^{(j)}\right\rangle$ as

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

Finally, set $\left|\phi_x^{(0)}\right\rangle = |0\rangle |0\rangle$ and the sequence of states that we observe during the computation depends only on the unspecified states $\left|\psi_{x,i}^{(j)}\right\rangle$ of the work space. Now, there are many alternatives for $\left|\psi_{x,i}^{(j)}\right\rangle$ that result in a valid quantum query algorithm; however, we first start with the simple choice of [17] by setting

$$\left|\psi_{x,i}^{(j)}\right\rangle = \sqrt{M_i^{(j)}}\,|x\rangle \quad , \tag{9}$$

where $\sqrt{M_i^{(j)}}$ are the positive semidefinite square root of $M_i^{(j)}$.

There remains a few steps before we explicitly define an exact quantum query algorithm (QQA). First, we have to show that the choice in (9) would produce at least one series of unitaries U_j that defines the algorithm. Also, we have to give an explicit method describing how to obtain them. Let us define the matrices

$$Y^{(j)} \triangleq \sum_{x\in S} \left|\psi_x^{(j)}\right\rangle \langle x| \text{ and } F^{(j)} \triangleq \sum_{x\in S} \left|\phi_x^{(j)}\right\rangle \langle x|$$

where each column of matrices $Y^{(j)}\left(F^{(j)}\right)$ is one of the vectors $\left|\psi_x^{(j)}\right\rangle\left(\left|\phi_x^{(j)}\right\rangle\right)$. We will use the following lemma [24]:

Lemma 1 *Let $\{|\psi_i\rangle\}$ and $\left\{\left|\phi_j\right\rangle\right\}$ be two sequences of m vectors of the same dimension. Define $Y^{(j)} \triangleq \sum_{x\in S} \left|\psi_x^{(j)}\right\rangle \langle x|$ and $F^{(j)} \triangleq \sum_{x\in S} \left|\phi_x^{(j)}\right\rangle \langle x|$. Then, there exists a unitary U such that $U\,|\phi_i\rangle = |\psi_i\rangle$ for all i if and only if $Y^\dagger Y = F^\dagger F$. The unitary U is constructed as follows. Let $Y = V\sqrt{Y^\dagger Y}$ and $F = W\sqrt{F^\dagger F}$ be the polar decompositions of Y and F, respectively. Complete the angular parts of decompositions V and W to unitary matrices V' and W' . Finally, define $U \triangleq V'\left(W'\right)^\dagger$.*

Lemma 1 guarantees the existence of a unitary sequence $U_0, \ldots, U_{t-1}$ of (8) for all x if and only if

$$\left(Y^{(j)}\right)^\dagger Y^{(j)} = \left(F^{(j)}\right)^\dagger F^{(j)} \quad . \tag{10}$$

And the choice in (9) allows us to write the LHS of 10 each $0 \le j < t$ as

$$\begin{aligned}\left\langle x\left|\left(Y^{(j)}\right)^\dagger Y^{(j)}\right|y\right\rangle &= \left\langle \psi_x^{(j)}\middle|\psi_y^{(j)}\right\rangle \\ &= \sum_{i=0}^{n}\left\langle \psi_{x,i}^{(j)}\middle|\psi_{y,i}^{(j)}\right\rangle \\ &= \sum_{i=0}^{n}\left\langle x\middle|M_i^{(j)}\middle|y\right\rangle \quad ,\end{aligned}$$

thus,

$$\left(Y^{(j)}\right)^\dagger Y^{(j)} = \sum_{i=0}^{n} M_i^{(j)} \quad .$$

Similarly, for each $1 \leq j < t$, we have

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

Connecting the two results by the second constraint given in (4) for all $1 \leq j \leq t-1$, Lemma 1 ensures that for any $1 \leq j \leq t-1$ and for all x, a unitary U_j that satisfy $U_j |\phi_x\rangle = |\psi_x\rangle$ can be explicitly constructed. For the initial value of $j = 0$, there exists a U_0 such that $U_0 |\phi_0\rangle = |\psi_0\rangle$ by a similar reasoning, but since $\left(F^{(j)}\right)^{\dagger} F^{(j)} = E_0$, the existence of U_0 is guaranteed by the constraint given in (3).

Up to now, we exactly followed the method by Montanaro et al. to determine the unitaries between the oracle calls. The only remaining part that is not specified is the complete set of orthonormal projection operators P_z. However, their derivation is rather straightforward using the same idea above and for most of our work, we will not be interested in the projectors themselves, so we refer to [17] for the final part of the proof. While the easiest choice given by Eq. (9) always result in a QQA that works, but results in $M_i^{(j)}$'s with maximal ranks. As noted in [17], carefully selected $M_i^{(j)}$'s which have ranks that are upper bounded by some limit r would result in $r-$dimensional states $\left|\psi_{x,i}^{(j)}\right\rangle$. Therefore, $\lceil \log r \rceil$ qubits would be enough to represent the states in the work register. Moreover, without loss of generality, all the states and unitaries can be taken to be real.

