Span-Program-based Quantum Algorithm for Evaluating Formulas

Ben W. Reichardt[*]
California Institute of Technology
breic@caltech.edu

Robert Špalek[†]
Google Inc.
spalek@google.com

ABSTRACT

We give a quantum algorithm for evaluating formulas over an extended gate set, including all two- and three-bit binary gates (e.g., NAND, 3-majority). The algorithm is optimal on read-once formulas for which each gate's inputs are balanced in a certain sense.

The main new tool is a correspondence between a classical linear-algebraic model of computation, "span programs," and weighted bipartite graphs. A span program's evaluation corresponds to an eigenvalue-zero eigenvector of the associated graph. A quantum computer can therefore evaluate the span program by applying spectral estimation to the graph.

For example, the classical complexity of evaluating the balanced ternary majority formula is unknown, and the natural generalization of randomized alpha-beta pruning is known to be suboptimal. In contrast, our algorithm generalizes the optimal quantum AND-OR formula evaluation algorithm and is optimal for evaluating the balanced ternary majority formula.

Categories and Subject Descriptors

F.1.2 [**Computation by Abstract Devices**]: Modes of Computation; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Theory

Keywords

Span programs, quantum computing, quantum algorithms, quantum walks, quantum phase estimation, quantum adversary bound, formula evaluation, balanced ternary majority formula, gadget graphs, spectral analysis

[*]Institute for Quantum Information. Supported by NSF Grants CCF-0524828 and PHY-0456720, and by ARO Grant W911NF-05-1-0294.

[†]Work conducted while at the University of California, Berkeley, supported by NSF Grant CCF-0524837 and ARO Grant DAAD 19-03-1-0082.

1. INTRODUCTION

A formula φ on gate set $\mathcal{S}$ and of size N is a tree with N leaves, such that each internal node is a gate from $\mathcal{S}$ on its children. The read-once formula evaluation problem is to evaluate $\varphi(x)$ given oracle access to the input string $x = x_1 x_2 \dots x_N$. An optimal, $O(\sqrt{N})$-query quantum algorithm is known to evaluate "approximately balanced" formulas over the gates $\mathcal{S} = \{\text{AND, OR, NOT}\}$ [4]. We extend the gate set $\mathcal{S}$. We develop an optimal quantum algorithm for evaluating balanced, read-once formulas over a gate set $\mathcal{S}$ that includes arbitrary three-bit gates, as well as bounded fan-in EQUAL gates and bounded-size $\{\text{AND, OR, NOT, PARITY}\}$ formulas considered as single gates. The correct notion of "balanced" for a formula including different kinds of gates turns out to be "adversary-balanced," meaning that the inputs to a gate must have exactly equal adversary lower bounds. The definition of "adversary-balanced" formulas also includes as a special case layered formulas in which all gates at a given depth from the root are of the same type.

The idea of our algorithm is to consider a weighted graph $G(\varphi)$ obtained by replacing each gate of the formula φ with a small gadget subgraph, and possibly also duplicating subformulas. Figure 1 has several examples. We relate the evaluation of φ to the presence or absence of small-eigenvalue eigenvectors of the weighted adjacency matrix $A_{G(\varphi)}$ that are supported on the root vertex of $G(\varphi)$. The quantum algorithm runs spectral estimation to either detect these eigenvectors or not, and therefore to evaluate φ.

As a special case, for example, our algorithm implies:

Theorem 1.1 *A balanced ternary majority* (MAJ_3) *formula of depth d, on $N = 3^d$ inputs, can be evaluated by a quantum algorithm with bounded error using $O(2^d)$ oracle queries, which is optimal.*

The classical complexity of evaluating this formula is known only to lie between $\Omega((7/3)^d)$ and $o((8/3)^d)$, and the previous best quantum algorithm, from [4], used $O(\sqrt{5}^d)$ queries.

The graph gadgets themselves are derived from "span programs" [19]. Span programs have been used in classical complexity theory to prove lower bounds on formula size [19, 5] and monotone span programs are related to linear secret-sharing schemes [8]. (Most, though not all [1], applications are over finite fields, whereas we use the definition over **C**.) We will only use compositions of constant-size span programs, but it is interesting to speculate that larger span programs could directly give useful new quantum algorithms.

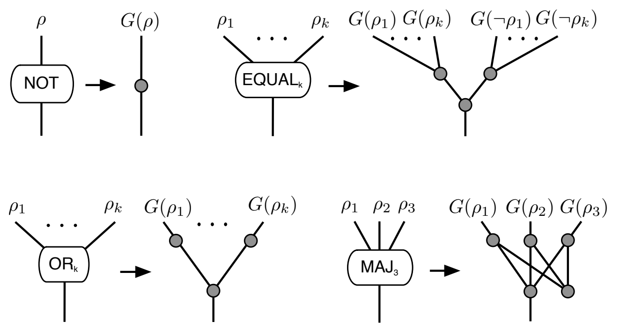


Figure 1: To convert a formula φ to the corresponding graph $G(\varphi)$, we recursively apply substitution rules starting at the root to convert each gate into a gadget subgraph. Some of the rules are shown here, except with the edge weights not indicated. The dangling edges at the top and bottom of each gadget are the input edges and output edge, respectively. To compose two gates, the output edge of one is identified with an input edge of the next.

Classical and quantum background.

The formula evaluation problem has been well-studied in the classical computer model. Classically, the case $\mathcal{S} = \{\text{NAND}\}$ is best understood. A formula with only NAND gates is equivalent to one with alternating levels of AND and OR gates, a so-called "AND-OR formula," also known as a two-player game tree. One can compute the value of a balanced binary AND-OR formula with zero error in expected time $O(N^{\log_2[(1+\sqrt{33})/4]}) = O(N^{0.754})$ [26, 24], and this is optimal even for bounded-error algorithms [25]. However, the complexity of evaluating balanced AND-OR formulas grows with the degree of the gates. For example, in the extreme case of a single OR gate of degree N, the complexity is $\Theta(N)$. The complexity of evaluating AND-OR formulas that are not "well-balanced" is unknown.

If we allow the use of a quantum computer with coherent oracle access to the input, however, then the situation is much simpler; between $\Omega(\sqrt{N})$ and $N^{\frac{1}{2}+o(1)}$ queries are necessary and sufficient to evaluate *any* {AND, OR, NOT} formula with bounded error. In one extreme case, Grover search [14, 15] evaluates an OR gate of degree N using $O(\sqrt{N})$ oracle queries and $O(\sqrt{N}\log\log N)$ time. In the other extreme case, Farhi, Goldstone and Gutmann recently devised a breakthrough algorithm for evaluating the depth-$\log_2 N$ balanced binary AND-OR formula in $O(\sqrt{N})$ time in the unconventional Hamiltonian oracle model [12]. Ambainis [3] improved this to $O(\sqrt{N})$-queries in the standard query model. Childs, Reichardt, Špalek and Zhang [9] gave an $O(\sqrt{N})$-query algorithm for evaluating balanced or "approximately balanced" formulas, and extended the algorithm to arbitrary {AND, OR, NOT} formulas with $N^{\frac{1}{2}+o(1)}$ queries, and also $N^{\frac{1}{2}+o(1)}$ time after a preprocessing step. (Ref. [4] contains the merged results of [3, 9].)

This paper shows other nice features of the formula evaluation problem in the quantum computer model. Classically, with the exception of {NAND}, {NOR} and a few trivial cases like {PARITY}, most gate sets are poorly understood. In 1986, Boppana asked the complexity of evaluating the balanced, depth-d ternary majority (MAJ$_3$) function [24], and today the complexity is only known to lie between $\Omega((7/3)^d)$ and $O((2.655\ldots)^d)$ [18]. In particular, the naïve generalization of randomized alpha-beta pruning—recursively evaluate two random immediate subformulas and then the third if they disagree—runs in expected time $O((8/3)^d)$ and is suboptimal. This suggests that the balanced ternary majority function is significantly different from the balanced k-ary NAND function, for which randomized alpha-beta pruning is known to be optimal. In contrast, we show that the optimal quantum algorithm of [9] does extend to give an optimal $O(2^d)$-query algorithm for evaluating the balanced ternary majority formula. Moreover, the algorithm also generalizes to a significantly larger gate set $\mathcal{S}$.

Organization.

We introduce span programs and explain their correspondence to weighted bipartite graphs in Section 2. The correspondence involves considering parts of a span program P as the weighted adjacency matrix for a corresponding graph G_P. We prove that the eigenvalue-zero eigenvectors of this adjacency matrix evaluate P (Theorem 2.5). This theorem provides useful intuition.

We develop a quantitative version of Theorem 2.5 in Section 3. We lower-bound the overlap of the eigenvalue-zero eigenvector with a known starting state. This lower-bound will imply completeness of our quantum algorithm. To show soundness of the algorithm, we also analyze small-eigenvalue eigenvectors in order to prove a spectral gap around zero. Essentially, we solve the eigenvalue equations in terms of the eigenvalue λ, and expand a series around $\lambda = 0$. The results for small-eigenvalue and eigenvalue-zero eigenvectors are closely related, and we unify them using a measure we term "span program witness size." In this extended abstract, we only sketch the proofs in Section 3, leaving the details to our extended arXiv preprint [23].

