Span-program-based quantum algorithm for evaluating formulas

Ben W. Reichardt[*] breic@caltech.edu

Robert Špalek[†] spalek@google.com

Abstract

We present a time-efficient and query-optimal quantum algorithm for evaluating adversary-bound-balanced formulas on an extended gate set. The allowed gates include arbitrary two- and three-bit gates, as well as bounded fan-in AND, OR, PARITY and EQUAL gates. The technique behind the formula evaluation algorithm is a new framework for quantum algorithms based on span programs.

For example, the classical complexity of evaluating the balanced ternary majority formula is unknown, and the natural generalization of the standard balanced AND-OR formula evaluation algorithm is known to be suboptimal. In contrast, a generalization of the optimal quantum {AND, OR, NOT} formula evaluation algorithm is optimal for evaluating the balanced ternary majority formula.

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 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 gates $\mathcal{S} = \{\text{AND, OR, NOT}\}$ [ACR+07]. We extend the gate set $\mathcal{S}$. We develop an optimal quantum algorithm for evaluating a certain broad definition of "adversary-balanced" formulas, over a gate set that includes arbitrary three-bit gates, as well as bounded fan-in EQUAL gates and bounded-size {AND, OR, NOT, PARITY} formulas considered as single gates. "Adversary-balanced" means that the inputs to a gate must have exactly equal adversary lower bounds.

The main idea of our algorithm is to consider a weighted graph $G(\varphi)$ obtained by replacing each gate of φ with a small gadget graph, and possibly also duplicating subformulas. Figure 1 gives several examples of these gadgets, known classically as "span programs." We relate the evaluation of the formula to the presence or absence of small-eigenvalue eigenstates of the weighted adjacency matrix $A_{G(\varphi)}$ that are supported on the root vertex of $G(\varphi)$.

As a special case, for example, our algorithm evaluates the depth-d balanced ternary majority formula using $O(2^d = N^{\log_3 2})$ 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 [CRŠZ07], used $O(\sqrt{5}^d)$ queries. 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.

Classical and quantum background

The formula evaluation problem has been well-studied in the classical computer model. Classically, the case $\mathcal{S} = \{\text{NAND}\}$ (or $\mathcal{S} = \{\text{NOR}\}$ equivalently) is best understood. A formula with only NAND, or

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

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

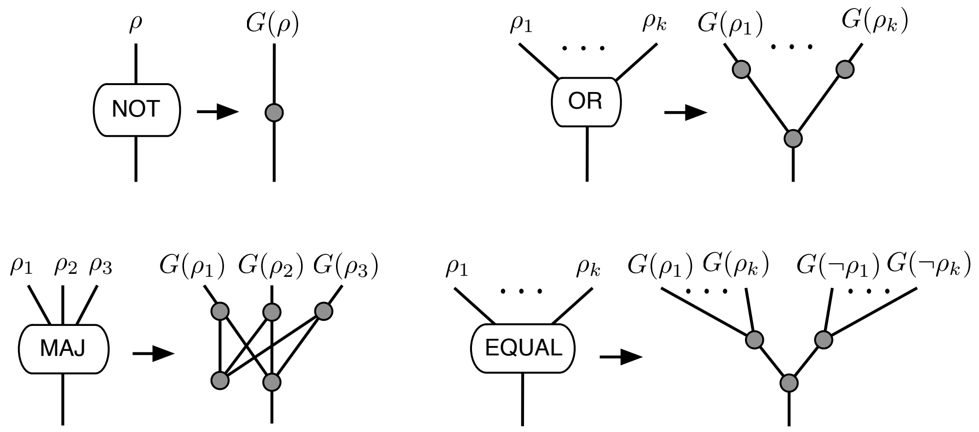

Figure 1: To convert a formula φ to the corresponding graph $G(\varphi)$, we recursively apply substitution rules starting at the root to convert each gate into a gadget subgraph. Some of the rules are shown here, except with the edge weights not indicated. The dangling edges at the top and bottom of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

only NOR, gates is equivalent to one with alternating levels of AND and OR gates, a so-called "AND-OR formula." One can compute the value of a balanced binary AND-OR formula with zero error in expected time $O(N^{\log_2[(1+\sqrt{33})/4]}) = O(N^{0.754})$ [Sni85, SW86], and this is optimal even for bounded-error algorithms [San95]. 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 $\Theta(N)$.

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 [Gro96, Gro02] 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 balanced binary AND-OR formulas in time and queries $N^{\frac{1}{2}+o(1)}$ [FGG07]. Ambainis [Amb07] gave an $O(\sqrt{N})$-query algorithm for evaluating balanced binary AND-OR trees. Childs, Reichardt, Špalek and Zhang [CRŠZ07] improved this to $O(\sqrt{N})$ queries for balanced or "approximately balanced" formulas, and extended the algorithm to evaluate 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.

We will see in this paper that the formula evaluation problem has other nice features in the quantum computing 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 ternary majority (MAJ_3) function [SW86], and the answer is still unknown. The complexity is only known to lie between $\Omega((7/3)^d)$ and $O((2.655\ldots)^d)$ for a depth-d formula [JKS03].[1] In particular, the naïve algorithm of recursively evaluating 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 the above short-circuit-style algorithm is known to be optimal.

In contrast, we show that the optimal quantum algorithm of [CRŠZ07] does extend to give an optimal $O(2^d)$-query algorithm for evaluating the balanced ternary majority formula. In fact, the algorithm also

[1]A slightly better analysis of the algorithm in [JKS03] gives $O((2.6537\ldots)^d)$, and the algorithm can likely be improved.

generalizes to a significantly larger gate set $\mathcal{S}$, as already mentioned.

Even though the quantum speedups we find are only polynomial—at best we obtain a quadratic improvement from N to $\sqrt{N}$ queries—the fact that studying formula evaluation in a quantum computer model leads to new connections between problems is, we feel, already sufficient motivation for the results. Additionally, though, we feel that there are likely more connections and more algorithmic applications to be developed or discovered. For example, our technique will make use of gadgets that are known in classical complexity theory as "span programs" [KW93]. Span programs have been used to prove lower bounds on formula size [KW93, BGW99] and monotone span programs are related to linear secret-sharing schemes [BGP96]. (Most, though not all [ABO99], applications are over finite fields, whereas we use the definition over **C**.) We will only use combinations of constant-size span programs, but it is interesting to speculate that larger span programs could directly give interesting new quantum algorithms.

Organization

After defining terms in Section 2 we formally state our result in Theorem 3.1, an optimal algorithm for evaluating adversary-balanced formulas (Definition 2.8) over the gate set $\mathcal{S}$ of Definition 2.1.

In Section 4, we introduce span programs. We define span program P computing a function f_P, its dual span program $P^\dagger$ computing $\neg f_P$. By considering parts of P as a weighted adjacency matrix, we define the corresponding weighted graph G_P. We prove Lemma 4.6, which states that eigenvalue-zero eigenstates of the adjacency matrix of G_P compute the function f_P in a certain dual-rail encoding.