4.1 EXACT$_2^4$

In [17], Montanaro et al. listed all the functions on n bits for $n \leq 4$ and all symmetric functions for $n \leq 6$. For each such function, they evaluated the success probability of a quantum algorithm computing that function and making t ($1 \leq t < n$) queries using the semidefinite method of Barnum et al. When they achieved a sufficiently small error like $\varepsilon < 0.001$, they reported it as a candidate for an exact query algorithm. In their numerical calculations, Montanaro et al. used the convex optimization package CVX [25,26] for MATLAB to implement the SDP method. CVX is a convex optimization package written for MATLAB. It allows the user to translate optimization problems to a compact and easily comprehensible MATLAB code using a strict ruleset checking convexity of the operations and variables. Currently, CVX supports two free solvers, SeDuMi and SDPT3 and two commercial solvers, Gurobi and Mosek.

Among those functions, EXACT_2^4, which outputs 1 if and only if its input has a Hamming weight 2 is shown to have a quantum query complexity of 2. We can also interpret this function as the simplest example of distinguishing balanced (n-bit)inputs and inputs with weights $\{0, 1, n-1, n\}$. Compare it with the single query Deutsch–Jozsa problem, where balanced inputs are distinguished from the sets with weights 0 or n.

Here, we will briefly summarize the exact QQA for EXACT_2^4 since one of our contributions is related to how the special algorithms that computes functions like EXACT_2^4 can be systematically generated. We begin with the standard initial state

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

where we only use an input register with basis states $\{|0\rangle, \ldots, |4\rangle\}$. This is the minimal configuration because we need a basis state for each input bit; therefore, we have $|1\rangle$ to $|4\rangle$ and the $|0\rangle$ state is there for the null query. We don't need a work space. The unitary that is key to the algorithm for evaluating EXACT_2^4 is the following 5×5 matrix:

$$U = \frac{1}{2}\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & \omega & \omega^2 \\ 1 & 1 & 0 & \omega^2 & \omega \\ 1 & \omega & \omega^2 & 0 & 1 \\ 1 & \omega^2 & \omega & 1 & 0 \end{pmatrix} \quad , \tag{11}$$

where $\omega = e^{2\pi i/3}$ is a complex root of 1. For each input x, we apply the corresponding oracle O_x, U and apply O_x one more time. Finally, we apply an operator that projects the final state onto the one-dimensional subspace spanned by the state $|\psi\rangle$ or to the subspace orthogonal to $|\psi\rangle$. It can be shown that the sequence of operations $O_x U O_x$ may bring only a phase factor in front of the initial state $|\psi\rangle$ when $|x| = 2$, but it maps $|\psi\rangle$ onto the ortogonal subspace when $|x| \neq 2$.

5 Improvements on the SDP method

In this work, we utilized the code given in the archive version of [17] and made our modifications and improvements on top of their code. We run our CVX programs on the solvers Mosek, SDPT3 and SeDuMi, yet sometimes SDPT3 gave results inconsistent with the other solvers or caused an error. In such instances, we report the results that are obtained using the other two solvers.

5.1 EXACT_2^4

The method in Sect. 4 explicitly describes how a QQA with a fixed number of queries that solves a total Boolean function computation problem can be constructed. However,

the default choice given in (9), often does not lead to optimal algorithms in terms of the dimension of the work space. Since the quantum resources are limited, we want a work space as small as possible. The solution to the EXACT_2^4 problem given in Sect. 4 is a clear example of this. In a closer inspection, we observe that the numerical evaluation with CVX results in rank 1 $M_i^{(0)}$ and rank 3 $M_i^{(1)}$ matrices. That leads to 15×15 real-valued unitary transformations instead of the more compact 5×5 complex-valued unitary matrix U given in (11). In Montanaro et al.'s paper, U was found by the inspiration from the numerical results. As our first contribution, we show that U in (11) can be constructed numerically, therefore an exact QQA can be constructed more systematically using a work space with the smallest dimension.

The first issue is that the choice in Eq. (9) leads to $s = \dim S$ dimensional vectors in general; therefore, the memory register might be unnecessarily large even for the algorithms computing highly symmetrical functions. In our approach, we first run the code given in [17] (see arXiv ver.) for the EXACT_2^4 problem. For this problem, we need at least 2 queries and if we make an error minimization, we obtain matrices $M_i^{(j)}$ with a maximum rank $r_{\max} = 3$, and a sufficiently small error $\varepsilon < 0.001$(or sufficiently high success probability, $p_S > 0.999$). Accepting this as a good evidence for the existence of an exact solution with rank 3 matrices, we now make two changes to the program. First, instead of the trivial choice given in (9), we evaluate the singular value decompositions

$$V_i^{(j)} \Sigma_i^{(j)} \left(W_i^{(j)} \right)^\dagger = M_i^{(j)} \ , \tag{12}$$

for each matrix $M_i^{(j)}$, where $\Sigma_i^{(j)}$ are diagonal matrices with the nonnegative singular values on the diagonal. Since $M_i^{(j)}$ are symmetric, the unitary parts of the decomposition, $V_i^{(j)}$ and $W_i^{(j)}$ are always the same matrices. Now, we make the choice