Section 4 applies the span program framework to the formula evaluation problem. Theorem 4.7 is our general result, an optimal quantum algorithm for evaluating formulas that are over the gate set $\mathcal{S}$ of Definition 4.1, and that are adversary-balanced (Definition 4.5). The proof of Theorem 4.7 has two parts. First, in Section 4.2, we display an optimal span program for each of the gates in $\mathcal{S}$. Second, we compose the span programs for the individual gates to obtain a span program for the full formula φ. This is equivalent to joining together the gadget graphs described in Figure 1 to obtain a graph $G(\varphi)$. We combine the spectral analyses of the individual span programs to analyze the spectrum of $G(\varphi)$ (Theorem 4.14). This analysis straightforwardly leads to a quantum algorithm based on phase estimation of a quantum walk on $G(\varphi)$, in Section 4.4.

Section 5 concludes with a discussion of some extensions to the algorithm.

2. SPAN PROGRAMS AND EIGENVALUE-ZERO GRAPH EIGENVECTORS

A span program P is a certain linear-algebraic way of specifying a function f_P. For details on span programs applied in classical complexity theory, we can still recommend the original reference [19] as well as, e.g., the more recent [13].

Definition 2.1 (Span program) *A span program P consists of a nonzero "target" vector t in a vector space over $\mathbf{C}$, together with "grouped input" vectors $\{v_j : j \in J\}$. Each v_j is labeled with a subset X_j of the literals $\{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$. To P corresponds a boolean function $f_P : \{0,1\}^n \to \{0,1\}$; defined by $f_P(x) = 1$ (i.e., true) if and only if there exists a linear combination $\sum_j a_j v_j = t$ such that $a_j = 0$ if any of the literals in X_j evaluates to zero (i.e., false).*

Example 2.2 *For example, the span program*

$$X_J = (\{x_1\} \quad \{x_2\} \quad \{x_3\} \;)$$
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 1 & e^{2\pi i/2} & e^{-2\pi i/3} \end{pmatrix}$$

computes the MAJ_3 *function. Indeed, at least two of the v_j must have nonzero coefficient in any linear combination equaling the target t. Of course, the second row of $(v_1 \quad v_2 \quad v_3)$ could be any $(\alpha \quad \beta \quad \gamma)$ with α, β, γ distinct and nonzero, and the span program would still compute* MAJ_3*. This specific setting is used to optimize the running time of the quantum algorithm (Claim 4.9).*

In this section, we will show that by viewing a span program P as the weighted adjacency matrix A_{G_P} of a certain graph G_P, the true/false evaluation of P on input x corresponds to the existence or nonexistence of an eigenvalue-zero eigenvector of $A_{G_P}(x)$ supported on a distinguished output node (Theorem 2.5).

In turn, this will imply that writing a span program P for a function f immediately gives a quantum algorithm for evaluating f, or for evaluating formulas including f as a gate (Section 4). The algorithm works by spectral estimation on $A_{G_P}(x)$. Its running time depends on the span program's "witness size" (Section 3). For example, if $f_P(x)$ is true, then the witness size is essentially the shortest squared length of any witness vector $(a_j)_{j \in J}$ in Definition 2.1.

Remark 2.3 *Let us clarify a few points in Definition 2.1.*

1. It is convenient, but nonstandard, to allow grouped inputs, i.e., literal subsets X_j possibly with $|X_j| > 1$, instead of just single literals, to label the columns. A grouped input j can be thought of as evaluating the AND of all literals in X_j. A span program P with some $|X_j| > 1$ can be expanded out so that all $|X_j| \leq 1$, without increasing $\sum_j |X_j|$, known as the size *of P.*

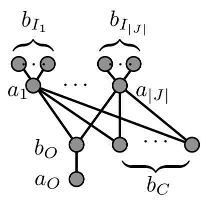
2. It is sometimes convenient to allow $X_j = \emptyset$. In this case, vector v_j is always available to use in the linear combination; grouped input j evaluates to true always. However, such vectors can be eliminated from P without increasing the size or changing t [19, Theorem 7].

3. By a basis change, one can always adjust the target vector t to $(1, 0, 0, \ldots, 0)$.

2.1 Span program as an adjacency matrix

A span program P with target vector $t = (1, 0, \ldots, 0)$ corresponds to a certain weighted bipartite graph.

Notation: For an index sequence $H = (h_1, \ldots, h_{|H|})$ and a set of variables $\{a_h\}$, let $a_H = (a_{h_1}, \ldots, a_{h_{|H|}})$. For example, v_J denotes the sequence of grouped input vectors. It will be convenient to define several more index sequences: O ("output"), C ("constraints") and I ("inputs"). Let O and C together index the coordinates of the vector space, with



$$A_{G_P} = \begin{array}{c} \begin{array}{cc} a_O & \overbrace{\;\;a_J\;\;} \end{array} \\ \left(\begin{array}{c|c} 1 & A_{OJ} \\ \begin{smallmatrix}0\\ \vdots \\ 0\end{smallmatrix} & A_{CJ} \\ \hline \begin{smallmatrix}0\\ \vdots \\ 0\end{smallmatrix} & A_{IJ} \end{array}\right) \begin{array}{l} b_O \\ \}b_C \\ \}b_I \end{array} \end{array}$$

Figure 2: The bipartite graph G_P corresponding to span program P (the output edge is (a_O, b_O), while the grouped inputs are $a_1, \ldots, a_{|J|}$).

$O = \{1\}$ being the first coordinate, and C the remainder. Let I_j index X_j for each $j \in J$, and let $I = \bigcup_{j \in J} I_j$ a disjoint union. (Since $|I_j| = |X_j|$, $|I| = \mathrm{size}(P)$.)

We will construct a graph G_P on $|I| + |J| + |C| + 2|O|$ vertices. Writing the grouped input vectors out as the columns of a matrix, let $\left(\begin{smallmatrix} A_{OJ} \\ A_{CJ} \end{smallmatrix}\right) = \sum_{j \in J} |v_j\rangle\langle j|$; A_{OJ} is a $1 \times |J|$ matrix row, and A_{CJ} is a $|C| \times |J|$ matrix. Let $A_{IJ} = \sum_{j \in J, i \in I_j} |i\rangle\langle j|$; A_{IJ} encodes P's grouped inputs. Now consider the bipartite graph G_P of Figure 2, the upper right block of whose weighted Hermitian adjacency matrix is A_{G_P}. (The adjacency matrix is block off-diagonal because the graph is bipartite.) The edges (a_j, b_i) for $j \in J$ and $i \in I_j$ are "input edges," while (a_O, b_O) is the "output edge." The input and output edges all have weight one. The weights of edges (b_O, a_j) for $j \in J$ are given by A_{OJ} (the first coordinates of the grouped input vectors v_J), while the weights of edges (b_c, a_j) for $c \in C, j \in J$ are given by A_{CJ} (the remaining coordinates of v_J).

Example 2.4 *For the* MAJ_3 *span program of Example 2.2, $|C| = 1$, $|I| = |J| = 3$, the graph G_P is shown in Figure 1, and the matrix A_{G_P} is*

$$\left(\begin{array}{c|ccc} 1 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 0 & 1 & e^{2\pi i/3} & e^{-2\pi i/3} \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right) .$$

2.2 Eigenvalue-zero eigenvectors of the span program adjacency matrix

Theorem 2.5 *For an input $x \in \{0,1\}^n$, define a weighted graph $G_P(x)$ by deleting from G_P the edges (a_j, b_i) if the ith literal in X_j is true. Consider all the eigenvalue-zero eigenvector equations of the weighted adjacency matrix $A_{G_P(x)}$, except for the constraint at a_O. These equations have a solution with support on vertex a_O if and only if $f_P(x) = 1$, and have a solution with support on b_O if and only if $f_P(x) = 0$.*

Proof. Notation: Use a_j, b_i, b_c, a_O, b_O to denote coefficients of a vector on the vertices of G_P. Let $A_{IJ}(x)$ include only edges to false inputs, i.e., $A_{IJ}(x) = \sum_{j \in J, \text{false } i \in I_j} |i\rangle\langle j|$.

The eigenvalue-λ eigenvector equations of $A_{G_P(x)}$ are

$$\lambda b_O = a_O + A_{OJ} a_J \tag{2.1a}$$
$$\lambda b_C = A_{CJ} a_J \tag{2.1b}$$
$$\lambda b_I = A_{IJ}(x) a_J \tag{2.1c}$$
$$\lambda a_O = b_O \tag{2.1d}$$
$$\lambda a_J = A_{OJ}^\dagger b_O + A_{CJ}^\dagger b_C + A_{IJ}^\dagger(x) b_I \tag{2.1e}$$

At $\lambda = 0$, these equations say that for each vertex, the weighted sum of the adjacent vertices' coefficients must be zero. We are looking for solutions satisfying all these equations except possibly Eq. (2.1d). Since the graph is bipartite, at $\lambda = 0$ the a coefficients do not interact with the b coefficients. In particular, Eqs. (2.1d,e) (resp. 2.1a-c) can always be satisfied by setting the b (resp. a) coefficients to zero.