Lemma 4.6 provides intuition for the more quantitative analysis we develop in Sections 5 and 6. In Section 5, we analyze small-eigenvalue eigenstates (Theorem 5.6). Essentially, we rigorously upper-bound higher-order terms in the eigenvalue λ to leave just the first-order term. In Section 6, we develop a quantitative version of Lemma 4.6, at eigenvalue $\lambda = 0$.

The results for small-eigenvalue eigenstates, and for eigenvalue-zero eigenstates are closely related. In Theorem 7, we restate Theorem 5.6 and Lemma 4.6 with a unified notation for small $\lambda \neq 0$ and for $\lambda = 0$. In Section 9, we consider the different gates in $\mathcal{S}$ one at a time, writing down a span program P for each and then immediately substituting it into Theorem 7. There are five cases.

In Section 11, we prove the main result, Theorem 3.1, by exhibiting the formula evaluation algorithm based on span program spectral analysis (Theorem 10.1). The graph $G(\varphi)$ of Figure 1 is G_P for a certain span program P. The algorithm is a straightforward application of phase estimation on a quantum walk on $G(\varphi)$.

Finally, we describe some extensions to the work in Section 12, and state open problems in Section 13.

2 Definitions

In this section, we define the gate set $\mathcal{S}$ and the type of "adversary-balanced" formulas on which our algorithm will run optimally.

Definition 2.1 (Extended gate set $\mathcal{S}$). *Let*

$$\begin{aligned} \mathcal{S}' &= \{\textit{arbitrary two- or three- bit gates}, O(1)\textit{-fan-in EQUAL gates}\} \\ \mathcal{S} &= \mathcal{S}' \cup \left\{ \begin{array}{c} O(1)\textit{-size } \{\textit{AND, OR, NOT, PARITY}\} \textit{ formulas} \\ \textit{on inputs that are themselves possibly elements of } \mathcal{S}' \end{array} \right\} . \end{aligned} \tag{2.1}$$

Example 2.2. Note that $\mathcal{S}$ 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 $\mathrm{MAJ}_3(x_1, x_2, x_3 \wedge x_4) \notin \mathcal{S}$.

In order to define "adversary-balanced" formulas, we need to define the quantum adversary bound (with nonnegative weights).

Table 1: 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 for all functions on up to four bits have been computed by [HLŠ06], and see [RŠ07].

Gate	Adversary lower bound
0	0
x_1	1
$x_1 \wedge x_2$	$\sqrt{2}$
$x_1 \oplus x_2$	2
$x_1 \wedge x_2 \wedge x_3$	$\sqrt{3}$
$x_1 \oplus x_2 \oplus x_3$	3
$x_1 \oplus (x_2 \wedge x_3)$	$1+\sqrt{2}$
$x_1 \vee (x_2 \wedge x_3)$	$\sqrt{3}$
$(x_1 \wedge x_2) \vee (\overline{x_1} \wedge x_3)$	2
$x_1 \vee (x_2 \wedge x_3) \vee (\overline{x_2} \wedge \overline{x_3})$	$\sqrt{5}$
$\mathrm{MAJ}_3(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee ((x_1 \vee x_2) \wedge x_3)$	2
$\mathrm{MAJ}_3(x_1, x_2, x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3})$	$\sqrt{7}$
$\mathrm{EQUAL}(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3})$	$3/\sqrt{2}$
$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$	$\sqrt{3+\sqrt{3}}$

Definition 2.3 (Nonnegative-weight adversary bound $A(\mathrm{f})$)**.** *Let $f : \{0,1\}^k \to \{0,1\}$. Define*

$$A(f) = \max_{\substack{\Gamma \geq 0 \\ \Gamma \neq 0}} \frac{\|\Gamma\|}{\max_i \|\Gamma \circ D_i\|} \ , \tag{2.2}$$

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 $A(f)$ gives a lower bound on the number of queries to the phase-flip input oracle O_x required to evaluate f on input x, i.e., a lower bound on the quantum query complexity:

Definition 2.4 (Phase-flip input oracle)**.** *The phase-flip input oracle for an input string x is a unitary operator*

$$O_x : |c\rangle \otimes |i\rangle \mapsto \begin{cases} (-1)^{x_i}|c\rangle \otimes |i\rangle & \text{if } c = 1 \\ |c\rangle \otimes |i\rangle & \text{if } c = 0 \end{cases}$$

Theorem 2.5 ([Amb06a, BSS03])**.** *The two-sided ϵ-bounded error quantum query complexity of function f, $Q_\epsilon(f)$, is at least $\frac{1-2\sqrt{\epsilon(1-\epsilon)}}{2}A(f)$.*

Therefore, to match the lower bound of Theorem 2.5, our goal will be to use $O(A(\varphi))$ queries to evaluate φ.

Remark 2.6. *We have used the nonnegative weight adversary bound A instead of the more general adversary bound $\mathrm{ADV}^{\pm}$ of [HLŠ07] because for any $g \in \mathcal{S}$, $A(g) = \mathrm{ADV}^{\pm}(g)$. See the open problems in Section 13.*

Example 2.7. The adversary bounds for all binary functions on up to three bits are given in Table 1. The adversary bounds for all functions on up to four bits are listed at the webpage [RŠ07], taken from [HLŠ06]; the list also includes $\mathrm{ADV}^{\pm}(f)$ and the minimum-size {AND, OR, NOT} formula for each f.

Definition 2.8 (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 $A(\varphi_{h_1}) = \cdots = A(\varphi_{h_k})$.*

To motivate Definition 2.8, we need a weak version of an adversary composition result from [HLŠ05, Amb06a]:

Theorem 2.9 ([HLŠ05, Cor. 4])**.** *Let $f = g \circ (h_1, \ldots, h_k)$, where $A(h_1) = \cdots = A(h_k)$ and $\circ$ denotes function composition. Then $A(f) = A(g)A(h_1)$.*

If φ is adversary-balanced, then by Theorem 2.9 $A(\varphi_g)$ is the product of the gate adversary bounds along any non-self-intersecting path χ from g up to an input, $A(\varphi_g) = \prod_{h\in\chi} A(h)$. Note that $A(\neg f) = A(f)$, so NOT gates can be inserted anywhere in an adversary-balanced formula.

Example 2.10. A special case of an adversary-balanced formula is a layered formula, in which all inputs are at the same depth and all gates at the same depth are of the same type, with the same fan-in.

3 Result

Theorem 3.1. *There exists a quantum algorithm that evaluates an adversary-balanced formula $\varphi(x)$ over $\mathcal{S}$ using $O(A(\varphi))$ 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 $A(\varphi)(\log N)^{O(1)}$.*

Remark 3.2. *By Theorem 2.5, the algorithm in Theorem 3.1 is query-optimal. Note though that $A(\varphi)$ is computed assuming that all N inputs to the formula φ are independent variables $x_1 x_2 \ldots x_N$. If we are promised that some of the inputs are the same, e.g., that $x_N = x_1$, then the algorithm still runs with the same query efficiency. However it may no longer be optimal on the restricted input set.*

A very special case of Theorem 3.1 solves in the quantum model the question posed by Boppana [SW86]:

Corollary 3.3. *A balanced ternary majority formula of depth d (on $N = 3^d$ inputs) can be evaluated by a quantum algorithm using $O(2^d) = O(N^{\log_3 2})$ input oracle queries and $O(N^{\log_3 2}(\log N)^{O(1)})$ time.*

From Table 1, the adversary bound $A(\mathrm{MAJ}_3) = 2$. By Theorem 2.9 the adversary bound for the balanced MAJ_3 formula of depth d is 2^d, so $O(2^d)$ queries is indeed optimal.