$$\left| \psi_{x,i}^{(j)} \right\rangle = V_i^{(j)} \sqrt{\Sigma_i^{(j)}} \, |x\rangle \ , \tag{13}$$

and obtain the states of the memory $\left| \psi_i^{(j)} \right\rangle$ and $\left| \phi_i^{(j)} \right\rangle$ at each time step j one more time. With this choice, we reduce the dimension of the work space from $\dim S$ to $\max \left(\mathrm{rank} M_i^{(j)} \right)$. For the EXACT_2^4 problem, since error minimization gives matrices $M_i^{(j)}$ with a rank at most 3, a minimum of 3 real dimensional work space is sufficient. However, as we have seen in Montanaro et al.'s example, a better solution exists. In the following, we explain how to achieve such a configuration.

The second issue is that the minimization of error in the SDP formulation of query complexity does not always lead to matrices $M_i^{(j)}$ with the lowest rank. It would be very nice if we could add a rank constraint to the SDP, however rank constraint is not convex in general [27]. Therefore, we tried minimizing a quantity that is convex and compatible with the CVX ruleset. We observe that the function EXACT_2^4 is symmetric on its input bits, so we expect to see some kind of indirect effect of this symmetry in the matrices corresponding to the queried bits within the same step. Thus, instead

of minimizing the error, we changed the inequality constraint in (7) to an equality constraint and minimized the sum

$$\tau = \min \left(\sum_{i=1}^{n} \mathrm{tr} M_i^{(1)} \right) , \tag{14}$$

where we take the sum only over the $j = 1$'st step because the $j = 0$ corresponds to the trivial step of application of a Hadamard transform that takes place in the beginning of all the algorithms. In a t-step algorithm, we have another sum over the values $1 \leq j \leq t$. We also did not include $\mathrm{tr} M_0^{(j)}$ in the sum. That is partly because we observe that we can already obtain rank-1 $M_0^{(j)}$ with the error minimization. In addition, we believe that null queries can be treated differently since the query bit symmetry does not cover the null queries.

When we run the CVX code with the trace minimization condition, we saw that we were able to obtain rank-2 matrices. In general, we obtain low rank solutions for symmetric problems like EXACT_k^n, although those solutions may not as optimal as EXACT_2^4. Before coming up with the solution of Montanaro et al. that does not use any work space, we need one final step. We make use of the observation that for the EXACT_2^4 problem, trace minimization leads to two repeated nonzero singular values. Therefore, we have an extra unitary degree of freedom in our choice of the unitary matrices of the singular value decomposition [24]. Using this extra freedom, the amplitudes that corresponds to two-dimensional work space part of $\left| \psi_{x,i}^{(j)} \right\rangle$ can be further rotated to a configuration that it can be rewritten as real and imaginary parts of a complex vector. Therefore, the work space part can altogether be discarded and described as relative phases of the complex-valued state vectors that lie only on the input and query subspaces.

5.2 EXACT_3^6 and $\mathrm{EXACT}_{2,4}^6$

We applied trace minimization to the functions EXACT_3^6 and $\mathrm{EXACT}_{2,4}^6$. Computation of both functions requires 3 queries, therefore we minimize the sum

$$\tau = \sum_{j=1}^{2} \sum_{i=1}^{n} \mathrm{tr} M_i^{(j)} , \tag{15}$$

where this time we also sum over j. In general, we want to minimize the trace of $M_i^{(j)}$ that correspond to all the non-trivial steps of the computation. Since there are 3 queries, there are 2 non-trivial steps defined by the constraint given in 4. We achieved a maximum rank of $r_{\max}^{t.m.} = \max \mathrm{rank} M_i^{(j)} = 6$ for the computation of EXACT_3^6 using the trace minimization condition. Compare it with the maximum rank $r_{\max}^{e.m.} = \max \mathrm{rank} M_i^{(j)} = 12$, which is obtained using error minimization. Although we can not totally discard the work space like we did for EXACT_2^4, replacing error

minimization with trace minimization enabled us to show the existence of an exact quantum algorithm with $\lceil \log 6 \rceil = 3$ qubits instead of $\lceil \log 12 \rceil = 4$ qubits. Also note that we would need 6 ancilla qubits if we implemented the trivial choice given in Eq. (9).

Similarly, we expect a decent reduction in the number of work qubits for the computation of $\mathrm{EXACT}^6_{2,4}$. As its name would imply, $\mathrm{EXACT}^6_{2,4}$ outputs 1 only when Hamming weight of its input equals to 2 or 4, otherwise the output is 0. Like EXACT^6_3, the number of qubits needed for the work space can be fairly reduced, as we observed an even smaller maximum rank $r^{e.m.}_{\max} = 10$. However, trace minimization is not as effective this time, as we can only reduce the maximum rank to $r^{t.m.}_{\max} = 9$ resulting in no further change in the work space.