By scaling, there is a solution with nonzero a_O iff there is a solution with $a_O = -1$. Then Eqs. (2.1a,b) are equivalent to $t = \begin{pmatrix} A_{OJ} \\ A_{CJ} \end{pmatrix} a_J = \sum_j a_j v_j$. Moreover, Eq. (2.1c) implies that a_j can be nonzero only if grouped input j is true. (If X_j includes any false inputs, then $A_{IJ}(x)|j\rangle \neq 0$, so $a_j = 0$.) These conditions are the same as those in Definition 2.1.

Next, we argue that there is a solution of Eq. (2.1e) with $\lambda = 0$ and $b_O = 1$ if and only if $f_P(x) = 0$. Indeed,

$$f_P(x) = 0 \Leftrightarrow t \notin \mathrm{Span}\{v_j : j \text{ true}\} \Leftrightarrow t \notin \mathrm{Range}\Big[\begin{pmatrix} A_{OJ} \\ A_{CJ} \end{pmatrix} \Pi\Big]$$

where $\Pi = \sum_{\text{true } j} |j\rangle\langle j|$. In turn, this holds iff there is a vector w orthogonal to the range and having inner product one with t—precisely constraint (2.1e) with $w = \begin{pmatrix} b_O \\ b_C \end{pmatrix}$. □

Remark 2.6 *By Theorem 2.5, we can think of the graph G_P as giving a "dual-rail" encoding of the function f_P: there is a $\lambda = 0$ eigenvector of $G_P(x)$ supported on a_O if and only if $f_P(x) = 1$, and there is one supported on b_O iff $f_P(x) = 0$. This justifies calling edge (a_O, b_O) the output edge of G_P.*

2.3 Dual span program

A span program P immediately gives a dual span program, denoted $P^\dagger$, such that $f_{P^\dagger}(x) = \neg f_P(x)$ for all $x \in \{0,1\}^n$. For our purposes, though, it suffices to define a NOT gate graph gadget to allow negation of subformulas.

Definition 2.7 (NOT gate gadget) *Implement a NOT gate $x \mapsto \overline{x}$ as two weight-one edges connected (Figure 1). The edge (a_i, b_i) is the input edge, while (a_O, b_O) is the output edge. The middle vertex $a_i = b_O$ is shared.*

At $\lambda = 0$, the coefficient on a_O is minus that on b_i, and $a_i = b_O$ by definition. Therefore, this gadget complements the dual rail encoding of Theorem 2.5.

The NOT gate gadget of Definition 2.7 can be used to define a dual span program $P^\dagger$ by complementing the output and all inputs with NOT gates, and also complementing all input literals in the sets X_J. Since it is not essential here, we leave the formal definition as an exercise. Alternative constructions of dual programs are given in [23, 11, 21].

Example 2.8 *For distinct, nonzero α, β, γ, the span program*

$$X_J = (\ \emptyset\ \ \emptyset\ \ \{\overline{x_1}\}\ \{\overline{x_2}\}\ \{\overline{x_3}\})$$
$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & \alpha & 1 & 0 & 0 \\ 1 & \beta & 0 & 1 & 0 \\ 1 & \gamma & 0 & 0 & 1 \end{pmatrix}$$

computes $\neg\,\mathrm{MAJ3}(x_1, x_2, x_3)$. It was constructed, by adding NOT gate gadgets, as the dual to the span program in Example 2.2, up to choice of weights.

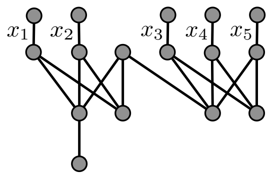


Figure 3: Graph for $\mathrm{MAJ3}(x_1, x_2, \mathrm{MAJ3}(x_3, x_4, x_5))$, with input edges labeled by the associated literals.

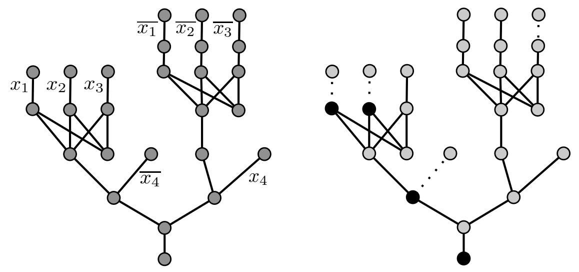


Figure 4: Graph for $\mathrm{MAJ3}(x_1, x_2, x_3) \oplus x_4$, and its evaluation on input $x = 1100$.

2.4 Span program composition

Definition 2.9 (Composed graph and span program) *Consider span program Q on $\{0,1\}^n$ and programs Q_i, $i \in [n] \equiv \{1, \ldots, n\}$, with corresponding graphs G_Q and G_{Q_i}. The composed graph is defined by identifying the input edges of G_Q with the output edges of copies of the other graphs. If an edge corresponds to input literal x_i, then identify that edge with the output edge of a copy of G_{Q_i}; and if an edge corresponds to $\overline{x_i}$, then insert a NOT gate gadget (i.e., an extra vertex, as in Definition 2.7) before a copy of G_{Q_i}. The composed span program, denoted $Q \circ Q_{[n]}$, is the program corresponding to the composed graph (i.e., $G_{Q \circ Q_{[n]}}$ is the composed graph). Thus $f_{Q \circ Q_{[n]}} = f_Q \circ f_{Q_{[n]}}$.*

Definition 2.10 (Formula graph and span program) *Given span programs for each gate in a formula φ, we define the span program $P(\varphi)$ as their composition according to the formula. Let $G(\varphi)$ be the composed graph, $G(\varphi) = G_{P(\varphi)}$.*

Example 2.11 *For example, the span program*

$$X_J = (\{x_1\}\ \{x_2\}\ \emptyset\ \{x_3\}\ \{x_4\}\ \{x_5\})$$
$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ \alpha & \beta & \gamma & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & \alpha & \beta & \gamma \end{pmatrix}$$

is a composed span program that computes the function $\mathrm{MAJ3}(x_1, x_2, \mathrm{MAJ3}(x_3, x_4, x_5))$, provided α, β, γ are distinct and nonzero. (See Example 2.2.) Figure 3 shows the associated composed graph.

Example 2.12 (Duplicating and negating inputs) *To the left in Figure 4 is the composed graph for the formula $\mathrm{MAJ3}(x_1, x_2, x_3) \oplus x_4 = \mathrm{EQUAL}_2(\mathrm{MAJ3}(x_{[3]}), \overline{x_4})$, obtained using the substitution rules of Figure 1. (A span program for*

PARITY will be given in Lemma 4.12.) Note that we are effectively negating some inputs twice, by putting NOT gate gadgets below the negated literals $\overline{x_1}, \overline{x_2}, \overline{x_3}$. This is of course redundant, and is only done to maintain the strict correspondence of graphs to span programs, as in Example 2.8, by keeping the input vertices b_I at odd distances from a_O.

To the right is the same graph evaluated on input $x = 1100$, i.e., with edges to true literals deleted. Since the formula evaluates to true, Theorem 2.5 promises that there is a $\lambda = 0$ eigenvector supported on a_O. In this case, that eigenvector is unique. It has support on the black vertices.

3. SPAN PROGRAM WITNESS SIZE

In Section 2, we established that by converting a formula φ into a weighted graph $G(\varphi)$, by replacing each gate with a gadget subgraph coming from a span program, the eigenvalue-zero eigenvectors of the graph effectively evaluate φ. The dual-rail encoding of $\varphi(x) = f_{P(\varphi)}(x)$ promised by Theorem 2.5 will suffice to give a phase-estimation-based quantum algorithm for evaluating φ. The goal of this section is to make Theorem 2.5 more quantitative, which will enable us to analyze the algorithm's running time.

In particular, we will lower-bound the achievable squared support on either a_O or b_O of a unit-normalized $\lambda = 0$ eigenvector. (This will enable the algorithm to detect if $\varphi(x) = 1$ by starting a quantum walk at a_O; if $\varphi(x) = 1$, then $|a_O\rangle$ will have large overlap with the $\lambda = 0$ eigenvector.)

We also study eigenvalue-λ eigenvectors of $G_{P(\varphi)}(x)$, for $|\lambda| \neq 0$ sufficiently small. At small enough eigenvalues, the dual-rail encoding property of Theorem 2.5 still holds, in a different fashion. Note that since the graph is bipartite, we may take $\lambda > 0$ without loss of generality. For small enough $\lambda > 0$, it will turn out that the function evaluation corresponds to the output *ratio* $r_O \equiv a_O/b_O$. If $f_P(x) = 1$, then r_O is large and negative, roughly of order $-1/\lambda$. If $f_P(x) = 0$, then r_O is small and positive, roughly of order λ. (Ultimately, the point of this analysis is to show that if the formula evaluates to false, then there do not exist any eigenvalue-λ eigenvectors supported on a_O for small enough $|\lambda| \neq 0$. This spectral gap will prevent the algorithm from outputting false positives.)