4 Span programs and eigenvalue-zero graph eigenstates

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 [KW93] as well as, for example, the more recent [GP03]. 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 a zero-eigenvalue eigenvector of $A_{G_P}(x)$ supported on a distinguished output node (Lemma 4.6). In turn, this will imply that giving a span program for a function immediately gives a quantum algorithm for evaluating that function, or for evaluating formulas including that the function as a gate. The quantum algorithm will work by spectral estimation on $A_{G_P}(x)$. The algorithm's running time depends on detailed properties of the span program.

4.1 Span program definition

Definition 4.1 (Span program)**.** *A span program P consists of a nonzero "target" vector t, together with "grouped input" vectors $\{v_j : j \in J\}$. Each v_j is labeled with a subset X_j of the literals $X = \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$. The* size *of P is defined as $\sum_j |X_j|$. 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 associated in X_j is zero (i.e., false).*

Example 4.2. For example, the span program

$$t = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 1 & \omega & \omega^2 \end{pmatrix} \quad ,$$

where $\omega = e^{2\pi i/3}$ and with v_i labeled by x_i, computes the MAJ_3 function. Indeed, at least two of the v_i must have nonzero coefficient in any linear combination equaling the target t. This span program corresponds to the MAJ_3 gate gadget of Section 9. Of course, the second row of $\begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix}$ could be any $\begin{pmatrix} \alpha & \beta & \gamma \end{pmatrix}$ with α, β, γ distinct and nonzero, and the span program would still compute MAJ_3. The specific setting $(\alpha, \beta, \gamma) = (1, \omega, \omega^2)$ is used to optimize the running time of the quantum algorithm.

Remark 4.3. *Let us clarify a few points in Definition 4.1.*

- *Although other fields can be used, for us all vectors and coefficients are over the complex numbers* $\mathbf{C}$.
- *It is convenient, but nonstandard, to allow grouped inputs, i.e., literal subsets X_j with sizes possibly > 1, instead of just single literals, to label the columns. A grouped input j can be thought of as evaluating to 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 the size of P.*
- *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 always evaluates to true. However, it is a simple linear algebra exercise to see that such vectors can always be eliminated from P without increasing the size or changing t (or see [KW93, Theorem 7]).*
- *By a basis change, one can always change the target vector t to $(1, 0, 0, \ldots, 0)$.*

4.2 Span program as a weighted adjacency matrix

Written out as a matrix, a span program P corresponds to the weighted adjacency matrix of a certain graph.

Consider a span program P with target vector $t = (1, 0, \ldots, 0)$, and grouped input vectors $\{v_j : j \in J\}$.

Notation: For a tuple $H = (h_1, \ldots, h_{|H|})$ and a set of variables $\{a_h\}$, let $a_H = (a_{h_1}, \ldots, a_{h_{|H|}})$. Assigning an arbitrary order to the grouped inputs index set J, the grouped input vectors can be written in shorthand as v_J.

To avoid any potential confusion of subscripts in the following, let us define three more tuples: I ("inputs"), O ("output") and C ("constraints"). Let $\{I_j : j \in J\}$ be pairwise disjoint tuples such that I_j indexes X_j, and let I be the combined tuple of all I_j, $j \in J$ (in an arbitrary order). (Since $|I_j| = |X_j|$, $|I| = \text{size}(P)$.) Let $O = \{1\}$ be a singleton set indexing the first, "output" component of the grouped input vectors, and let C index the remaining components. We may assume that the index tuples/sets I, J, O, C are pairwise disjoint.

We will construct a graph G_P on $|I| + |J| + |C| + 2|O|$ vertices. Writing the grouped input vectors out as a matrix, let $\begin{pmatrix} A_{OJ} \\ A_{CJ} \end{pmatrix} = \begin{pmatrix} v_1 & \cdots & v_{|J|} \end{pmatrix}$; A_{OJ} is a $1 \times |J|$ matrix row, and A_{CJ} is a $|C| \times |J|$ matrix. We may assume $A_{OJ} \neq 0$; otherwise $f_P \equiv 0$. Let $A_{IJ} = \sum_{j \in J, i \in I_j} |i\rangle\langle j|$; A_{IJ} encodes the grouped inputs of P. Now consider the bipartite graph G_P of Figure 2, one half of whose weighted Hermitian adjacency matrix is

$$A = \begin{array}{c} \begin{array}{cc} a_O & \overbrace{\;\;a_J\;\;} \end{array} \\ \left(\begin{array}{cc} 1 & A_{OJ} \\ \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} & A_{CJ} \\ \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} & A_{IJ} \end{array} \right) \begin{array}{l} b_O \\ \}b_C \\ \}b_I \end{array} \end{array} \tag{4.1}$$

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 set of first

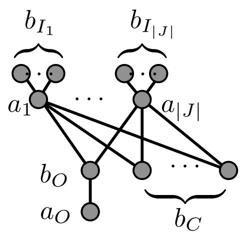

Figure 2: The bipartite graph G_P corresponding to span program P; the output edge is (a_O, b_O), while the grouped inputs are $a_1, \ldots, a_{|J|}$.

components 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 components of v_J).[2]

4.3 Span program duality

A span program P computing a function f_P immediately gives a dual span program, denoted $P^\dagger$, such that $f_{P^\dagger} = \neg f_P$ pointwise, i.e., for all $x \in \{0,1\}^n$, $f_{P^\dagger}(x) = \neg f_P(x)$. This will be useful later because it will allow us freely to insert NOT gates into the formula φ without worrying about any monotonicity constraints.

Definition 4.4 (Dual span program $P^\dagger$)**.** *Let P be a span program with target vector $t = (1, 0, \ldots, 0)$ and grouped input vectors v_j corresponding to grouped input $X_j \subseteq X$, for $j \in J$. As above, let $1 + |C|$ be the dimension of the vectors t and $\{v_j\}$, and recall the above definitions of A_{OJ}, A_{CJ}, A_{IJ}.*

Define span program $P^\dagger$ as follows: the target vector is $A^\dagger_{OJ}$, and the grouped input vectors are the columns of the matrix $\left({A_{CJ}}^\dagger \; {A_{IJ}}^\dagger \right)$. The first $|C|$ grouped input vectors—the columns of ${A_{CJ}}^\dagger$—are each associated with no literals. The other grouped input vectors—the columns of ${A_{IJ}}^\dagger$—are all of the form $|j\rangle$ for some $j \in J$, i.e., they have a single 1 component and $|J| - 1$ zero components. For each j there are $|I_j| = |X_j|$ constraints $|j\rangle$; associate each to the negation of an element in X_j.

Note that the size of $P^\dagger$ is the same as the size of P. The dimension of the vectors in $P^\dagger$ is $|J|$, the number of grouped input vectors in P. We have chosen the notation $P^\dagger$ because the target and grouped input vectors of $P^\dagger$ put together give the Hermitian conjugate of a portion of the matrix A given in Eq. (4.1). Different constructions of dual span programs have been given in [CF02, NNP05].