6 Application of SDP method to promise problems

In Barnum et al.'s definition of the SDP method, the function f is given as general as possible, i.e., f neither needs to be total nor restricted to be a decision function. Montanaro et al. used this method to obtain exact query complexities of total functions of few bits. Here, we aim to show that the SDP method can be used in a broader sense in such a way that it can also be utilized to obtain query complexities of the exact versions of promise problems (if not already) and non-decision functions. In this part, our primary goal is not to find better query complexities for the well-known problems, but to show that the method's applicability to a more general class of problems in a practical sense. In particular, we applied the generalized SDP method to Deutsch–Jozsa, and Grover's problems. In addition, we employed the method to the weight decision problem for several bits and were able improve a previous result [28].

Since each problem needs a slightly different formulation of SDP, we added a "wrapper" code in MATLAB around the original CVX code of Montanaro et al. and made the necessary modifications to incorporate the original formulation of Barnum et al. Extending the method to functions with more than two outputs is rather trivial. Adding extra Γ matrices to cover all possible outputs of the function (i.e., the number of elements of the codomain T) to implement the 4th and 5th conditions given in Eqs. (6) and (7) is enough. In order to deal with promise problems, we administered several additional modifications. First, we introduced an index set I in order to manage domains $S \subsetneq \{0,1\}^n$ that are strict subsets of the corresponding Boolean domain and extended the definition of E_i matrices to their index-set-limited versions. As a comparison, the index set were hard-coded to the set $\{1, \ldots, 2^n\}$ and the matrices E_i were always $2^n \times 2^n$ in [17]. We also made error minimization the default option but added a trace minimization option for the post-optimized evaluation of the functions like EXACT^4_2. The typical execution of our program is as follows:

1. Initialize the program for the desired choice of the problem. For each problem, extra options(if necessary) and parameters are recorded to a structure via a configuration step. This may also be implemented using a configuration file.

2. Using the configuration structure in step 1, generate a new file that consists of the core CVX part of the solution. Then, execute this new program with error minimization and allow for $M_i^{(j)}$ matrices with maximal rank.
3. Evaluate the maximum rank of the $M_i^{(j)}$ matrices that are found in Step 2, and run the optimization for a second time with setting maximum rank to this newly obtained value. Repeat this step unless the maximum rank is the same as value that was found in the previous optimization.
4. If trace minimization is needed, optimize using trace minimization for one last time.
5. Perform the post-processing steps of Sect. 5. These include calculation of the state vectors of the memory at each time step and evaluation of the unitary matrices that link them. If any real to complex conversion is needed that is carried out in this step.

Up to now, we have drawn a general picture of how we improved the SDP method and how we implemented our modified version. Let us see how we can apply it to promise problems.

6.1 Deutsch–Jozsa algorithm

Deutsch–Jozsa algorithm distinguishes the balanced n-bit long strings of Hamming weight $|x| = n/2$ and constant bit strings of Hamming weights $|x| = 0$ or $|x| = n$. For $n > 2$, it is a promise problem. Although Deutsch–Jozsa problem is the easiest to adapt in our setting, it is not the ideal problem to see the full power of the method. We already know that only a single quantum query is enough to distinguish balanced and constant strings; therefore, there is not much room for improvement that could be hidden between the intermediate steps of computation. The method can at best give the original Deutsch–Jozsa algorithm. We run the program for several times and for $n = 2, 4, 6$ and 8, we verified that exact quantum query complexity is 1 for each of those values of n. For larger values, the computation becomes intractable on an Intel i7-6700 3.40 GHz×4cores desktop computer with 32GB RAM. The method gives some asymmetric solutions $\left|\psi_{x,i}^1\right\rangle$ that satisfy the no-extra-work-space and zero-error conditions; however, they do not lead to the known Deutsch–Jozsa algorithm. We believe that with further modifications, it may be possible to recreate Deutsch–Jozsa algorithm, only using the numerical methods but our motivation is to show the wide range of applicability of the SDP method rather than recreating specific algorithms. Therefore, we move on to other examples.

6.2 Grover's algorithm

Like Deutsch–Jozsa problem, we define Grover's problem in terms of bit strings as the following: we are given a bit string $x_\omega = 0 \ldots 010 \ldots 0$ with length n and the only nonzero bit is at the ω'th position. We wish to distinguish all the strings x_ω and $x_{\omega'}$ from each other for any $1 \leq \omega < \omega' \leq n$. This formulation yields a definition with a partial function f, because we prohibited strings with Hamming weight other than

Table 1 Comparison of query complexities for Grover's algorithm in the literature and the those found by the SDP method

n	Original	SDP
2–4	1	1
5–10	2	2
11–20	3	3
21–25	4	4

1. However, as far as the query complexity is concerned, our numerical evaluation of query complexity should be compatible with the original Grover's algorithm. Think of this adaptation as an extra step of demultiplexing $\lceil \log_2 n \rceil$ bits into n bits. This may cause a constant memory overhead but it will not change the query complexity. In this problem, we see the first advantage of the SDP method, that is, we can easily obtain exact query complexities with minimal modification. Compare it with the well-known fact that the original Grover's algorithm being a bounded-error algorithm except for the case of discriminating 4 different oracles, although there are several ways to make it exact [29–31].