Consider a span program P. Let us generalize the setting of Theorem 2.5 to allow P's inputs to be themselves span programs, as in Definition 2.9. Assume that for some x, every $\lambda \in [0, \Lambda)$ and each input $i \in I$, we have constructed unit-normalized states $|\psi_i(\lambda)\rangle$ satisfying the eigenvalue-λ constraints for all the ith subgraph's vertices except a_i.

Definition 3.1 (Subformula complexity) *At $\lambda = 0$, for each input $i \in I_j$, let γ_i lower-bound $|\psi_i\rangle$'s squared support on either a_j or b_i, depending on whether the input evaluates to true or false, respectively.*

For $\lambda > 0$, assume that the coefficients of $|\psi_i\rangle$ along the ith input edge are nonzero, and let $r_i = a_j/b_i$ be their ratio. If the literal associated to input i evaluates to false, then let $s_i = r_i/\lambda$; if it is true, then let $s_i = -1/(r_i\lambda)$. Assume that $s_i > 0$ and let $S_i \geq s_i$ for each i.

For an input $i \in I$, its subformula complexity *is*

$$\sigma_i = \max_x \max\left\{1/\gamma_i, \max_{\lambda\in(0,\Lambda)} S_i\right\} \ . \tag{3.1}$$

For example, if σ_i is small, then $|\psi_i(0)\rangle$ has large support on either a_i or b_i. In general, $\sigma_i \geq 1$. If input i corresponds to a literal and not the output edge of another span program, then $\sigma_i = 1$.

We construct a normalized state $|\psi_O(\lambda)\rangle$ that satisfies all the eigenvalue-λ eigenvector constraints of the composed graph, except at a_O. We construct $|\psi_O\rangle$ by putting together the scaled $|\psi_i\rangle$'s and also assigning coefficients to the vertices in G_P. Similarly to Eq. (3.1), define

$$\sigma_O(x) = \max\left\{1/\gamma_O, \max_{\lambda\in(0,\Lambda)} s_O\right\} \ , \tag{3.2}$$

where γ_O is the squared support of $|\psi_O(0)\rangle$ on a_O or b_O, and s_O is r_O/λ or $-1/(r_O\lambda)$ for $\lambda > 0$. We will relate $\sigma_O = \max_x \sigma_O(x)$ to the input complexities σ_I (Theorem 3.6).

First of all, notice that if $|I_j| > 1$, then several of the input subgraphs share the vertex a_j. They must be scaled so that their coefficients at a_j all match, motivating the following definition. (Recall that grouped input j evaluates to true iff all inputs in I_j are true.)

Definition 3.2 *The* grouped input complexity *of $j \in J$ is*

$$\tilde\sigma_j(x) = \begin{cases} \max\left\{\sum_{i\in I_j}\sigma_i, 1\right\} & \textit{if } j \textit{ is true} \\ \left(\sum_{\textit{false } i\,\in\, I_j}\sigma_i^{-1}\right)^{-1} & \textit{otherwise} \end{cases} \tag{3.3}$$

When j is false, some input $i \in I_j$ is false, so the coefficient at a_j must be set to zero at $\lambda = 0$. However, for each false $i \in I_j$, $|\psi_i\rangle$ can be scaled arbitrarily. The definition in Eq. (3.3) comes from choosing scale factors f_i in order to maximize the sum of the scaled coefficients on the vertices b_i, under the constraint that the total norm be one, $\sum_{i\in I_j}|f_i|^2 = 1$.

A few more definitions are needed to state Theorem 3.6.

Definition 3.3 (Matrix notations) *For a given input x, let $\Pi = \sum_{\textit{true } j}|j\rangle\langle j|$ a projection onto the true grouped inputs, $\overline{\Pi} = 1 - \Pi$, and $S = \sum_j \sqrt{\tilde\sigma_j(x)}|j\rangle\langle j|$, i.e., a diagonal matrix of the grouped input complexity square roots. Let $A = \begin{pmatrix} A_{OJ} \\ A_{CJ}\end{pmatrix} = \sum_j |v_j\rangle\langle j|$ with columns the vectors $|v_j\rangle$.*

Definition 3.4 (Moore-Penrose pseudoinverse) *For a matrix M, let M^+ denote its Moore-Penrose pseudoinverse. If the singular-value decomposition of M is $M = \sum_k m_k|k\rangle\langle k'|$ with all $m_k \neq 0$ and for sets of orthonormal vectors $\{|k\rangle\}$ and $\{|k'\rangle\}$, then $M^+ = \sum_k m_k^{-1}|k'\rangle\langle k|$. Note that $MM^+ = \sum_k |k\rangle\langle k|$ is the projection onto M's range.*

Definition 3.5 (Span program witness size) *For span program P and input subformula complexities σ_I, the witness size of P on input x, $\mathrm{wsize}(P, x)$, is defined as follows:*

- *If $f_P(x) = 1$, then $|t\rangle \in \mathrm{Range}(A\Pi)$, so there is a witness $|w\rangle$ of length $|J|$ satisfying $A\Pi S^{-1}|w\rangle = |t\rangle$. Then the witness size is the minimum squared length of any such witness:*

$$\begin{aligned}\mathrm{wsize}(P, x) &= \min_{|w\rangle : A\Pi S^{-1}|w\rangle = |t\rangle} \||w\rangle\|^2 \\ &= \|(A\Pi S^{-1})^+|t\rangle\|^2 \ .\end{aligned} \tag{3.4}$$

- *If $f_P(x) = 0$, then $|t\rangle \notin \mathrm{Range}(A\Pi)$. Therefore there is a witness $|w'\rangle$ of length $|C| + 1$ satisfying $\langle t|w'\rangle = 1$ and*

$\Pi A^\dagger |w'\rangle = 0$. Then

$$\mathrm{wsize}(P,x) = \min_{\substack{|w'\rangle : \langle t|w'\rangle = 1 \\ \Pi A^\dagger |w'\rangle = 0}} \| S A^\dagger |w'\rangle \|^2 \tag{3.5}$$
$$= \big\| \big(1 + (\overline{\Pi}(AS)^+ AS - 1)^+ \Pi\big)(AS)^+ |t\rangle \big\|^{-2} \ ,$$

the inverse squared length of the projection of $(AS)^+|t\rangle$ onto the intersection of $\overline{\Pi}$ and $\mathrm{Range}(SA^\dagger)$.

The witness size of P is $\mathrm{wsize}(P) = \max_x \mathrm{wsize}(P,x)$. By $|w_x\rangle$, resp. $|w'_x\rangle$, we denote an optimal witness for input x achieving the minimum in Eq. (3.4), resp. (3.5).

The span program witness size is easily computed on any given input x. Now our main result is:

Theorem 3.6 *Consider a constant span program P. Assume that $\Lambda \sigma_i \leq \epsilon$ for a small enough constant $\epsilon > 0$ to be determined and for all $i \in I$. Assume also that $(\max_{i \in I} \sigma_i)/(\min_{i \in I} \sigma_i) = O(1)$. Let $a \lesssim b$ mean $a \leq$ constant $+ b(1 +$ constant$'|\lambda| \max_i \sigma_i)$. Then*

$$\sigma_O(x) \lesssim \mathrm{wsize}(P,x) \ . \tag{3.6}$$

For $\lambda = 0$, Eq. (3.6) says that the achievable squared magnitude on a_O or b_O of a normalized eigenvalue-zero eigenvector is at least $1/\mathrm{wsize}(P,x)$, up to small controlled terms. For $\lambda > 0$, Eq. (3.6) says that the ratio $r_O = a_O/b_O$ is either in $(0, \mathrm{wsize}(P,x)\lambda]$ or $(-\infty, -1/(\mathrm{wsize}(P,x)\lambda)]$, up to small controlled terms, depending on whether $f_P(x)$ is false or true. Note that $\mathrm{wsize}(P,x)$ is monotone in S and thus also in all S_i, and therefore we get a valid bound on $\sigma_O(x)$ even without knowing the actual values of $s_i \leq S_i$.

Proof sketch of Theorem 3.6. At $\lambda = 0$, the proof of Theorem 3.6 is the same as that of Theorem 2.5, except scaling the inputs so as to maximize the squared magnitude on a_O or b_O. This maximization problem is essentially the same as the problems stated in Definition 3.5 (up to additive constants). The explicit expressions for the solutions follow by geometry.

