Semidefinite Program Solver to Find the Optimal Quantum Query Complexity and Query Optimal Quantum Algorithm of Boolean Functions

Middlebury College, Fall 2019

MICHAEL CZEKANSKI, Middlebury College
R. TEAL WITTER, Middlebury College

ABSTRACT

We implement an algorithm that takes in a Boolean function and outputs the optimal quantum query complexity of the function as well as a query optimal span program for evaluating it on a quantum computer. Let f be a Boolean function and x be a bitstring input to f. We consider the query model where an algorithm evaluates $f(x)$ by asking an oracle about the bits of x. The optimal quantum query complexity of f is the fewest queries any quantum algorithm takes to determine $f(x)$ over all inputs x. We find the optimal quantum query complexity of f by solving a semidefinite programming (SDP) problem. The SDP solution also provides a span program that evaluates f in optimal quantum query complexity. Our numerical results prove to be in line with general analytic understanding, and the span programs we implement are shown to correctly evaluate the given function.

1 INTRODUCTION

An important area of research in quantum computing is to determine the problems for which quantum computers can outperform classical computers. To avoid technical problems in the comparison of computers–the largest of which is that large-scale quantum computers currently do not exist–we turn to theoretical analysis to characterize the efficiency of quantum and classical algorithms.

We implement a tool that places an asymptotic lower bound on runtime by calculating the optimal quantum query complexity of a Boolean function. The query complexity of a function is the number of times we must examine the contents of the input string to determine the correct output of the function. Each query of the given function takes a certain amount of time, therefore the query complexity places a lower bound on runtime. The reverse is not true: the runtime may asymptotically exceed query complexity if the algorithm performs substantially more computations than queries.

To give a concrete example of query complexity, recall the canonical OR function. The n-bit OR function returns 0 if there are only 0's in the input string and 1 if there are any 1's in the input string. In the worst case, a classical algorithm must check that every single bit of the input is a 0 to return 0. Therefore the classical query complexity of the n-bit OR function is n. However, the quantum algorithm Grover's search can solve the function OR in $\sqrt{n}$ queries [Grover 1996]. In the case of OR, we conclude that the quantum algorithm outperforms the classical algorithm.

For an arbitrary function f, Reichardt presents a semidefinite program (SDP) whose solution corresponds to the optimal quantum query complexity of f [Reichardt 2009]. In addition, the solution of the same SDP can be used to construct a span program that, when run on a quantum computer, meets the optimal quantum query complexity.

Our contribution is an SDP solver that finds the optimal quantum query complexity and query optimal quantum algorithm for given Boolean functions. While the SDP can be solved for arbitrary functions with finite outputs, we limit the scope of our work to Boolean functions.

Although there are many SDP solver libraries such as CVXOPT and SDPA for convex optimization problems, none easily support solving Reichardt's SDP [Andersen et al. 2013; SDPA-Author 2013]. As a result, we develop an implementation that solves the SDP and includes optimizations specific to our problem formulation. We first convert Reichardt's form into the standard SDP form as described by Boyd [Boyd and Vandenberghe 2004], which we verify here. To solve the SDP, we implement Wen et al.'s alternating direction method (ADM) algorithm in order to exploit the sparsity of our SDP. We finally optimize the functions and data structures we use to speed up our program [Wen et al. 2010].

Our program takes as input a set of bitstrings D and a Boolean function $f : D \to \{0,1\}$. By solving Reichardt's SDP problem with Wen et al.'s ADM algorithm, we return the optimal quantum query complexity of f and a quantum algorithm that meets this query complexity. We hope that our solver proves useful for researchers constructing optimal quantum algorithms.