Using the modification above, we run the SDP algorithm for $n = 2$ to $n = 25$ bit inputs. Classically, one needs to make $O(n)$ checks to find the only nonzero output. First, we give a quick analysis for the original Grover's algorithm with a suitable modification in the last step such that it outputs the results with no error. To do this, we apply the Grover operator until the state of the computer overshoots the solution state (see [32] for a geometric interpretation of the Grover's algorithm). We assume that it is possible to design an exact Grover's algorithm with the known methods that uses the same number of queries for the first overshoot. The query complexities of the original exact Grover's algorithm are shown in the second column in Table 1. We list the exact query complexities that we have found with the SDP method on the third column. Like Montanaro et al, we first run our modified program for $t = 1$ queries and increased the number of queries by one and run again until we reach a zero-error algorithm. For the practical purposes, we accept errors $\varepsilon < 10^{-6}$ as zero error in this analysis. As Table 1 indicates, our results fully agree on the exact query complexity for the strings with lengths varying between 2 and 25. Unfortunately, numerical optimization becomes computationally impractical for problems with $n >\sim 25$ with the current typical desktop computers.

6.3 Weight decision

In this problem, we are guaranteed that a bit string with a Hamming weight equal to either ω_1 or ω_2 is given to us but we do not know which one is given. Any string with either one of those weights is equally probable. We are tasked with determining the weight of the input making minimal quantum queries. Clearly, this is a promise problem. As an example, let us consider that we want to discriminate the $n = 8$ bit long strings with weights $\omega_1 = 0$ and $\omega_1 = 4$. We know that this can be done with only one query because Deutsch–Jozsa algorithm already discriminates strings with weights $\omega_1 \in \{0, 8\}$ from the strings with $\omega_2 = 4$ making a single query. Although one can use quantum counting [33] to differentiate bit strings of any different weights, algorithms

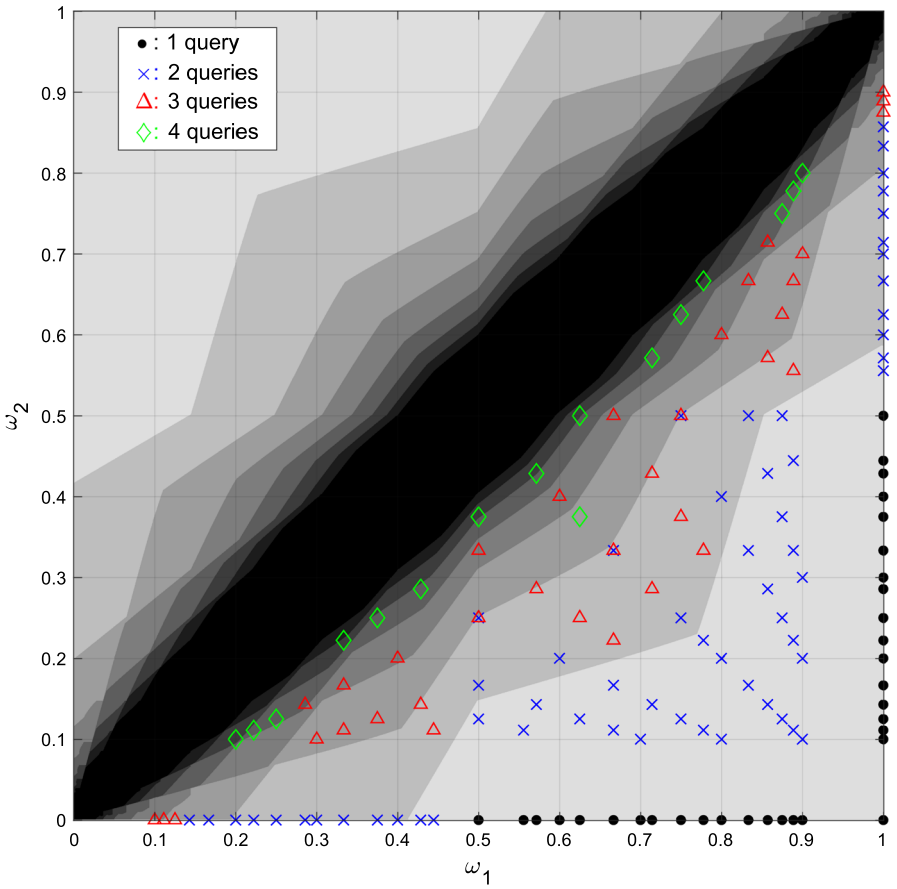

Fig. 1 Exact query complexities for the weight decision problem. The filling colors indicate the query complexities given in [28]. It should be understood that each time a region is filled with a darker shade, a weight combination in that region requires one more query. Current results are shown with 4 different markers (•, ×, △, ◇) that indicate query complexities from 1 to 4 (Color figure online)

that use slightly less resources exist [28,34] for the special problem of discriminating only two weights. All of these quantum algorithms are based on Grover's algorithm; therefore, they all exhibit a square root speedup for the weight decision problem. It turns out that the SDP method is quite suitable for discriminating short bit strings.