For $\lambda > 0$, we solve the eigenvalue equations (2.1a-c) by inverting a matrix and applying the Woodbury formula. We argue that all inverses exist in the given range of λ. We obtain

$$r_O = a_O/b_O = \lambda + \langle o|\tilde{R}|o\rangle + \lambda \langle o|\tilde{R} A_{CJ}^\dagger X^{-1} A_{CJ} \tilde{R}|o\rangle \ ,$$

where $|o\rangle = A_{OJ}^\dagger$, $\tilde{R} = -\frac{1}{\lambda} S^{-2}\Pi + \lambda S^2 \overline{\Pi}$ and $X = A_{CJ} S^{-2} \Pi A_{CJ}^\dagger - \lambda^2 A_{CJ} S^2 \overline{\Pi} A_{CJ}^\dagger - \lambda^2$. The largest term in X, $A_{CJ} S^{-2} \Pi A_{CJ}^\dagger$, is only invertible restricted to its range, $\Delta = A_{CJ}\Pi(A_{CJ}\Pi)^+$. Therefore, we compute the Taylor series against λ of the pseudoinverse of $\Delta X \Delta$ and of its Schur complement, $(X/(\Delta X \Delta))$, separately, and then recombine them. The lowest-order term in the solution again corresponds to Definition 3.5 (if $f_P(x)$ is false, the $1/\lambda$ term is zero), and we bound the higher-order terms. □

Remark 3.7 *In case $f_P(x) = 0$, $A^\dagger|w'\rangle$ appears also in the "canonical form" of P [19].*

The above analysis of span programs does not apply to the NOT gate, because the ability to complement inputs was assumed in Definition 2.1. Implementing the NOT gate $x \mapsto \overline{x}$ with a span program on the literal $\overline{x}$ would be circular. Therefore we give a separate analysis.

Lemma 3.8 (NOT gate) *Consider a NOT gate, implemented as two weight-one edges connected as in Definition 2.7. Assume $|\lambda| \leq 1/(\sqrt{2\sigma_i})$. Then $\sigma_O \lesssim \sigma_i$.*

Proof. *Analysis at $\lambda = 0$.* If the input is true, then γ_i measures the squared support on a_i of a normalized $\lambda = 0$ eigenvector. Then $\gamma_O = \gamma_i$, since $a_i = b_O$ the output vertex. If the input is false, so $b_i = \sqrt{\gamma_i}$, then $b_i + a_O = 0$. Therefore, we simply need to renormalize: $\gamma_O = \gamma_i/(1+\gamma_i)$, or equivalently $\frac{1}{\gamma_O} = \frac{1}{\gamma_i} + 1$.

Analysis for small $\lambda > 0$. We are given $r_i = a_i/b_i$. The eigenvector equation is $b_i + a_O = \lambda a_i = \lambda b_O$. Therefore, $r_O = a_O/b_O = \lambda - 1/r_i$. If the input is false, so $s_i = r_i/\lambda$, then $s_O = -1/(\lambda r_O) = s_i/(1 - \lambda^2 s_i)$. Therefore, $s_i \leq s_O \leq s_i(1 + 2\lambda^2 s_i)$ since $\lambda^2 s_i \leq 1/2$. If the input is true, so $s_i = -1/(\lambda r_i)$, then $s_O = r_O/\lambda = s_i + 1$.

Therefore $\sigma_O \lesssim \sigma_i$ as claimed. Note that w.l.o.g. we may assume there are never two NOT gates in a row in the formula φ, so the additive constants lost do not accumulate. □

4. FORMULA EVALUATION ALGORITHM

In Section 4.1, we specify the gate set $\mathcal{S}$ (Definition 4.1) and define the correct notion of "balance" for a formula that includes different kinds of gates (Definition 4.5). These two definitions allow us to formulate the general statement of our results, Theorem 4.7, of which Theorem 1.1 is a corollary.

In Section 4.2, we present span programs for each of the gates in $\mathcal{S}$ having optimal witness size. In Section 4.3, we plug together the spectral analyses of the individual span programs to obtain a spectral analysis of $G(\varphi)$. Finally, in Section 4.4, we sketch how this implies a quantum algorithm, therefore proving Theorem 4.7.

4.1 General formula evaluation result

Definition 4.1 (Extended gate set $\mathcal{S}$) *Let*

$$\mathcal{S}' = \left\{ \begin{array}{c} \textit{arbitrary 1-, 2-, or 3-bit gates,} \\ O(1)\textit{-fan-in EQUAL gates} \end{array} \right\}$$
$$\mathcal{S} = \left\{ \begin{array}{c} O(1)\textit{-size } \{\textit{AND, OR, NOT, PARITY}\} \\ \textit{read-once formulas composed onto} \\ \textit{the gates from } \mathcal{S}' \end{array} \right\} \tag{4.1}$$

Example 4.2 *The gate set $\mathcal{S}$ includes simple gates like AND, as well as substantially more complicated gates like $\mathrm{MAJ}_3(x_1,x_2,x_3) \wedge (x_4 \oplus x_5 \oplus \cdots \oplus x_{k-1} \oplus (x_k \vee x_{k+1}))$, provided $k = O(1)$. It does not include gates from $\mathcal{S}'$ composed onto gates from $\mathcal{S}$: for example $\mathrm{MAJ}_3(x_1, x_2 \oplus x_3, x_4 \wedge x_5) \notin \mathcal{S}$.*

To define "adversary-balanced" formulas, we need to define the nonnegative-weight quantum adversary bound.

Definition 4.3 (Nonnegative adversary bound) *Let $f : \{0,1\}^k \to \{0,1\}$. Define*

$$\mathrm{Adv}(f) = \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma \circ D_i\|} \ , \tag{4.2}$$

where $\Gamma \circ D_i$ denotes the entrywise matrix product between Γ and D_i a zero-one-valued matrix defined by $\langle x|D_i|y\rangle = 1$ if and only if bitstrings x and y differ in the ith coordinate, for $i \in \{1,\ldots,k\}$. The maximum is over all $2^k \times 2^k$ symmetric matrices Γ with nonnegative entries satisfying $\langle x|\Gamma|y\rangle = 0$ if $f(x) = f(y)$.

The motivation for this definition is that $\mathrm{Adv}(f)$ gives a lower bound on the number of queries to the phase-flip input oracle O_x required to evaluate f on input x.

Theorem 4.4 (Adversary lower bound [2, 7]) *The two-sided ϵ-bounded error quantum query complexity of function f, $Q_\epsilon(f)$, is at least $\frac{1-2\sqrt{\epsilon(1-\epsilon)}}{2}\mathrm{Adv}(f)$.*

To match the lower bound of Theorem 4.4, our goal will be to use $O(\mathrm{Adv}(\varphi))$ queries to evaluate φ.

Definition 4.5 (Adversary-balanced formula) *For a gate g in formula φ, let φ_g denote the subformula of φ rooted at g. Define φ to be adversary-balanced if for every gate g, the adversary lower bounds for its input subformulas are the same; if g has children $h_1, \ldots, h_k$, then $\mathrm{Adv}(\varphi_{h_1}) = \cdots = \mathrm{Adv}(\varphi_{h_k})$.*

To motivate Definition 4.5, we need a version of an adversary composition result [2, 17]:

Theorem 4.6 (Adversary composition [17]) *Let $f = g \circ (h_1, \ldots, h_k)$, where $\mathrm{Adv}(h_1) = \cdots = \mathrm{Adv}(h_k)$ and $\circ$ denotes function composition. Then $\mathrm{Adv}(f) = \mathrm{Adv}(g)\mathrm{Adv}(h_1)$.*

If φ is adversary-balanced, then by Theorem 4.6 $\mathrm{Adv}(\varphi_g)$ is the product of the gate adversary bounds along any non-self-intersecting path χ from g up to an input, $\mathrm{Adv}(\varphi_g) = \prod_{h\in\chi}\mathrm{Adv}(h)$. Note that $\mathrm{Adv}(\neg f) = \mathrm{Adv}(f)$, so NOT gates can be inserted anywhere in an adversary-balanced formula.

The main result of this paper is

Theorem 4.7 (Main result) *There exists a quantum algorithm that evaluates an adversary-balanced formula $\varphi(x)$ over $\mathcal{S}$ using $O(\mathrm{Adv}(\varphi))$ queries to the phase-flip input oracle O_x. After efficient classical preprocessing independent of the input x, and assuming $O(1)$-time coherent access to the preprocessed classical string, the running time of the algorithm is $\mathrm{Adv}(\varphi)(\log \mathrm{Adv}(\varphi))^{O(1)}$.*

From Figure 5, the adversary bound $\mathrm{Adv}(\mathrm{MAJ}_3) = 2$. By Theorem 4.6 the adversary bound for the balanced MAJ_3 formula of depth d is 2^d. Theorem 1.1 is therefore essentially a corollary of Theorem 4.7 (for the balanced MAJ_3 formula, coherent access to a preprocessed classical string is not needed).

4.2 Optimal span programs for gates in $\mathcal{S}$

In this section, we will plug specific span programs into Definition 3.5, in order to prove:

Theorem 4.8 *Let $\mathcal{S}$ be the gate set of Definition 4.1. For every gate $f \in \mathcal{S}$, there exists a span program P computing $f_P = f$, such that the witness size of P (Definition 3.5) on equal input complexities $\sigma_i = 1$ is*

$$\mathrm{wsize}(P) = \mathrm{Adv}(f) \ . \tag{4.3}$$

$\mathrm{Adv}(f)$ is the adversary bound for f (Definition 4.3).