2 METHODOLOGY

2.1 Original Form of the SDP

Consider a Boolean function f such that $f : D \to E$ where $D \subseteq \{0,1\}^n$ and $E \subseteq \{0,1\}$. We state Reichardt's formulation of the SDP [Reichardt 2009]. (Note that $[n]$ denotes the array $[1, 2, ..., n]$.) The quantum query complexity of f is

$$\min_{\mathbb{X}} \max_{y \in D} \sum_{j \in [n]} \langle y, j | \, \mathbb{X} \, | y, j \rangle \tag{1}$$

for some $\mathbb{X}$ subject to the constraints that $\mathbb{X}$ is positive semidefinite (all its eigenvalues are non-negative) and for ordered pairs (y,z) such that $f(y) \neq f(z)$,

$$\sum_{j \in [n] : y_j \neq z_j} \langle y, j | \, \mathbb{X} \, | z, j \rangle = 1. \tag{2}$$

Authors' addresses: Michael Czekanski, Middlebury College; R. Teal Witter, Middlebury College.

2.2 Standard Form of the SDP

In order to implement an algorithm that solves Reichardt's formulation, we convert the SDP into the canonical standard form [Boyd and Vandenberghe 2004]. Then the standard form objective function of the SDP is

$$\mathrm{tr}(C\mathcal{X}) \tag{3}$$

for some fixed matrix C subject to the constraints that $\mathcal{X}$ is semidefinite and for $i \in [p]$

$$\mathrm{tr}(A_i\mathcal{X}) = b_i \tag{4}$$

for some fixed matrices A_i and b_i.

2.3 Converting from Original to Standard Form

We now convert the original form of the SDP to the standard form by constructing appropriate matrices A_i, b_i, $\mathcal{X}$, and C.

To understand this conversion, we must first gain some intuition about $\mathbb{X}$. It is clear from eq. (1) that for each $y \in D$, there exists a corresponding $n\times n$ sub-matrix of $\mathbb{X}$ along its diagonal. Then $\mathbb{X}$ must have a total of $n|D|$ rows and $n|D|$ columns. In general, $|D| = 2^n$ but it is possible to take worst-case subsets of D and still find the correct solution, as discussed later.

To convert eq. (2) into eq. (4), we must construct A_i, b_i for $i \in [p]$. Let $p = |F|$, then each pair $(y, z) \in F$ is associated with a single A_i. For each (y, z), we are interested in the entries in $\mathbb{X}$ that correspond to the bits y_i and z_i where $y_i \neq z_i$ for $i \in [n]$. These entries are intuitively of interest because they are the bits that must lead to different outputs. They are necessarily off of the main diagonal, because $y \neq z$, so our A_i will map them to the main diagonal and ensure that the trace of $A_i\,\mathbb{X}$ is 1. Note that $b_i = 1$.

To place particular entries of $\mathbb{X}$ onto the main diagonal, let A_i be the transpose of the zero matrix with ones in the particular entries of interest. We illustrate on a small example. Let

$$\mathbb{X} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and the particular entry of interest be the top right value of $\mathbb{X}$. Then

$$A_i = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

so that

$$A_i \cdot \mathbb{X} = \begin{bmatrix} 2 & 0 \\ 4 & 0 \end{bmatrix}$$

and $\mathrm{tr}(A_i \cdot \mathbb{X}) = 2$ as desired.

Having converted the constraints, we must also convert the objective functions. Let c_i be the sum of the n entries in the i^{th} section of the main diagonal of $\mathbb{X}$ such that

$$c_i = \sum_{j\in[n]} \langle y, j|\,\mathbb{X}\,|y, j\rangle$$

where y is the i^{th} element of D for $i \in [|D|]$. Let z be $\mathrm{tr}(C\,\mathbb{X})$. The last challenge is to satisfy eq. (3) by enforcing that z is the maximum c_i. Our solution is to introduce non-negative slack variables s_i for $i \in [|D|]$ and the constraints $c_i + s_i = z$.

We construct a new matrix $\mathcal{X}$ to account for these slack variables. Call S the zero matrix whose diagonal holds the entries $s_1, \ldots, s_{|D|}, z$. Then

$$\mathcal{X} = \begin{bmatrix} \mathbb{X} & 0 \\ 0 & S \end{bmatrix}.$$

Since $\mathcal{X}$ is semidefinite, the main diagonal entries of $\mathcal{X}$ must be non-negative so $s_i \geq 0$. Thus C is defined to have $\mathrm{tr}\,C\mathcal{X}$ extract the bottom right entry of $\mathcal{X}$. Therefore, let C be the zero matrix with a single one in the bottom right entry.