Following similar steps, we investigated the exact query complexity of weight decision of bit strings of lengths up to $n = 10$. Using a simple heuristic, we estimated the complexity of computation of doing this task on a computer and we sorted the weight decision problems from the least complex to the most complex using this rough estimate. Then, we run the optimization code for each of these weight combinations. We run the code for this problem on an Intel Xeon 20×core server with a 64 GB RAM. Clearly, for small n, (e.g., $n \leq 5$) the optimization does not take too much time. However, near the other end of the sorted list, all of three solvers cause a computational error because of the size of the optimization problem. Thus, we were not able to obtain

a complete list of weight decision complexities for strings with lengths up to $n = 10$. Even so, our results show that there is a room for improvement over the existing methods.

Our current results are shown in Fig. 1. In order to make a comparison with the previous works, we plot our current results on top of the symmetrized results from a previous work [28]. The newly evaluated number of queries needed for the discrimination of weight combinations are indicated by the 4 different types of markers. It is clear that current results are consistent with the previous ones as one would expect. See for example, except for a few instances, the area where (blue $\times$)'s are dense mostly overlap with the area with the lightest shade of gray. Those two different indicators both show that 2 queries are required in the current analysis and the analysis in [28]. However, for the decision problems that require more than 2 queries, the superiority of the results found by the SDP method becomes more apparent as it can be seen from the graph that the 3-query points (red $\triangle$) and the 4-query points (green $\diamond$) further penetrate into the territories with darker shades (up to 6 queries with the method in [28]). There are a few points indicated with more than one value that is because some weight combinations are integer multiples of already evaluated combinations. As an example, consider the decision problem of x_1 and x_2 with lengths $n = 4$ with $|x_1| = 1$ and $|x_2| = 2$ and strings with $|x_1| = 2$ and $|x_2| = 4$ of length $n = 8$. They both correspond to the same point on the graph but may have different query complexities.

6.4 Other algorithms

Finally, we computed the exact query complexity of some selected functions. Specifically, we applied the SDP method to the computation of

- And–Or trees,
- Kushilevitz function, which is first mentioned in [35] and also discussed in [36] and [37],
- maximum deviation, and
- sorted input bits, which is also known as Ambainis function [36].

The collection of our results for these selected functions are given in an arranged form in Table 2. In order to avoid uncertainty that may be caused by the numerical errors, we tried to set the optimal error probability limit as low as possible. Ideally, this error should reduce to 0. Fortunately, for all the problems we considered, at some point, when we add one more query, the optimized error decreases significantly. Therefore, it was easy to set a point of threshold to accept the computation as an exact computation. In the following analyses, we did not change the error limitation $\varepsilon < 10^{-6}$ in order to make it compatible with the other algorithms that we studied. If necessary, the limit can be easily made orders of magnitude tighter as it can be seen from Table 2. Still, we must emphasize that the exact queries we provide here should rather be accepted as a strong evidence for the existence of exact query algorithms. The algorithms found by the SDP method must be further analyzed and a decision regarding the exactness of the algorithm should be made after a careful theoretical consideration. Also, we should mention that in Table 2, we included only the errors that contributed to the decision of the exactness of the algorithm.

Table 2 Exact query complexities and minimal errors for some special Boolean functions of several bits. Columns indicate the following in the respective order: function f, the number of input bits n, classical query complexity $D(f)$, exact quantum query complexity $Q_E(f)$ and the minimized errors for the corresponding SDP with queries $1-5$. Partial functions are indicated by the symbol $(*)$

f	n	$D(f)$	$Q_E(f)$	1	2	3	4	5
And–Or trees	4	4	3	2.9×10^{-1}	3.1×10^{-2}	7.1×10^{-14}		
And–Or trees	6	6	4		1.2×10^{-1}	9.1×10^{-3}	2.0×10^{-12}	
Kushilevitz$*$	6	5	3		1.3×10^{-1}	1.0×10^{-14}		
Kushilevitz [35,36]	6	6	4			2.6×10^{-2}	1.9×10^{-13}	
Maximum dev.$*$	4	2	1	3.5×10^{-14}				
Maximum dev.$*$	6	5	2	2.7×10^{-1}	1.3×10^{-13}			
Maximum dev.$*$	8	7	4			2.6×10^{-2}	1.9×10^{-13}	
S. input bits [36]	3	3	2	1.0×10^{-1}	2.1×10^{-13}			
Sorted input bits	4	4	2	1.5×10^{-1}	2.5×10^{-13}			
Sorted input bits	5	5	3		3.7×10^{-2}	2.0×10^{-12}		
Sorted input bits	6	6	4			9.0×10^{-3}	2.8×10^{-12}	
Sorted input bits	7	7	5			2.1×10^{-2}	2.4×10^{-3}	1.6×10^{-11}

Due to the rapidly increasing computational complexity of convex optimization with the number of input bits n, we could evaluate query complexities for only a few bits of inputs. Concerning And–Or trees, we were able to obtain complexities only for the functions $f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ and $f(x_1, x_2, x_3, x_4, x_5, x_6) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (x_5 \wedge x_6)$. Next, we evaluated both versions of the Kushilevitz function. We first evaluate the query complexity of the partially defined version whose output is given as

- $f(x) = 0$ if $|x| = 3$ and x is one of the following: 111000, 011100, 001110, 100110, 110010, 101001, 010101, 010011, 001011.
- $f(x) = 1$ if $|x| = 3$ and x is one of the following: 110100, 110001, 101100, 101010, 100011, 011010, 011001, 010110, 000111.