Proof sketch. We analyze five of the fourteen inequivalent binary functions on at most three bits, listed in Figure 5: 0 and x_1 (both trivial), the MAJ_3 gate (Claim 4.9), the k-bit EQUAL_k gate (Claim 4.10), and a certain three-bit function, $g(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$ (Claim 4.11).

For all the remaining gates in $\mathcal{S}$, it suffices to analyze the NOT gate (Lemma 3.8), and OR and PARITY gates on *unbalanced* inputs (Lemma 4.12). That is, we allow σ_1 and σ_2 to be different, with $\sigma_1/\sigma_2, \sigma_2/\sigma_1 = O(1)$. For functions b and b' on disjoint inputs, $\mathrm{Adv}(b \oplus b') = \mathrm{Adv}(b) + \mathrm{Adv}(b')$, and $\mathrm{Adv}(b \vee b') = \sqrt{\mathrm{Adv}(b)^2 + \mathrm{Adv}(b')^2}$ [6, 17]; we obtain matching upper bounds for span program witness size. Then, e.g., the function $\mathrm{EXACT}_{2\text{ of }3}(x_1, x_2, x_3) = \mathrm{MAJ}_3(x_1, x_2, x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$, so Lemma 4.12 implies a span program for $\mathrm{EXACT}_{2\text{ of }3}$ with witness size $\sqrt{7} = \sqrt{\mathrm{wsize}(\mathrm{MAJ}_3)^2 + \mathrm{wsize}(\mathrm{OR}_3)^2}$. $\square$

Claim 4.9 $\mathrm{wsize}(P_{\mathrm{MAJ}_3}) = 2 = \mathrm{Adv}(\mathrm{MAJ}_3)$, *where P_{MAJ_3} is the span program from Example 2.2.*

Proof. Substitute P_{MAJ_3} into Definition 3.5. Some of the witness vectors are $|w'_{000}\rangle = (1, 0)$, $|w'_{100}\rangle = (1, -1/\sqrt{3})$, and $|w_{110}\rangle = (e^{-i\pi/6}, e^{i\pi/6}, 0)$, $|w_{111}\rangle = (1, 1, 1)/\sqrt{3}$. $\square$

Claim 4.10 *Letting $\alpha = \sqrt[4]{k-1}$, the span program*

$$\begin{array}{rl} & X_J = (\{x_1, x_2, \ldots, x_k\} \ \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k}\}) \\ t = (1)\,, & v_J = (\qquad \alpha \qquad\qquad\qquad \alpha \qquad) \end{array}$$

computes EQUAL_k with witness size $\frac{k}{\sqrt{k-1}} = \mathrm{Adv}(\mathrm{EQUAL}_k)$.

Proof. Substitute into Definition 3.5. The witnesses are $|w'_{\mathrm{unequal}}\rangle = (1)$, $|w_{0^k}\rangle = \big(0, \frac{\sqrt{k}}{\alpha}\big)$ and $|w_{1^k}\rangle = \big(\frac{\sqrt{k}}{\alpha}, 0\big)$. $\square$

Claim 4.11 *Let $g(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$. Letting $\alpha_1 = \sqrt[4]{1 + \frac{1}{\sqrt{3}}}$ and $\alpha_2 = \sqrt{\sqrt{3} - 1}$, the span program*

$$\begin{array}{rl} & X_J = (\{x_1, x_2\} \ \{\overline{x_1}, \overline{x_2}\} \ \{x_3\}) \\ t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & v_J = \begin{pmatrix} \alpha_1 & \alpha_1\alpha_2 & 0 \\ \alpha_2 & 0 & 1 \end{pmatrix} \end{array}$$

computes g with witness size $\sqrt{3 + \sqrt{3}} = \mathrm{Adv}(g)$.

Proof. By substitution into Definition 3.5. $\square$

Lemma 4.12 *Consider $f(x, x') = f'(b(x), b'(x'))$, with $f' \in \{\mathrm{PARITY}, \mathrm{OR}\}$, and b and b' functions on $O(1)$ bits. Assume that there exist span programs P_b and $P_{b'}$ for b and b' with respective witness sizes $B = \mathrm{wsize}(P_b)$ and $B' = \mathrm{wsize}(P_{b'})$. Then there exists a span program P for f with witness size $\mathrm{wsize}(P) = B + B'$ if $f' = \mathrm{PARITY}$, or $\sqrt{B^2 + B'^2}$ if $f' = \mathrm{OR}$.*

Proof. Substitute the following span programs with zero constraints into Definition 3.5:

$$\begin{array}{rl} & X_J = (\ \{x_1, \overline{x_2}\} \quad \{\overline{x_1}, x_2\}\ ) \\ P_{\mathrm{PARITY}} : t = (1)\,, & v_J = (\qquad 1 \qquad\qquad 1 \qquad)\ , \end{array}$$

$$\begin{array}{rl} & X_J = (\quad \{x_1\} \qquad \{x_2\} \quad) \\ P_{\mathrm{OR}} : t = (1)\,, & v_J = \left(\frac{\sqrt{B}}{\sqrt[4]{B^2 + B'^2}} \ \ \frac{\sqrt{B'}}{\sqrt[4]{B^2 + B'^2}} \right)\ . \end{array}$$

The witness vectors for PARITY are $|w'_{00}\rangle = (1)$ and $|w_{10}\rangle = (\sqrt{B^2 + B'^2}, 0)$, and the witness vectors for OR are $|w'_{00}\rangle = (1)$, $|w_{10}\rangle = (\sqrt[4]{B^2 + B'^2}, 0)$, and $|w_{11}\rangle = (1, 1) \cdot \frac{1}{2}\sqrt[4]{B^2 + B'^2}$. $\square$

Gate f	$\mathrm{Adv}(f)$	Gate f	$\mathrm{Adv}(f)$
0	0	$\mathrm{MAJ}_3(x_1,x_2,x_3) = (x_1 \wedge x_2) \vee ((x_1 \vee x_2) \wedge x_3)$	2
x_1	1	$\mathrm{EQUAL}_3(x_1,x_2,x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3})$	$3/\sqrt{2}$
$x_1 \wedge x_2$	$\sqrt{2}$	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$	$\sqrt{3+\sqrt{3}}$
$x_1 \oplus x_2$	2	$x_1 \vee (x_2 \oplus x_3)$	$\sqrt{5}$
$x_1 \wedge x_2 \wedge x_3$	$\sqrt{3}$	$x_1 \oplus (x_2 \wedge x_3)$	$1+\sqrt{2}$
$x_1 \vee (x_2 \wedge x_3)$	$\sqrt{3}$	$\mathrm{EXACT}_{2 \text{ of } 3}(x_1,x_2,x_3) = \mathrm{MAJ}_3(x_1,x_2,x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$	$\sqrt{7}$
$(x_1,x_2)_{1+x_3} = (x_3 \wedge x_2) \vee (\overline{x_3} \wedge x_1)$	2	$x_1 \oplus x_2 \oplus x_3$	3

Figure 5: Binary gates on up to three bits. Up to equivalences—permutation of inputs, complementation of some or all inputs or output—there are fourteen binary gates on three inputs x_1, x_2, x_3. Adversary bounds $\mathrm{Adv}(f)$ for all functions f on up to four bits have been computed by [16], and see [22].

Remark 4.13 *Our procedure for analyzing a function f has been as follows:*

1. *First determine a span program P computing $f_P = f$. The simplest span program is derived from the minimum-size {AND, OR, NOT} formula for f.*
2. *Next, compute* $\mathrm{wsize}(P,x)$ *for each input x, as a function of the variable weights of P.*
3. *Finally, optimize the free weights of P to minimize* $\mathrm{wsize}(P) = \max_x \mathrm{wsize}(P,x)$*. For example, note that scaling A_{OJ} up helps the true cases in Definition 3.5, and hurts the false cases; therefore choose a scale to balance the worst true case against the worst false case.*

   We respect the symmetries of f during optimization. On the other hand, if two literals are not treated symmetrically by f, then we do not group them together in any grouped input X_j. For example, in Claim 4.11 we do not group x_3 together with x_1 and x_2 in X_1.