To enforce that z is the maximum c_i over all i, we construct A'_i and set b'_i to 0 for $i \in [|D|]$. For the purposes of the standard form, we now have two sets of A_i and b_i constraints. Let A'_i be the zero matrix with ones along the entries corresponding to the input i of D, a one in the entry corresponding to s_i, and a negative one in the bottom right entry corresponding to the objective function z. Then

$$\mathrm{tr}(\mathrm{A}'_i\,\mathcal{X}) = c_i + s_i - z = 0.$$

Because s_i is non-negative and z is being minimized, z is both greater than or equal to all c_i and meets some c_i. Thus z is the maximum c_i as desired.

The solution $\mathcal{X}$ that satisfies the standard form according to the constructed A_i, A'_i, b_i, b'_i, and C will also hold the solution $\mathbb{X}$ to the original form. Therefore, to find the optimal quantum query complexity of f, we need only solve eq. (3) subject to our constraints.

2.4 Query Efficient Span Programs

A span program is a model of computation that outputs a Boolean value corresponding to whether or not a set of vectors "spans" to a target vector. The algorithm consists of a set of vectors such that on a given input, the vectors are partitioned into available and unavailable sets. The available vectors are turned into the columns of a matrix A. These vectors form a linear system $Ax = \tau$ for a fixed target vector τ. If this system is homogeneous– there exists some non-trivial vector x that satisfies the equation– then the algorithm returns true. Otherwise, the algorithm returns false.

Both Childs and Reichardt mathematically formulate algorithms for turning the results of the SDP we solved into a span program [Childs [n. d.]; Reichardt 2009]. To begin, we must construct a set of vectors $\langle v_{y,i}|$ for all $y \in D$ where $i = [n]$. Given any two input strings $y, z \in D$ such that $y \neq z$,

$$\sum_{i:y_i\neq z_i} \langle v_{y,i}|v_{z,i}\rangle = 1 - \delta_{f(y),f(z)}$$

where $\delta_{f(y),f(z)} = 1$ if $f(y) = f(z)$ and 0 otherwise.

Recall from eq. (2) that $\mathbb{X}$ satisfies

$$\forall(y, z) \in F \sum_{j\in[n]:y_j\neq z_j} \langle y, j|\,\mathbb{X}\,|z, j\rangle = 1.$$

We would like to construct $\langle v_{y,i}|$ and $|v_{z,i}\rangle$ from $\mathbb{X}$. So we define a matrix L such that $L^\dagger L = \mathbb{X}$. Then $\langle v_{y,i}| = \langle y_i|\,L^\dagger$. We can now reformulate the requirement of vectors $\langle v_{y,i}|$ as

$$\sum_{i:y_i\neq z_i} \langle v_{y,i}|v_{z,i}\rangle = \sum_{i:y_i\neq z_i} \langle y,i|\, L^\dagger L\, |z,i\rangle$$
$$= \sum_{i:y_i\neq z_i} \langle y,i|\, \mathbb{X}\, |z,i\rangle = \begin{cases} 1 & f(y)\neq f(z) \\ 0 & f(y)=f(z) \end{cases}$$

The constraints enforce the $f(y)\neq f(z)$ case and we programatically check the $f(y)=f(z)$ case before returning $\mathbb{X}$.