Lemma 4.5 (Span program duality)**.** *For all $x \in \{0,1\}^n$, $f_{P^\dagger}(x) = \neg f_P(x)$.*

Proof. The function $f_P(x)$ evaluates to zero if and only if the vector consisting of the true columns of A_{OJ} is linearly dependent on the rows of A_{CJ} restricted to the same columns. In $P^\dagger$ therefore, those entries of the target ${A_{OJ}}^\dagger$ are given for free. For each false column j there is at least one false input $i \in I_j$; this gives us $|j\rangle$, and arbitrary multiples of $|j\rangle$, in $P^\dagger$. $\square$

4.4 Zero-eigenvalue eigenvectors of the span program adjacency matrix

Lemma 4.6. *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. Then there exists a zero-eigenvalue eigenstate of the weighted adjacency matrix $A_{G_P(x)}$ with support on vertex a_O if and only if $f_P(x) = 1$. After deleting the output edge (a_O, b_O), there exists a zero-eigenvalue eigenstate of the remaining adjacency matrix with support on b_O if and only if $f_P(x) = 0$.*

[2]Of course, this correspondence also goes in the other direction, from bipartite graphs with certain marked input and output vertices, to span programs.

Proof. Notation: Use a_j, b_i, b_c, a_O, b_O to denote coefficients of a vector on the vertices of A_{G_P}. Let $A_{IJ}(x)$ include only the edges to true inputs. Then the eigenvalue-λ eigenvector equations of $A_{G_P(x)}$ are

$$\lambda b_C = A_{CJ} a_J \tag{4.2a}$$
$$\lambda b_O = A_{OJ} a_J + a_O \tag{4.2b}$$
$$\lambda a_J = A_{IJ}(x)^\dagger b_I + A_{OJ}{}^\dagger b_O + A_{CJ}{}^\dagger b_C \tag{4.2c}$$
$$\lambda b_I = A_{IJ}(x) a_J \tag{4.2d}$$
$$\lambda a_O = b_O \tag{4.2e}$$

Being a ($\lambda = 0$)-eigenvalue eigenvector means that for each vertex, the weighted sum of the coefficients on adjacent vertices must be zero. In particular, note that Eq. (4.2d) implies that a_j can be nonzero only if grouped input j is true. (If there is any dangling edge (a_j, b_i)—any false input in X_j—then Eq. (4.2d) says $a_j = 0$.) Since the graph is bipartite, at $\lambda = 0$ the a coefficients do not interact with the b coefficients.

Assume $f_P(x) = 1$. Then by Definition 4.1 there exists a linear combination $t = \sum_j a_j v_j = (A_{OJ} + A_{CJ}) a_J$ for which $a_j = 0$ if j is false. That is, $A_{OJ} a_J = 1$ and $A_{CJ} a_J = 0$. Setting $a_O = -1$ and $b_O = b_I = b_C = 0$, all the equations in (4.2) are satisfied.

Conversely, if $f_P(x) = 0$, then for every setting of a_J such that $a_j = 0$ for false j, $A_{CJ} a_J = 0$ implies $A_{OJ} a_J = 0$ (if $A_{CJ} a_J = 0$ and $A_{OJ} a_J \neq 0$, then we can scale all the coefficients so $A_{OJ} a_J = 1$, a contradiction). Therefore, Eq. (4.2b) gives $a_O = 0$.

Next, consider deleting the edge (a_O, b_O). We argue that there exists a zero-eigenvalue eigenstate of the remaining adjacency matrix with support on b_O if and only if $f_P(x) = 0$. Now, an input b_i can only possibly be nonzero if the ith input is false; if i is true, then b_i has been deleted. Lemma 4.5 for the dual span program $P^\dagger$ therefore implies that Eq. (4.2c) has a solution with b_O nonzero if and only if $f_{P^\dagger}(x) = \neg f_P(x) = 1$. The other equations can always be satisfied by letting $a_J = 0$. $\square$

Remark 4.7. *By Lemma 4.6, we can think of the graph G_P as giving a "dual-rail" encoding of the function f_P: there is a zero-eigenvalue eigenstate supported on a_O if and only if $f_P(x) = 1$, and there is one supported on b_O if and only if $f_P(x) = 0$. This justifies calling the edge (a_O, b_O) the output edge of G_P. Later, we will plug together multiple span program gadgets, by identifying output edges of some gadgets with the input edges of another.*

We will need stronger, more quantitative, statements than Lemma 4.6—namely, Lemma 6.2 and Theorem 5.6—but the lemma is still useful for giving intuition, motivation and direction.

5 Small-eigenvalue spectral analysis of A_{G_P}

In order to analyze the running time of a quantum algorithm based on span programs, we need to understand not only zero-eigenvalue eigenstates of $A_{G_P}(x)$, but also small-eigenvalue eigenstates. In this section, we will investigate eigenstates with eigenvalue $\lambda \neq 0$. Since the graph is bipartite, we may, and will, assume without loss of generality that $\lambda > 0$.

At small enough eigenvalues, the dual-rail encoding property described in Remark 4.7 still holds, but in a slightly different fashion. At $\lambda = 0$, we saw that, roughly, the existence of an eigenvector supported on a_O meant that $f_P(x) = 1$, while the existence of an eigenvector supported on b_O meant that $f_P(x) = 0$. For small enough $\lambda > 0$, it will turn out instead that the function's behavior corresponds to the output *ratio* $r_O \equiv a_O/b_O$. If $f_P(x) = 1$, then the output ratio r_O will turn out to be large and negative, roughly of order $-1/\lambda$. If $f_P(x) = 0$, then r_O will be small and positive, roughly of order λ. In this section, we will make this statement precise.

5.1 General span program spectral analysis

Definition 5.1 (Input ratios)**.** *Encode the input x to f_P in the coefficient ratios $r_i = a_j/b_i$ for $i \in I_j$ and $j \in J$, provided $a_j, b_i \neq 0$. If the literal associated to input i is false, then let $s_i = r_i/\lambda$; if it is true, then let*

$s_i = -1/(r_i\lambda)$. Assume that $0 < s_i \leq S_i$ for each i. If $a_j = b_i = 0$, then input i will not matter. To unify notation, though, define r_i in this case as either λS_i or $-1/(\lambda S_i)$ if i is true or false, respectively. Then $s_i = S_i$. The cases $a_j = 0$, $b_i \neq 0$ and $a_j \neq 0$, $b_i = 0$ will not arise.

In Lemma 4.6, we were given that either $a_i = 0$ or $b_i = 0$ according to whether input i evaluated to 0 or 1, which could be determined recursively by an earlier span program with output edge (a_i, b_i). Now, we are given the nonzero input ratios r_I, possibly determined by earlier span programs, and will solve for the output ratio r_O.

That is, letting R be a diagonal matrix with the ratios r_I along its diagonal, we are given

$$b_I = R^{-1} A_{IJ} a_J \ , \tag{5.1}$$

assuming that for no $j \in J$ and $i \in I_j$ is either $a_j = 0$, $b_i \neq 0$ or $a_j \neq 0$, $b_i = 0$. This assumption implies that for each j either $a_j \neq 0$ and all $b_i \neq 0$, $i \in I_j$, or $a_j = 0$ and all $b_i = 0$, $i \in I_j$.