Including the partial version of the Kushilevitz function, we indicate the partial functions with a symbol $(*)$ in Table 2. The total version of Kushilevitz function accepts all $2^6 = 64$ possible inputs and it is additionally defined to be 0 on inputs x with $|x| = 0,\ 4$ or 5 and it is defined to be 1 on inputs with $|x| = 1,\ 2$ or 6. See also [36,37] for more details about the Kushilevitz function.

Another partial function we analyzed is the maximum deviation function which we define as follows: given an even number n, we promise that the n-bit input bit string x is balanced, that is it has equal number of 0's and 1's. On input x, the maximum deviation function f outputs an integer between 1 and $n/2$, which is the maximum cyclic length of consecutive 0's or consecutive 1's. For example $f(01010101) = 1$ and $f(01110010) = 3$. Classically, we have to wait until the one before the last bit for inputs longer than $n = 4$. If the inputs had been further restricted to the perfectly periodic series of 0's and 1's, a single query and a quantum Fourier transform would be enough. However, our definition is sitting somewhere between a completely random function and a perfectly periodic function, therefore we expect a modest speedup in the exact evolution of f. As we see in Table 2, we verify this conjecture at least for inputs with lengths 4, 6 and 8 bits. Lastly, we evaluated the exact QQC of the sorted input bits function [36,38]. Its output is defined to be 1 only when input bits are sorted (reverse order is also accepted) and it is defined to be 0 otherwise. The 4 bit version of the function was crucial in proving the maximal separations using the adversary bounds [37]. It is clear from Table 2 that, for strings with 4–7 bits, there is a 2 query difference between quantum and classical exact query complexities.

7 Conclusions and further study

We have shown that the SDP method by Barnum et al. [15] is a powerful and versatile tool for calculating exact query complexities of partial Boolean functions. We have demonstrated that minimizing total of the traces instead of optimizing error and using the singular value decomposition of the overlap matrices instead of the trivial choice in (9) can give better results once we are sure that an exact QQA exists for a given problem. For EXACT_2^4, in addition to those modifications, using the symmetry of repeated singular values, we were able to reproduce the efficient algorithm in [17], thus achieving the same design in a more systematic way. An immediate open problem is whether these improvements would enable us to design more efficient quantum algorithms (in

terms of memory) for computing more general functions like $\text{EXACT}^n_{n/2}$. We have already shown that using singular value decomposition, we were able to reduce the number of necessary ancilla qubits for EXACT^6_3 and $\text{EXACT}^6_{2,4}$.

Next, we applied the SDP method to promise problems and showed that the method can be easily applied to these problems once they were properly identified with evaluation of (mostly) a Boolean function in the query complexity model. We successfully verified the exact query complexities of Deutsch–Jozsa and Grover's problems. Then, we employed the method for the weight decision problem of bit strings with lengths up to 10-bits and showed that the known complexities [28,34] for the problem can be further improved. Finally, we tested the SDP method with four more special functions, some of which were already shown to exhibit quantum speedup with other methods and we saw that the quantum speedup can clearly be achieved for small-input-size versions of these functions.

Although the numerical evaluation based on the SDP method cannot replace other methods like the polynomial method or the adversary methods, it can be quickly adapted to any given problem and can be used as preliminary test before one would continue with a more detailed analysis. An exhaustive search for exact QQA's for all partial functions like the one for total functions which was given in [17] would evidently be an important work since it could lead us to new classical to quantum separations. The biggest obstacle to doing so is the obvious huge size of the search space. A further task along this direction could be finding heuristics such as utilizing symmetries in such functions to make the problem more manageable.

Acknowledgements This work has been supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) - 2218 Fellowship for Postdoctoral Researchers. We would like to thank Özgür Çakır for helpful discussions.

References

1. Buhrman, H., De Wolf, R.: Complexity measures and decision tree complexity: a survey. Theor. Comput. Sci. **288**(1), 21–43 (2002)
2. Deutsch, D., Jozsa, R.: Rapid solution of problems by quantum computation. Proc. R. Soc. Ser. A **439**(1907), 553–558 (1992)
3. Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Phys. Rev. Lett. **79**, 325 (1997)
4. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007)
5. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM J. Comput. **26**(5), 1510–1523 (1997)
6. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)
7. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
8. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: A limit on the speed of quantum computation in determining parity. Phys. Rev. Lett. **81**, 5442–5444 (1998)
9. Ambainis, A.: Superlinear advantage for exact quantum algorithms. SIAM J. Comput. **45**(2), 617–631 (2016)
10. Ambainis, A., Balodis, K., Belovs, A., Lee, T., Santha, M., Smotrovs, J.: Separations in query complexity based on pointer functions. J. ACM **64**(5), 1–24 (2017)
11. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. J. ACM **48**, 778–797 (2001)