Given the set of vectors $\langle v_{y,i}|$ for all $y\in D$, $i=0,1,\ldots,n-1$, we construct matrix I that contains the vectors for the span program. We will make use of the sets $F_i=\{y\in D: f(y)=i\}$. (Note that $F_i[k]$ denotes the k^{th} element of F_i.)

$$I = \sum_{k\in[|F_0|],j\in[n],y=F_0[k]} |k\rangle\, \langle j,\overline{y_j}|\, \langle v_{y,j}| \tag{5}$$

This matrix I is divided into sub-matrices that correspond to different bits in the input to the algorithm. The columns of I are evenly divided into n chunks corresponding to each input bit. Each of these sub-matrices is further evenly divided into a left and right sub-matrix, corresponding to a 0 or 1 in the relevant bit of the input. Let $I(y)$ be the set of columns in I that are available on input $y\in D$. The target vector τ is a vector of ones vector with the same dimension as I. The output is true if and only if the available vectors $I(y)$ span to τ. Reichardt proves that the span program I and τ is a query optimal quantum algorithm for f [Reichardt 2009].

3 RESULTS

3.1 Step-by-step Example with OR

Recall the two bit function OR which returns 1 if there is at least a single 1 in the input and returns 0 otherwise. To understand Reichardt's formulation and the conversion to Boyd's standard form, consider as inputs to our tool the case with function $f: D\rightarrow E$ where $D\subseteq\{0,1\}^2$ and $E=\{OR(x): x\in D\}$. Let F be the set of (y,z) such that $f(y)\neq f(z)$. In this case, $F=\{(00,01),(00,10),(00,11),(01,00),(10,00),(11,00)\}$

Let

$$\mathbb{X} = \begin{array}{c c} & \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array} & \begin{bmatrix} X_{(00,00)} & X_{(00,01)} & X_{(00,10)} & X_{(00,11)} \\ X_{(01,00)} & X_{(01,01)} & X_{(01,10)} & X_{(01,11)} \\ X_{(10,00)} & X_{(10,01)} & X_{(10,10)} & X_{(10,11)} \\ X_{(11,00)} & X_{(11,01)} & X_{(11,10)} & X_{(11,11)} \end{bmatrix} \end{array}.$$

The objective function of the SDP is

$$\max_{y\in D} \sum_{j\in[n]} \langle y,j|\, \mathbb{X}\, |y,j\rangle \tag{6}$$
$$= \max\{\operatorname{tr} X_{(00,00)}, \operatorname{tr} X_{(01,01)}, \operatorname{tr} X_{(10,10)}, \operatorname{tr} X_{(11,11)}\}$$

subject to constraints

$$\mathbb{X}\succcurlyeq 0$$

and

$$\forall (y,z)\in F \sum_{j\in[n]:y_j\neq z_j} \langle y,j|\, \mathbb{X}\, |z,j\rangle = 1. \tag{7}$$

Observe that eq. (7) is equivalent to

$$\operatorname{tr} X_{(00,01)} = \operatorname{tr} X_{(00,10)} = \operatorname{tr} X_{(00,11)} = 1$$
$$\operatorname{tr} X_{(01,00)} = \operatorname{tr} X_{(10,00)} = \operatorname{tr} X_{(11,00)} = 1.$$

Our goal is to minimize M, and thus f_{bound} by finding the optimal value of $\mathbb{X}$. By running our SDP solver, we obtain $\mathbb{X}$ with an objective function value of $\sqrt{2}$.

$$\mathbb{X} = \left[\begin{array}{cc|cc|cc|cc} 0.7 & 0 & 0 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 0.7 & 0 & 1 & 0 & 0 & 0 & 0.5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \sqrt{2} & 0 & 0 & 0 & 0.7 \\ \hline 1 & 0 & 0 & 0 & \sqrt{2} & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0.5 & 0 & 0 & 0 & 0.7 & 0 & 0.6 & 0 \\ 0 & 0.5 & 0 & 0.7 & 0 & 0 & 0 & 0.6 \end{array}\right]$$