The eigenvector equations for eigenvalue λ, at internal vertices of the span program graph A_{G_P} are

$$\begin{aligned} \lambda b_C &= A_{CJ} a_J && \text{(5.2a)} \\ \lambda b_O &= A_{OJ} a_J + a_O && \text{(5.2b)} \\ \lambda a_J &= A_{IJ}{}^\dagger b_I + A_{OJ}{}^\dagger b_O + A_{CJ}{}^\dagger b_C && \text{(5.2c)} \end{aligned}$$

We want to allow other graphs, for other span programs, to be attached to the input vertices b_I and to the output vertex a_O, so we will leave the equations at b_I and a_O to be solved in the adjacent span programs.

Now we will solve for the output ratio r_O. Letting $s_O = r_O/\lambda$ in case $f_P(x) = 0$, or $s_O = -1/(\lambda r_O)$ in case $f_P(x) = 1$, we aim to show that $0 < s_O \leq S_O$ for some S_O. This statement is weaker for larger S_O, but as long as $S_O < 1/\lambda$, it still allows for distinguishing case $f_P(x) = 0$ from $f_P(x) = 1$ based on $|r_O|$. We therefore hope that S_O is not too much larger than the input bounds S_I.

Lemma 5.2. *Let $r_O = a_O/b_O$ be the output ratio for the gadget of Eq. (4.1) and Figure 2, solving Eqs. (5.1) and (5.2). Let $\tilde{R} = (A_{IJ}{}^\dagger R^{-1} A_{IJ} - \lambda)^{-1}$ (a diagonal matrix). Then $a_O = 0$ if $b_O = 0$, and otherwise,*

$$r_O = \lambda + A_{OJ} \left(\tilde{R} - \frac{1}{\lambda} \tilde{R} A_{CJ}{}^\dagger (1 + \tfrac{1}{\lambda} A_{CJ} \tilde{R} A_{CJ}{}^\dagger)^{-1} A_{CJ} \tilde{R} \right) A_{OJ}{}^\dagger \ , \tag{5.3}$$

provided that $\tilde{R}$ and $(1 + \frac{1}{\lambda} A_{CJ} \tilde{R} A_{CJ}{}^\dagger)^{-1}$ exist.

Proof. Substituting Eqs. (5.1) and (5.2a) into (5.2c), and rearranging terms gives

$$\left(\lambda - A_{IJ}{}^\dagger R^{-1} A_{IJ} - \frac{1}{\lambda} A_{CJ}{}^\dagger A_{CJ} \right) a_J = A_{OJ}{}^\dagger b_O \ .$$

From Eq. (5.2b), if $b_O \neq 0$, then $a_O/b_O = \lambda - A_{OJ} a_J / b_O$, so

$$\begin{aligned} r_O &= \lambda + A_{OJ} (\tilde{R}^{-1} + \frac{1}{\lambda} A_{CJ}{}^\dagger A_{CJ})^{-1} A_{OJ}{}^\dagger && \text{(5.4)} \\ &= \lambda + A_{OJ} \left(\tilde{R} - \frac{1}{\lambda} \tilde{R} A_{CJ}{}^\dagger (1 + \tfrac{1}{\lambda} A_{CJ} \tilde{R} A_{CJ}{}^\dagger)^{-1} A_{CJ} \tilde{R} \right) A_{OJ}{}^\dagger \ , \end{aligned}$$

by the Woodbury matrix identity [GV96], provided that $\tilde{R}$ and $(1 + \frac{1}{\lambda} A_{CJ} \tilde{R} A_{CJ}{}^\dagger)^{-1}$ exist. □

Remark 5.3 (Form of Eq. (5.3))**.** *Note from Eq. (5.3) that r_O is a real number provided that all the input ratios r_I are themselves reals. Also, note that r_O depends on A_{CJ} only through $A_{CJ}{}^\dagger A_{CJ}$ (see too Eq. (5.4) in the proof); in particular, left-multiplying A_{CJ} by U where U is any linear isometry (i.e., satisfying $U^\dagger U = 1$) has no effect. Since the grouped input vectors v_J can be arbitrary in Definition 4.1, $A_{CJ}{}^\dagger A_{CJ}$ is in general an arbitrary $k \times k$ positive semidefinite matrix.*

In fact, $\tilde{R}$ does exist provided λ is small enough. $\tilde{R}_{jj}$ for a grouped input j is analogous to an input ratio r_i: $\tilde{R}_{jj}$ is large and negative if j is true, and small and positive if j is false:

Lemma 5.4. *Assume that $0 < \lambda \leq \epsilon / S_i$ for an arbitrarily small positive constant ϵ and for all $i \in I$. Then $\tilde{R} = ({A_{IJ}}^\dagger R^{-1} A_{IJ} - \lambda)^{-1}$ exists. Moreover, consider any grouped input $j \in J$.*

- *If j evaluates to true, i.e., all inputs in I_j are true, let $\tilde{s}_j = -1/(\lambda \tilde{R}_{jj})$ and let $\tilde{S}_j = 1 + \sum_{i \in I_j} S_i$.*
- *If j is false, i.e., at least one $i \in I_j$ is false, let $\tilde{s}_j = \tilde{R}_{jj}/\lambda$ and let $\tilde{S}_j = \big(\sum_{\text{false } i \in I_j} S_i^{-1}\big)^{-1}(1 + \epsilon' \lambda \max_{\text{false } i \in I_j} S_i)$, where ϵ' is an arbitrarily small positive constant.*

Then in each case, $0 < \tilde{s}_j \leq \tilde{S}_j$.

Proof. By definition,

$$\tilde{R}_{jj} = (-\lambda + \sum_{i \in I_j} r_i^{-1})^{-1} = \Big(-\lambda - \lambda \sum_{\text{true } i \in I_j} s_i + \frac{1}{\lambda} \sum_{\text{false } i \in I_j} s_i^{-1} \Big)^{-1} .$$

If all inputs in I_j are true, then

$$\tilde{s}_j = -1/(\lambda \tilde{R}_{jj}) = 1 + \sum_{i \in I_j} s_i \ ,$$

so $1 < \tilde{s}_j \leq 1 + \sum_{i \in I_j} S_i = \tilde{S}_j$.

Now assume at least one input in I_j is false. The true terms can be upper-bounded by $\lambda \sum_{\text{true } i \in I_j} s_i \leq \lambda \sum_{i \in I_j} S_i \leq |I_j| \epsilon$. On the other hand, if i is false then $(\lambda s_i)^{-1} \geq (\lambda S_i)^{-1} \geq 1/\epsilon$. Therefore, $\tilde{s}_j > 0$, and we also get the claimed upper bound. ☐

5.2 Two-dimensional span program spectral analysis

While software algebra packages like Mathematica can be used to investigate Eq. (5.3) (or Eqs. (5.1), (5.2) directly), it turns out that considerable simplifications can be made in the case where there is at most one constraint, $|C| \leq 1$. Now let us specialize to this case. Then $(1 + \frac{1}{\lambda} A_{CJ} \tilde{R} {A_{CJ}}^\dagger)^{-1}$ in Eq. (5.3) is simply a scalar inverse. (In this case, the Woodbury formula says that $(1-M)^{-1} = 1 + M/(1 - \operatorname{tr} M)$ for M any rank-one square matrix with $\operatorname{tr} M \neq 1$.)