12. Ambainis, A.: Quantum lower bounds by quantum arguments. J. Comput. Syst. Sci. **64**, 750–767 (2002)
13. Vazirani, U.: On the power of quantum computation. Philos. Trans. R. Soc. London Ser. A: Math. Phys. Sci. 356 **356**, 1759–1768 (1998)
14. Høyer, P., Spalek, R.: Lower bounds on quantum query complexity. Bull. EATCS **87**, 78–103 (2005)
15. Barnum, H., Saks, M., Szegedy, M.: Quantum query complexity and semi-definite programming. In: Proceedings of 18th Annual IEEE Conference on Computational Complexity, pp. 179–193. (2003)
16. Spalek, R., Szegedy, M.: All quantum adversary methods are equivalent. Theory Comput. **2**(1), 1–18 (2006)
17. Montanaro, A., Jozsa, R., Mitchison, G.: On exact quantum query complexity. Algorithmica **71**, 775–796 (2015). arXiv:quant-ph/1111.0475v2
18. Gruska, J., Qiu, D., Zheng, S.: Generalizations of the distributed Deutsch–Jozsa promise problem. Math. Struct. Comput. Sci. **27**(3), 311–331 (2017)
19. Zheng, S., Qiu, D.: From Quantum Query Complexity to State Complexity, Gruska Festschrift. Lecture Notes in Computer Science, vol. 8808, pp. 231–245. Springer, Cham (2014)
20. Even, S., Selman, A.L., Yacobi, Y.: The complexity of promise problems with applications to public-key cryptography. Inf. Control **61**, 159–173 (1984)
21. Canteaut, A., Videau, M.: Symmetric boolean functions. IEEE Trans. Inf. Theory **51**(8), 2791–2811 (2005)
22. Goldreich, O.: On promise problems: a survey. Essays Mem. Shimon Even, LNCS **3895**, 254–290 (2006)
23. Ambainis, A., Spalek, R., de Wolf, R.: A new quantum lower boundmethod, with applications to direct product theorems and time-space tradeoffs. Algorithmica **55**(3), 422–461 (2009)
24. Horn, R.A., Johnson, C.R.: Matrix Analysis, 2nd edn. Cambridge University Press, New York (2013)
25. Grant M., Boyd, S.: CVX: Matlab software for disciplined convex programming, version 2.0 beta. (2013). http://cvxr.com/cvx. Accessed Dec 2017
26. Grant M., Boyd, S.: Graph Implementations for Nonsmooth Convex Programs, Recent Advances in Learning and Control (a tribute to M. Vidyasagar), V. Blondel, S. Boyd, and H. Kimura, (eds.), 95–110, Lecture Notes in Control and Information Sciences, Springer, (2008) http://stanford.edu/~boyd/graph_dcp.html
27. Dattorro, J.: Convex Optimization and Euclidean Distance Geometry. $\mathcal{M}\epsilon\beta oo$, Palo Alto (2005)
28. Uyanık, K., Turgut, S.: A new sure-success generalization of Grover iteration and its application to weight decision problem of Boolean functions. Quantum Inf Process. **12**(11), 3395–3409 (2013)
29. Brassard, G., Høyer, P., Mosca M., Tapp. A.: Quantum amplitude amplification and estimation. In: Lomonaco, S.J. Jr. (ed.) Quantum Computation and Quantum Information: A Millennium Volume, of AMS Contemporary Mathematics Series, vol. 305, pp. 53–74 (2002). arXiv:quant-ph/0005055
30. Høyer, P.: Arbitrary phases in quantum amplitude amplification. Phys. Rev. A **62**, 052304 (2000)
31. Long, G.L.: Grover algorithm with zero theoretical failure rate. Phys. Rev. A **64**, 022307 (2001)
32. Nielsen, M.A., Chuang, I.L.: Quantum Comput. Quantum Inf. Cambridge University Press, Cambridge (2000)
33. Brassard G., Høyer P., Tapp A.: Quantum counting. In: Proceedings of the 25th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 1443, pp. 820–831 (1998)
34. Choi, B.S., Braunstein, S.L.: Quantum algorithm for the asymmetric weight decision problem and its generalization to multiple weights. Quantum Inf. Process. **10**(2), 177–188 (2011)
35. Nisan, N., Wigderson, A.: On rank vs. communication complexity. Combinatorica **15**(4), 557–565 (1995)
36. Ambainis, A.: Polynomial degree vs. quantum query complexity. J. Comput. Syst. Sci. **72**(2), 220–238 (2006)
37. Høyer, P., Lee, T., Spalek, R.: Negative weights makeadversaries stronger. In: Proceedings of 39th ACM STOC, pp. 526–535(2007)
38. Laplante, S., Lee, T., Szegedy, M.: The quantum adversary method and classical formula size lower bounds. Comput. Complex. **15**(2), 163–196 (2006)