4.3 Span program spectral analysis of φ

Theorem 4.14 *Consider an adversary-balanced formula φ on the gate set $\mathcal{S}$, with adversary bound* $\mathrm{Adv}(\varphi)$*. Let P be the composed span program computing $f_P = \varphi$. For an input $x \in \{0,1\}^N$, recall the definition of the weighted graph $G_P(x)$ from Theorem 2.5; if the literal on an input edge evaluates to true, then delete that edge from G_P. Let $\tilde{G}_P(x)$ be the same as $G_P(x)$ except with the weight on the output edge (a_O, b_O) set to $w = \epsilon_w/\sqrt{\mathrm{Adv}(\varphi)}$ (instead of weight one), where $\epsilon_w > 0$ is a sufficiently small constant. Then,*

- *If $\varphi(x) = 1$, there exists a normalized eigenvalue-zero eigenvector of the adjacency matrix $A_{\tilde{G}_P(x)}$ with $\Omega(1)$ support on the output vertex a_O.*
- *If $\varphi(x) = 0$, then for some small enough constant $\epsilon > 0$, $A_{\tilde{G}_P(x)}$ does not have any eigenvalue-λ eigenvectors supported on a_O or b_O for $|\lambda| \leq \epsilon/\mathrm{Adv}(\varphi)$.*

Proof. The proof of Theorem 4.14 has two parts. First, we will prove by induction that $\sigma_g = O(\mathrm{Adv}(\varphi_g))$. Then, by considering the last eigenvector constraint, $\lambda a_O = w b_O$, we either construct the desired eigenvector or derive a contradiction, depending on whether $\varphi(x)$ is true or false.

Base case. Consider an input x_i to the formula φ. If $x_i = 1$, then the corresponding input edge (a_j, b_i) is not in $G_P(x)$. In particular, the input i does not contribute to the expression for $\tilde{\sigma}_j(x)$ in Eq. (3.3), so S_i and γ_i may be left undefined. If $x_i = 0$, then the input edge (a_j, b_i) is in $G_P(x)$. The eigenvalue-λ equation at b_i is $\lambda b_i = a_j$. For $\lambda = 0$, this is just $a_j = 0$, so let $\gamma_i = 1$. For $\lambda > 0$, this is $r_i = \lambda s_i = a_j/b_i = \lambda$, so $s_i = S_i = 1$.

Induction. Assume that $|\lambda| \leq \epsilon/\mathrm{Adv}(\varphi)$, for some small enough constant $\epsilon > 0$.

Consider a gate g. Let $h_1, \ldots, h_k$ be the inputs to g. Let φ_g denote the subformula of φ based at g. By Theorem 3.6 and Theorem 4.8, the output bound σ_g satisfies

$$\sigma_g \lesssim \mathrm{Adv}(g) \max_i \sigma_{h_i} \ , \tag{4.4}$$

or equivalently

$$\sigma_g \leq c_1 + \mathrm{Adv}(g)(\max_i \sigma_{h_i})(1 + c_2 \cdot |\lambda| \mathrm{Adv}(\varphi_g)) \tag{4.5}$$

for certain constants c_1, c_2. Different kinds of gates give different constants in Eq. (4.5), but since the gate set is finite, all constants are uniformly $O(1)$.

Since $|\lambda| \leq \epsilon/\mathrm{Adv}(\varphi)$, the recurrence Eq. (4.5) has solution

$$\sigma_g \leq O\left(\max_{\chi} \prod_{h \in \chi} \mathrm{Adv}(h) \Big(1 + \epsilon' \frac{\mathrm{Adv}(\varphi_h)}{\mathrm{Adv}(\varphi)} \Big) \right) \ ,$$

where the maximum is taken over the choice of χ a non-self-intersecting path from g up to an input. Because φ is by assumption adversary balanced (Definition 4.5), $\prod_{h \in \chi} \mathrm{Adv}(h) = \mathrm{Adv}(\varphi_g)$ (Theorem 4.6). Also, $\prod_{h \in \chi}(1 + \epsilon' \frac{\mathrm{Adv}(\varphi_h)}{\mathrm{Adv}(\varphi)}) = O(1)$. Therefore, the solution satisfies

$$\sigma_g = O(\mathrm{Adv}(\varphi_g)) \ . \tag{4.6}$$

Final amplification step. Assume $\varphi(x) = 1$. Then by Eq. (4.6), there exists a normalized eigenvalue-zero eigenvector of the graph $G_P(x)$ with squared amplitude $|a_O|^2 \geq \gamma_O = 1/O(\mathrm{Adv}(\varphi))$. Recall that $w = \epsilon_w/\sqrt{\mathrm{Adv}(\varphi)}$ is the weight of the output edge (a_O, b_O) of P in $\tilde{G}_P(x)$, and let $\hat{a}_O = w a_O$. The $\lambda = 0$ eigenvector equations for $\tilde{G}_P(x)$ are the same as those for $G_P(x)$, except with $\hat{a}_O$ in place of a_O. Therefore, we may take $|\hat{a}_O|^2 = 1/O(\mathrm{Adv}(\varphi))$, so for a normalized eigenvalue-zero eigenvector of $\tilde{G}_P(x)$, $|a_O|^2 = \Omega(1)$. By reducing the weight of the output edge from 1 to w, we have amplified the support on a_O up to a constant.

Now assume that $\varphi(x) = 0$. By Theorem 2.5, there does not exist any eigenvalue-zero eigenvector supported on a_O. Also $b_O = 0$ at $\lambda = 0$ by the constraint $\lambda a_O = w b_O$. For $\lambda \neq$

0, $|\lambda| \leq \epsilon/\mathrm{Adv}(\varphi)$, Eq. (4.6) implies that in any eigenvalue-λ eigenvector for $\tilde{G}_P(x)$, either $\hat{a}_O = b_O = 0$ or the ratio $|\hat{a}_O/b_O| \leq |\lambda| \cdot O(\mathrm{Adv}(\varphi))$, so

$$|a_O/b_O| \leq \text{constant} \cdot \frac{|\lambda|}{w} \mathrm{Adv}(\varphi) \qquad (4.7)$$

for some constant that does not depend on w. We have not yet used the eigenvector equation at a_O, $\lambda a_O = w b_O$. Combining this equation with Eq. (4.7), we get $w^2 \leq \text{constant} \cdot \lambda^2 \mathrm{Adv}(\varphi) \leq \text{constant} \cdot \epsilon^2/\mathrm{Adv}(\varphi)$. Substituting $w = \epsilon_w/\sqrt{\mathrm{Adv}(\varphi)}$, this is a contradiction provided we set ϵ_w so $\epsilon_w^2 > \text{constant} \cdot \epsilon^2$. Therefore, the adjacency matrix of $\tilde{G}_P(x)$ cannot have an eigenvalue-λ eigenvector supported on a_O or b_O. $\square$

4.4 Quantum algorithm

We apply Theorem 4.14 and the Szegedy correspondence between discrete- and continuous-time quantum walks [28] to design the optimal quantum algorithm needed to prove Theorem 4.7. The approach is similar to that used for the NAND formula evaluation algorithm of [9], with only technical differences. For details, see Ref. [23].

The main idea is to construct a discrete-time quantum walk $U_x = O_x U_{0^N}$ on the directed edges of G_P whose spectrum and eigenvectors correspond exactly to those of $A_{\tilde{G}_P(x)}$. Here U_{0^N} is a fixed unitary operator only depending on the formula graph $A_{\tilde{G}_P(0^N)}$, which can be implemented efficiently without access to the input x, and O_x is the input oracle mapping

$$O_x|v,w\rangle = \begin{cases} (-1)^{x_{i(v)}}|v,w\rangle & \text{if } v \text{ is a leaf} \\ |v,w\rangle & \text{otherwise} \end{cases}$$

where $i(v)$ is the index of the input variable corresponding to the leaf v.

Now starting at the output edge $|a_O, b_O\rangle$, run phase estimation [10] on U_x with precision $\delta_p = 1/O(\mathrm{Adv}(\varphi))$ and error δ_e a small enough constant. Output "$\varphi(x) = 1$" iff the output phase is zero. The query complexity of this algorithm is $O(1/\delta_p) = O(\mathrm{Adv}(\varphi))$. The first part of Theorem 4.14 implies completeness, because the initial state has constant overlap with an eigenstate of U_x with phase zero. The second part of Theorem 4.14 implies soundness, because the spectral gap away from zero is greater than the precision δ_p.

5. EXTENSIONS

Theorem 4.7 can be extended in several directions.

5.1 Unbalanced formulas

Can the restriction that the gates have adversary-balanced inputs be significantly weakened? So far, we have only analyzed the PARITY and OR gates for unbalanced inputs, in Lemma 4.12. For the MAJ_3 gate, we have found an optimal span program for the case in which only two of the inputs are balanced:

Lemma 5.1 *Let* $f(x,x',x'') = \mathrm{MAJ}_3(b(x), b'(x'), b''(x''))$ *with* b, b', b'' *functions on* $O(1)$ *bits computed by span programs* $P_b, P_{b'}, P_{b''}$ *with witness sizes* $B = \mathrm{wsize}(P_b) = \mathrm{Adv}(b) = \mathrm{wsize}(P_{b'}) = \mathrm{Adv}(b')$ *and* $B'' = \mathrm{wsize}(P_{b''}) = \mathrm{Adv}(b'')$. *Let* $\beta = B''/B$. *Then there exists a span program* P *for* f *with* $\mathrm{wsize}(P) = \frac{1}{2}\big(\sqrt{8+\beta^2}+\beta\big)B = \mathrm{Adv}(f)$*:*

$$X_J = (\{x_1\} \ \{x_2\} \qquad \{x_3\} \quad)$$
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} \alpha & \alpha & \sqrt{\frac{1}{2}+\beta\alpha^2} \\ i & -i & 2\alpha \end{pmatrix} \ ,$$

where $\alpha = \frac{1}{2\sqrt{2}}(\sqrt{8+\beta^2}-\beta)^{1/2}$.