Definition 5.5 (Single-constraint notation)**.** *If $|C| \leq 1$, let $|c\rangle = {A_{CJ}}^\dagger = \sum_j c_j |j\rangle$ and let $|o\rangle = {A_{OJ}}^\dagger = \sum_j o_j |j\rangle \neq 0$. (If there are no constraints, $|C| = 0$, then set $|c\rangle = 0$.) Write the diagonal matrix $\tilde{R}$ as $\tilde{R} = \sum_j \tilde{R}_{jj} |j\rangle\langle j|$. Finally, let $d_{jj'} = c_{j'} o_j - c_j o_{j'} = \operatorname{Det} \left(\begin{smallmatrix} o_j & o_{j'} \\ c_j & c_{j'} \end{smallmatrix}\right)$.*

Our main goal of this section is to prove Theorem 5.6 below. Theorem 5.6 simplifies and carefully bounds higher-order in λ terms in Eq. (5.3). For each of three different cases, it gives an expression for the leading term that does not depend on λ and into which one can just plug in values of $|o\rangle$ and $|c\rangle$.

Theorem 5.6 (Single-constraint spectral analysis)**.** *Assume that $0 < \lambda \leq \epsilon / S_i$ for an arbitrarily small positive constant ϵ and for all $i \in I$. Assume that there is at most one constraint, $|C| \leq 1$. Then either $a_O = b_O = 0$ or $a_O, b_O \neq 0$.*

- *If $f_P(x)$ is true, recall that $s_O = \frac{-1}{\lambda r_O}$. Let $|c(x)\rangle = \sum_{\text{true } j} \frac{c_j}{\sqrt{\tilde{S}_j}} |j\rangle$ and $|o(x)\rangle = \sum_{\text{true } j} \frac{o_j}{\sqrt{\tilde{S}_j}} |j\rangle$. Then*

$$s_O = \frac{-1}{\lambda r_O} \leq \left[\langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle} \Big) |o(x)\rangle \right]^{-1} (1 + \textit{constant} \cdot \lambda \max_j \tilde{S}_j) \ . \qquad (5.5)$$

- *If $f_P(x)$ is false, recall $s_O = r_O/\lambda$. Then,*
  - *If all grouped inputs are false, then*

$$s_O = \mathit{constant} + \langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle}\Big) |o(x)\rangle \ , \tag{5.6}$$

where $|o(x)\rangle = \sum_j \sqrt{\tilde{S}_j} o_j |j\rangle$ and $|c(x)\rangle = \sum_j \sqrt{\tilde{S}_j} c_j |j\rangle$.

  - *If there is at least one true grouped input j', then*

$$s_O \leq \mathit{constant} + \frac{1}{|c_{j'}|^2} \sum_{\mathit{false}\ j} |d_{jj'}|^2 \tilde{S}_j \left(1 + \epsilon' \lambda \max_j \tilde{S}_j\right) \ . \tag{5.7}$$

In the above equations, ϵ' is an arbitrarily small constant, and constants depend only on the weights $|o\rangle, |c\rangle$, not on the bounds $\tilde{S}_J$.

Remark 5.7. *For a vector $|v\rangle$ and a projection Π, the expression $\langle v|(1 - \Pi)|v\rangle$ is the squared distance of endpoint of $|v\rangle$ to the subspace defined by Π. In particular, this distance can only increase if one scales up any coordinate of the ambient space. Therefore, the right-hand sides of the expressions in Theorem 5.6 are all monotone increasing functions of each $\tilde{S}_j$.*

Remark 5.8. *The true case of Theorem 5.6 involves only $\tilde{S}_j$ for true inputs j, and the false cases involve only $\tilde{S}_j$ for false inputs j. This is similar to Lemma 4.6, and is because the graph is bipartite. If the output is true, then false inputs contribute only higher-order terms that are easily bounded, and vice versa. We conjecture that the same simplification should hold in the general case Eq. (5.3).*

Remark 5.9. *By assuming $|C| \leq 1$, we reduce the expressive power of span programs of a given complexity. Because we are allowing grouped inputs with $|I_j| > 1$ possibly, any function can be computed by a span program with $|C| = 0$, based on the disjunctive normal form (DNF) formula expansion. However the DNF span program may not be optimal. It turns out that span programs with $|C| \leq 1$ do suffice to compute all three-bit functions optimally (Section 9). For four-bit functions, this seems no longer to hold, and in Section 12.1 we discuss analyses of larger span programs carried out with Mathematica.*[3] *We suspect that Theorem 5.6 can be generalized, and conjecture in particular that Eqs. (5.5) and (5.6) hold even for $|C| > 1$, with the projections $|c(x)\rangle\langle c(x)|/\langle c(x)|c(x)\rangle$ replaced by projections onto the analogous subspace. (See also Lemma 6.2.)*

Lemma 5.10. *Assume that there is at most one constraint, $|C| \leq 1$. Then the output ratio $r_O = a_O/b_O$ is*

$$r_O = \frac{\frac{1}{2}\sum_{j,j'} |d_{jj'}|^2 \tilde{R}_{jj} \tilde{R}_{j'j'} + \lambda \langle o|\tilde{R}|o\rangle}{\langle c|\tilde{R}|c\rangle + \lambda} + \lambda \ , \tag{5.8}$$

provided that $\tilde{R}$ exists, and that $\langle c|\tilde{R}|c\rangle = \sum_j |c_j|^2 \tilde{R}_{jj} \neq -\lambda$.

Proof. Provided that $\tilde{R}$ exists and $\langle c|\tilde{R}|c\rangle \neq -\lambda$, we have from Eq. (5.3) that

$$\begin{aligned} r_O &= \lambda + \langle o| \left(\tilde{R} - \frac{\tilde{R}|c\rangle\langle c|\tilde{R}}{\langle c|\tilde{R}|c\rangle + \lambda}\right) |o\rangle \\ &= \lambda + \frac{\langle c|\tilde{R}|c\rangle\langle o|\tilde{R}|o\rangle - |\langle c|\tilde{R}|o\rangle|^2 + \lambda\langle o|\tilde{R}|o\rangle}{\langle c|\tilde{R}|c\rangle + \lambda} \ . \end{aligned} \tag{5.9}$$

[3] Another generalization that is often helpful in practice is to allow variable weights on the inputs in an I_j. An equivalent effect can be achieved, though, by inserting two NOT gates before each input, and adjusting the ratio of the edge weights. Here, we will assume that all input and output edges always have weight one.