We use our solution to construct the input vectors to the span program. Consider L such that $L^\dagger L=\mathbb{X}$. Then construct $\langle v_{x,i}|$ for all $x\in D$, $i\in[n]$. With eq. (5),

$$I = \begin{bmatrix} \langle 1|\,\langle v_{00,1}| & \langle 1|\,\langle v_{00,2}| \\ \langle 1|\,\langle v_{01,1}| & \langle 0|\,\langle v_{01,2}| \\ \langle 0|\,\langle v_{10,1}| & \langle 1|\,\langle v_{10,2}| \\ \langle 0|\,\langle v_{11,1}| & \langle 0|\,\langle v_{11,2}| \end{bmatrix}$$
$$= \left[\begin{array}{c|c|c|c} 0\cdots 0 & \langle v_{00,1}| & 0\cdots 0 & \langle v_{00,2}| \\ 0\cdots 0 & \langle v_{01,1}| & \langle v_{01,2}| & 0\cdots 0 \\ \langle v_{10,1}| & 0\cdots 0 & 0\cdots 0 & \langle v_{10,2}| \\ \langle v_{11,1}| & 0\cdots 0 & \langle v_{11,2}| & 0\cdots 0 \end{array}\right]$$

The columns of I are the columns used in the span program. If we remove rows of I corresponding to elements $x\in D$ such that $f(x)=1$ and columns of I that are the all zero vector, we obtain an equivalent span program where

$$I = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}$$

and $\tau=1$. The left two partitions correspond to the first bit in the input, and the right two partitions correspond to the second bit of the input. For each input bit, the relevant block of I is broken into left and right halves, corresponding to whether the bit is 0 (left) or 1 (right). It is clear from this matrix that if either input bit is 1 the function will return true and otherwise will return false.

3.2 Accuracy with OR and Parity

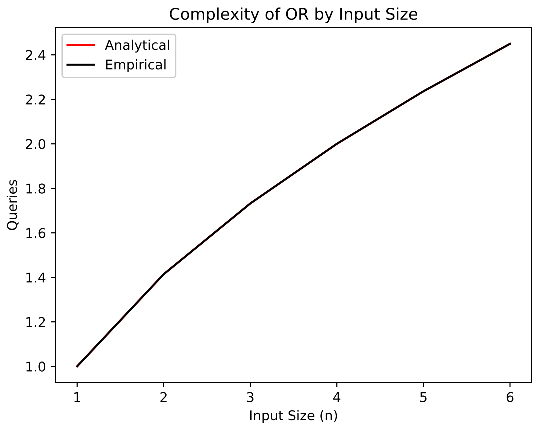

Fig. 1. The proven analytical optimal query complexity and calculated empirical optimal query complexity by size of input bitstring for OR.

We verify that the implementation works correctly with results for OR on various input sizes. The optimal quantum query complexity of OR is $\sqrt{n}$ and, as demonstrated in fig. 1, the empirical results match the analytical to the thousandth place.

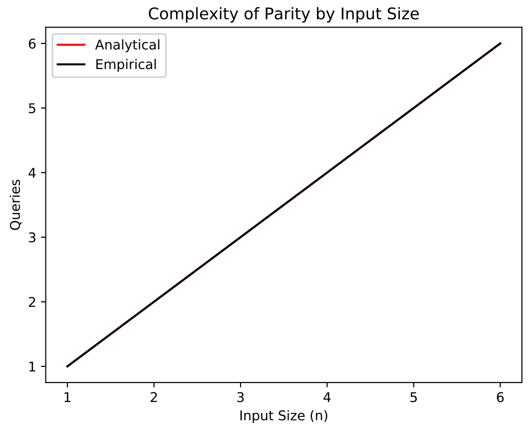

Fig. 2. The proven analytical optimal query complexity and calculated empirical optimal query complexity by size of input bitstring for parity.

The parity function, determining whether there are an even number of 1's an input bitstring, is known to have optimal quantum query complexity linear to the number of bits. The implementation also correctly calculates the complexity of parity to the thousandth place as demonstrated in fig. 2.

The results for OR and parity demonstrate that the implementation accurately calculates optimal quantum query complexity.