Therefore, for example, the gates $\mathrm{MAJ}_3(x_1, x_2, x_3 \wedge x_4)$ and $\mathrm{MAJ}_3(x_1, x_2, x_3 \oplus x_4)$ can be added into $\mathcal{S}$ without affecting the correctness of Theorem 4.7. However, we do not have an understanding of MAJ_3 when all three input complexities differ, and for most other gates we know similarly little.

5.2 Four-bit gates

The gate set $\mathcal{S}$ includes all three-bit binary gates. What about four-bit gates? Up to symmetries, there are 208 inequivalent binary functions that depend on exactly four input bits $x_1, \ldots, x_4$. The functions we have considered so far are listed at the webpage [22]. To summarize,

- Thirty functions can be written as a PARITY or OR of two subformulas on disjoint inputs. These functions are already included in $\mathcal{S}$ (Definition 4.1).
- For 24 additional functions, we have found a span program with witness size matching the adversary lower bound. These functions can be added to $\mathcal{S}$.

Example 5.2 (Threshold 2 of 4) *In analogy to Example 2.2, one might consider the span program*

$$X_J = (\{x_1\} \ \{x_2\} \ \{x_3\} \ \{x_4\})$$
$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \end{pmatrix} \ .$$

This span program computes Threshold$_{2 \textit{ of } 4}(x_{[4]})$*—*$\mathrm{MAJ}_3$ *is Threshold*$_{2 \textit{ of } 3}$*—but it is not optimal. Intuitively, the problem is that the different pairs of inputs are not symmetrical. An optimal span program, with witness size* $\sqrt{6}$*, is*

$$X_J = (\{x_1\}\{x_1\}\{x_2\}\{x_2\}\{x_3\}\{x_3\}\{x_4\}\{x_4\})$$
$$t = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad v_J = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & i & -i & i & i \\ i & -i & i & i & 1 & 1 & 1 & -1 \end{pmatrix}$$

It was derived by embedding a four-simplex symmetrically in the 2×2 *unitaries, in correct analogy to Example 2.2.*

It seems that inevitably k-bit gates are going to require more involved techniques to evaluate optimally, for k large enough. It may well be that four-bit gates are already interesting in this sense.

5.3 Witness vectors and the adversary bound

The witnesses in Definition 3.5 have an interesting property related to a dual version of the adversary bound [20, 27]: Assume that all $|X_j| = 1$ and $S = 1$. For $x, y \in \{0,1\}^n$ with $f_P(x) = 1$, $f_P(y) = 0$, consider the witnesses $|w_x\rangle$, $|w'_y\rangle$ achieving the minima in Eqs. (3.4), (3.5), and let $|w_y\rangle = A^\dagger|w'_y\rangle$. Then $|w_x\rangle = \Pi(x)|w_x\rangle$ and $\overline{\Pi}(y)|w_y\rangle = |w_y\rangle$, so

$$\langle w_x|\Pi(x)\overline{\Pi}(y)|w_y\rangle = \langle w_x|A^\dagger|w'_y\rangle = \langle t|w'_y\rangle = 1 \ .$$

Therefore, if we define $p_x(i) = \frac{1}{\||w_x\rangle\|^2} \sum_{\substack{j:\ X_j=\{x_i\} \\ \vee X_j=\{\overline{x_i}\}}} |\langle j|w_x\rangle|^2$ for each x (for both true and false $f_P(x)$) and for $i \in [n]$, then we get a feasible set of probability distributions for the minimax formulation of the adversary bound [27]. If $\mathrm{wsize}(P) = \mathrm{Adv}(f_P)$, then this set of probability distributions is optimal.

In this paper, we only use the nonnegative version of the adversary bound. Høyer, Lee and Špalek introduced a generalized adversary bound $\mathrm{Adv}^{\pm}$, with negative entries allowed in the adversary matrix, and showed that it is also a lower bound on the quantum query complexity [17]. For most functions f on 4 bits, $\mathrm{Adv}^{\pm}(f) > \mathrm{Adv}(f)$ [16]. Since $\mathrm{Adv}^{\pm}$ composes similarly to Theorem 4.6, one gets an asymptotically higher lower bound for formulas with such functions as gates than using Adv. We have not been able to find a matching span program for any such function. The dual formulation of $\mathrm{Adv}^{\pm}$ cannot be expressed using probability distributions and one therefore cannot hope for a simple correspondence with the witnesses like described above.

Both variants of the adversary bound, Adv and $\mathrm{Adv}^{\pm}$, can be expressed as optimal solutions of certain semidefinite programs. Can one find a semidefinite formulation of span program witness size?

6. ACKNOWLEDGEMENTS

We thank Troy Lee for pointing out span programs to us. B.R. would like to thank Sean Hallgren and Shengyu Zhang for helpful conversations, and Cris Moore for discussions on the classical iterated MAJ_3 evaluation algorithm.

7. REFERENCES

[1] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8:99–126, 1999. Preliminary version in *Proc. of 28th ACM STOC*, 1996.

[2] A. Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Syst. Sci.*, 72(2):220–238, 2006. Preliminary version in *Proc. of 44th IEEE FOCS*, 2003.

[3] A. Ambainis. A nearly optimal discrete query quantum algorithm for evaluating NAND formulas. arXiv:0704.3628 [quant-ph], 2007.

[4] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In *Proc. of 48th IEEE FOCS*, pages 363–372, 2007.

[5] L. Babai, A. Gál, and A. Wigderson. Superpolynomial lower bounds for monotone span programs. *Combinatorica*, 19(3):301–319, 1999. Preliminary version in *Proc. of 28th ACM STOC*, 1996.

[6] H. Barnum and M. Saks. A lower bound on the quantum query complexity of read-once functions. *J. Comput. Syst. Sci.*, 69(2):244–258, 2004.

[7] H. Barnum, M. Saks, and M. Szegedy. Quantum decision trees and semidefinite programming. In *Proc. of 18th IEEE Complexity*, pages 179–193, 2003.

[8] A. Beimel, A. Gál, and M. Paterson. Lower bounds for monotone span programs. *Computational Complexity*, 6:29–45, 1996. Preliminary version in *Proc. of 36th IEEE FOCS*, 1995.

[9] A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Every NAND formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. quant-ph/0703015, 2007.

[10] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proc. R. Soc. London A*, 454(1969):339–354, 1998.

[11] R. Cramer and S. Fehr. Optimal black-box secret sharing over arbitrary Abelian groups. In *Proc. CRYPTO 2002*, LNCS vol. 2442, pages 272–287. Springer-Verlag, 2002.

[12] E. Farhi, J. Goldstone, and S. Gutmann. A quantum algorithm for the Hamiltonian NAND tree. quant-ph/0702144, 2007.

[13] A. Gál and P. Pudlák. A note on monotone complexity and the rank of matrices. *Information Processing Letters*, 87(6):321–326, 2003.

[14] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. of 28th ACM STOC*, pages 212–219, 1996.

[15] L. K. Grover. Tradeoffs in the quantum search algorithm. quant-ph/0201152, 2002.

[16] P. Høyer, T. Lee, and R. Špalek. Source codes of semidefinite programs for $\mathrm{ADV}^{\pm}$. `http://www.ucw.cz/~robert/papers/adv/`, 2006.

[17] P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proc. of 39th ACM STOC*, pages 526–535, 2007.

[18] T. S. Jayram, R. Kumar, and D. Sivakumar. Two applications of information complexity. In *Proc. of 35th ACM STOC*, pages 673–682, 2003.

[19] M. Karchmer and A. Wigderson. On span programs. In *Proc. of 8th IEEE Symp. Structure in Complexity Theory*, pages 102–111, 1993.

[20] S. Laplante and F. Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In *Proc. of 19th IEEE Complexity*, pages 294–304, 2004.

[21] V. Nikov, S. Nikova, and B. Preneel. On the size of monotone span programs. In *Proc. SCN 2004*, LNCS vol. 3352, pages 249–262, 2005.

[22] B. W. Reichardt and R. Špalek. Quantum query complexity of up to 4-bit functions. `http://www.ucw.cz/~robert/papers/gadgets/`, 2007.

[23] B. W. Reichardt and R. Špalek. Span-program-based quantum algorithm for evaluating formulas. arXiv:0710.2630 [quant-ph], 2007.

[24] M. Saks and A. Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. of 27th IEEE FOCS*, pages 29–38, 1986.

[25] M. Santha. On the Monte Carlo decision tree complexity of read-once formulae. *Random Structures and Algorithms*, 6(1):75–87, 1995. Preliminary version in *Proc. of 6th IEEE Structure in Complexity Theory*, 1991.

[26] M. Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[27] R. Špalek and M. Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2(1):1–18, 2006. Earlier version in ICALP'05.

[28] M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proc. of 45th IEEE FOCS*, pages 32–41, 2004.