The numerator can be further simplified:

$$\begin{aligned}
\langle c|\tilde{R}|c\rangle\langle o|\tilde{R}|o\rangle - |\langle c|\tilde{R}|o\rangle|^2 &= \sum_{j,j'} \tilde{R}_{jj}\tilde{R}_{j'j'}|c_j|^2|o_{j'}|^2 - \Big|\sum_j \tilde{R}_{jj} c_j^* o_j\Big|^2 \\
&= \sum_{j,j'} \tilde{R}_{jj}\tilde{R}_{j'j'}\left(|c_j o_{j'}|^2 - (c_j o_{j'})^*(c_{j'} o_j)\right) \\
&= \tfrac{1}{2}\sum_{j,j'} \tilde{R}_{jj}\tilde{R}_{j'j'}\begin{pmatrix} |c_j o_{j'}|^2 - (c_j o_{j'})^*(c_{j'} o_j) \\ +|c_{j'} o_j|^2 - (c_{j'} o_j)^*(c_j o_{j'}) \end{pmatrix} \\
&= \tfrac{1}{2}\sum_{j,j'} \tilde{R}_{jj}\tilde{R}_{j'j'}|d_{jj'}|^2 \ ,
\end{aligned} \tag{5.10}$$

where $d_{jj'} = c_{j'} o_j - c_j o_{j'}$ (Definition 5.5). Substituting (5.10) back into (5.9), we obtain Eq. (5.8). □

Let us now justify the two provisions of Lemma 5.10. In the proof of Corollary 5.11, and in the rest of our computations, we will frequently use ϵ, ϵ', etc., to mean arbitrarily small positive constants.

Corollary 5.11. *Assuming that $|C| \leq 1$ and λS_i is at most an arbitrarily small constant for all $i \in I$: $\tilde{R}_{jj}$ exists for all $j \in J$, and $\langle c|\tilde{R}|c\rangle \neq -\lambda$.*

Proof. $\tilde{R}_{jj}$ exists by Lemma 5.4. To show $\langle c|\tilde{R}|c\rangle \neq -\lambda$, we will use the assumption that the weights c_j are constants. Each $|c_j|$ is upper-bounded by a constant, and either $c_j = 0$ or $|c_j|$ is lower-bounded by a positive constant.

Now, $\langle c|\tilde{R}|c\rangle = \sum_j |c_j|^2 \tilde{R}_{jj} = \sum_{\text{true } j} |c_j|^2 \frac{-1}{\lambda \tilde{s}_j} + \sum_{\text{false } j} |c_j|^2 \lambda \tilde{s}_j$. If all the grouped inputs j are false, then $\langle c|\tilde{R}|c\rangle > 0$. If there is at least one true term with nonzero coefficient, though, then upper-bound $\sum_{\text{false } j} |c_j|^2 \lambda \tilde{s}_j \leq \epsilon'$ for an arbitrarily small constant ϵ' This uses that each $\lambda \tilde{s}_j \leq \epsilon$, that the coefficients $|c_j|^2$ are bounded by constants, and that there are only a constant number of terms. On the other hand, $\sum_{\text{true } j} |c_j|^2 \frac{-1}{\lambda \tilde{s}_j} \leq -1/\epsilon''$, since each nonzero coefficient is bounded away from zero. Therefore, $\langle c|\tilde{R}|c\rangle \leq -1/\epsilon''' < -\lambda$. □

Proof of Theorem 5.6. **Case O is true:** If the output is true, then either $d_{jj'} \neq 0$ for some true j, j', or $d_{jj'} = 0$ for all true j, j'. In the latter case, the vectors $\{(o_j, c_j) : j \text{ true}\}$ are all proportional to each other; for the output to be true it must be that $c_j = 0$ for all true j. Since we may assume that for no j is $c_j = o_j = 0$, for then the span program would not depend on j, we may assume $o_j \neq 0$ for all true j.

From Eq. (5.8), we are trying to upper-bound

$$s_O = \frac{-1}{\lambda r_O} = \frac{-\lambda - \sum_j |c_j|^2 \tilde{R}_{jj}}{\frac{\lambda}{2}\sum_{j,j'} |d_{jj'}|^2 \tilde{R}_{jj}\tilde{R}_{j'j'} + \lambda^2 \sum_j (|c_j|^2 + |o_j|^2)\tilde{R}_{jj} + \lambda^3} \ . \tag{5.11}$$

- First, assume that there exist true j, j', such that $d_{jj'} = \text{Det}\left(\begin{smallmatrix} o_j & o_{j'} \\ c_j & c_{j'} \end{smallmatrix}\right) \neq 0$.
  The numerator and denominator in Eq. (5.11) are both positive, from Lemma 5.4, so $s_O > 0$. Drop negative terms from the numerator, and positive terms besides those multiplying $|d_{jj'}|^2$ with j, j' both true from the denominator. This leaves

$$\begin{aligned}
s_O &\leq \frac{-\sum_{\text{true } j} |c_j|^2 \tilde{R}_{jj}}{\frac{\lambda}{2}\sum_{\text{true } j,j'} |d_{jj'}|^2 \tilde{R}_{jj}\tilde{R}_{j'j'} + \lambda^2 \sum_{\text{true } j} (|c_j|^2 + |o_j|^2)\tilde{R}_{jj}} \\
&= \frac{\frac{1}{\lambda}\sum_{\text{true } j} |c_j|^2 \frac{1}{\tilde{s}_j}}{\frac{1}{2\lambda}\sum_{\text{true } j,j'} |d_{jj'}|^2 \frac{1}{\tilde{s}_j \tilde{s}_{j'}} - \lambda \sum_{\text{true } j} (|c_j|^2 + |o_j|^2)\frac{1}{\tilde{s}_j}}
\end{aligned}$$

Let $t_1 = \sum_{\text{true } j} |c_j|^2 \frac{1}{\tilde{s}_j}$, $t_2 = \sum_{\text{true } j} |o_j|^2 \frac{1}{\tilde{s}_j}$, and $t_3 = \frac{1}{2} \sum_{\text{true } j,\, j'} |d_{jj'}|^2 \frac{1}{\tilde{s}_j \tilde{s}_{j'}}$. Assume that $t_1/t_3, t_2/t_3 \leq \frac{\epsilon'}{\lambda}$, where ϵ' is an arbitrarily small constant. Then, we get

$$\begin{aligned} s_O &\leq \frac{t_1}{t_3 - \lambda^2(t_1 + t_2)} \\ &= \frac{t_1}{t_3}\left(1 + \lambda^2 \frac{t_1/t_3 + t_2/t_3}{1 - \lambda^2(t_1/t_3 + t_2/t_3)}\right) \\ &\leq \frac{t_1}{t_3}(1 + \lambda\epsilon) \ , \end{aligned} \tag{5.12}$$

which is almost our goal, Eq. (5.5). By Eq. (5.10), t_3/t_1 can be interpreted as the squared distance from a vector to a subspace. However, to get from Eq. (5.12) to Eq. (5.5), we need to show that t_1/t_3 is maximized by setting each $\tilde{s}_j$ to $\tilde{S}_j$. This is geometrically clear, because stretching a vector space can only increase the distance from a point to a certain subspace.