3.3 Runtime with OR

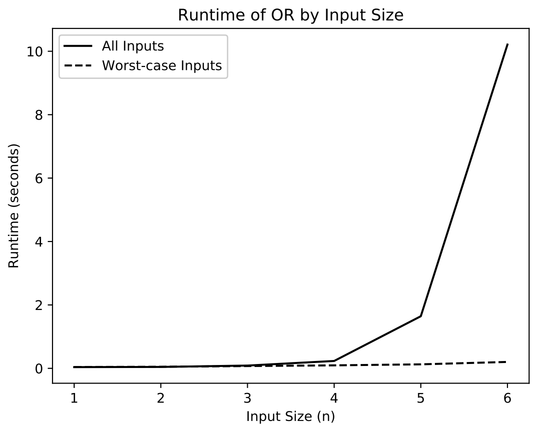

Fig. 3. Runtime of SDP solver by size of input strings.

Recall that the size of the input to the implementation does not have to be all 2^n input bitstrings for input size n. In fact, by choosing different and smaller inputs, we can improve runtime while maintaining accuracy.

In fig. 3, observe that the runtime of the implementation on OR with all inputs increases exponentially with the input size n. (Since the number of inputs grows exponentially, it follows that the size of the problem grows exponentially, too.) As a note, our runtime results are presented as the result of timing our program once. If we were trying to make an argument about the absolute runtime of our algorithm, it would've been better to show a distribution. We want to highlight the asymptotic behavior of runtime which is presented here and consistent between runs of our program. The bottleneck in the implementation is the eigenvalue decomposition of the current solution at each step of the iteration.

If we limit the inputs to the ‘worst-case‘ bitstrings we observe much lower runtime, as shown in 3. It is in general very difficult to find the worst-case inputs and OR is a special case. When searching for at least one 1, the most difficult bitstrings to search are those with a single 1. So, by supplying only the worst-case inputs that have at most one 1, we can maintain the same level of accuracy while greatly reducing runtime. We know of no other Boolean functions with similarly intuitive worst-case sets of inputs.

4 CONCLUSION

Our algorithm correctly identifies the optimal quantum query complexity for the Boolean functions we are familiar with, as well as produces valid span programs for each function we have tested. Our aim is that this program could be a tool in identifying and implementing functions and algorithms with quantum speed-ups, which is an important area of quantum computing research.

While the SDP solution doesn't necessarily provide an immediate use to researchers, as we are unfamiliar with Boolean functions with known optimal quantum query complexity, we hope that it can be used as an aid for verifying existing understand. Furthermore, our production of the query optimal span program will hopefully prove helpful in the development of optimal quantum algorithms and gaining intuition into optimal quantum algorithms.

Although there is much our program has accomplished, there are several possible avenues for future work. Additional functions with worst-case inputs may be explored to reduce runtime as seen with OR. We could also expand the class of functions our algorithm can address as Reichardt's proofs demonstrate that a similar approach can be used for functions that map to any finite set of outputs. Once the SDP is solved, it could be helpful to then have a feature to identify the best polynomial fit (e.g. squareroot or linear) toautomatically categorize the asymptotic behavior of a given Boolean function. Our solver currently solver each bitstring length and presents the results visually, but the asymptotic behavior could be classified algorithmicallty. Finally, further optimizations in the eigenvalue decomposition step of our iteration can be made to further reduce runtime.

ACKNOWLEDGMENTS

We would like to thank Professors Shelby Kimmel and Philip Caplan for their invaluable help and support.

REFERENCES

M Andersen, Joachim Dahl, and Lieven Vandenberghe. 2013. CVXOPT: A Python package for convex optimization. *abel. ee. ucla. edu/cvxopt* (2013).

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press.

Andrew M. Childs. [n. d.].

Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. *28th Annual ACM Symposium on the Theory of Computing*.

Ben W Reichardt. 2009. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 544–551.

SDPA-Author. 2013. CVXOPT: A Python package for convex optimization. *abel. ee. ucla. edu/cvxopt* (2013).

Zaiwen Wen, Donald Goldfarb, and Wotao Yin. 2010. Alternating direction augmented Lagrangian methods for semidefinite programming. *Mathematical Programming Computation* 2, 3-4 (2010), 203–230.