Therefore, it suffices to bound crudely $t_1/t_3, t_2/t_3 \leq \epsilon'/\lambda$. Since t_2/t_3 has the same form as t_1/t_3, except with c replaced by o, we will only consider t_1/t_3:

$$\begin{aligned} \frac{t_1}{t_3} &= \frac{\sum_{\text{true } j} |c_j|^2 \frac{1}{\tilde{s}_j}}{\frac{1}{2}\sum_{\text{true } j,\, j'} |d_{jj'}|^2 \frac{1}{\tilde{s}_j \tilde{s}_{j'}}} \\ &\leq \sum_{\text{true } j} |c_j|^2 \min_{\text{true } j'} \frac{\tilde{s}_{j'}}{|d_{jj'}|^2} \ . \end{aligned}$$

The denominator in the above expression is nonzero for each j. Indeed, fix some true j', j'' such that $d_{j'j''} \neq 0$. (Equivalently, the vectors $(o_{j'}, c_{j'})$ and $(o_{j''}, c_{j''})$ are linearly independent of each other.) Then if j is true and $(o_j, c_j) \neq (0, 0)$, at least one of $d_{jj'}, d_{jj''}$ must be nonzero. Therefore, using the small-λ assumption, that $\tilde{s}_j \leq \epsilon''/\lambda$, we get $t_1/t_3 \leq \epsilon'/\lambda$. Similarly, $t_2/t_3 \leq \epsilon'/\lambda$, so Eq. (5.12) holds.

- Next, consider the case that all true j have $c_j = 0$ and $o_j \neq 0$. In particular, $d_{jj'} = 0$ if j and j' are both true, and $d_{jj'} = o_j c_{j'}$ if j is true and j' false.

  In this case, Eq. (5.11) simplifies to

$$\begin{aligned} s_O &= \frac{\lambda + \sum_{\text{false } j} |c_j|^2 \tilde{R}_{jj}}{\left(\begin{array}{c} -\lambda \sum\limits_{\substack{\text{true } j \\ \text{false } j'}} |o_j|^2 |c_{j'}|^2 \tilde{R}_{jj} \tilde{R}_{j'j'} - \frac{\lambda}{2} \sum\limits_{\text{false } j,\, j'} |d_{jj'}|^2 \tilde{R}_{jj} \tilde{R}_{j'j'} \\ -\lambda^2 \sum\limits_{\text{true } j} |o_j|^2 \tilde{R}_{jj} - \lambda^2 \sum\limits_{\text{false } j} (|c_j|^2 + |o_j|^2) \tilde{R}_{jj} - \lambda^3 \end{array}\right)} \\ &= \frac{1 + \sum_{\text{false } j} |c_j|^2 \tilde{s}_j}{\left(\begin{array}{c} \sum\limits_{\substack{\text{true } j \\ \text{false } j'}} |o_j|^2 |c_{j'}|^2 \frac{\tilde{s}_{j'}}{\tilde{s}_j} + \sum\limits_{\text{true } j} |o_j|^2/\tilde{s}_j \\ -\frac{\lambda^2}{2} \sum\limits_{\text{false } j,\, j'} |d_{jj'}|^2 \tilde{s}_j \tilde{s}_{j'} - \lambda^2 \sum\limits_{\text{false } j} (|c_j|^2 + |o_j|^2) \tilde{s}_j - \lambda^2 \end{array}\right)} \ . \end{aligned}$$

Note that by Lemma 5.4, $\lambda^2 \sum_{\text{false } j} (|c_j|^2 + |o_j|^2) \tilde{s}_j / \sum_{\text{true } j} (|o_j|^2/\tilde{s}_j) \leq \lambda\epsilon \max_j \tilde{S}_j$. Also, since

$|d_{jj'}| \leq 2\max\{|c_j o_{j'}|, |c_{j'} o_j|\}$, we may bound

$$\begin{aligned}
\frac{\frac{\lambda^2}{2}\sum_{\text{false } j,\, j'} |d_{jj'}|^2 \tilde{s}_j \tilde{s}_{j'}}{\sum_{\substack{\text{true } j'' \\ \text{false } j'}} |o_{j''}|^2 |c_{j'}|^2 \frac{\tilde{s}_{j'}}{\tilde{s}_{j''}}} &\leq \frac{4\lambda^2 \sum_{\text{false } j,\, j'} |c_{j'}|^2 |o_j|^2 \tilde{s}_j \tilde{s}_{j'}}{\sum_{\substack{\text{true } j'' \\ \text{false } j'}} |o_{j''}|^2 |c_{j'}|^2 \frac{\tilde{s}_{j'}}{\tilde{s}_{j''}}} \\
&\leq 4\lambda^2 \sum_{\text{false } j,\, j'} \frac{\cancel{|c_{j'}|^2} |o_j|^2 \tilde{s}_j \cancel{\tilde{s}_{j'}}}{|o_{j''}|^2 \cancel{|c_{j'}|^2} \cancel{\tilde{s}_{j'}} / \tilde{s}_{j''}} \leq \lambda \epsilon \max_j \tilde{S}_j \ ,
\end{aligned}$$

where we have fixed some true j'' in the second inequality. Therefore, $s_O > 0$, and we can drop all the negative terms from the denominator, including $-\lambda$, at a cost of a multiplicative factor of at most $(1 + \text{constant} \cdot \lambda \max_j \tilde{S}_j)$:

$$\begin{aligned}
s_O &\leq \frac{1 + \sum_{\text{false } j'} |c_{j'}|^2 \tilde{s}_{j'}}{\sum_{\substack{\text{true } j \\ \text{false } j'}} |o_j|^2 |c_{j'}|^2 \frac{\tilde{s}_{j'}}{\tilde{S}_j} + \sum_{\text{true } j} |o_j|^2 / \tilde{S}_j} (1 + \text{constant} \cdot \lambda \max_j \tilde{S}_j) \\
&= \frac{1}{\sum_{\text{true } j} |o_j|^2 / \tilde{S}_j} (1 + \text{constant} \cdot \lambda \max_j \tilde{S}_j) \ , \qquad (5.13)
\end{aligned}$$

by canceling a factor of $(1 + \sum_{\text{false } j'} |c_{j'}|^2 \tilde{s}_{j'})$.

Case O is false: Otherwise the output is false. Therefore, there are no true j with $c_j = 0$ and $o_j \neq 0$; and for all true j, j', $d_{jj'} = 0$. (Consider a graph on J with an edge (j, j') when $d_{jj'} = 0$. Then the graph consists of disjoint cliques. All the true variables need to lie within one of the cliques for $f_P(x)$ to be false.)

We have from Eq. (5.8)

$$s_O = \frac{r_O}{\lambda} = 1 + \frac{\frac{1}{2\lambda}\sum_{j,j'} |d_{jj'}|^2 \tilde{R}_{jj} \tilde{R}_{j'j'} + \sum_j |o_j|^2 \tilde{R}_{jj}}{\lambda + \sum_j |c_j|^2 \tilde{R}_{jj}} \ . \qquad (5.14)$$

There are once again two cases. Either all inputs j are false, or there exists at least one true input j with $c_j \neq 0$.

- First, consider the case where all inputs are false. Then both the numerator and the denominator in Eq. (5.14) are positive, by Lemma 5.4, so $s_O > 0$. We may simplify Eq. (5.14) to

  $$s_O = 1 + \frac{\frac{1}{2}\sum_{j,j'} |d_{jj'}|^2 \tilde{s}_j \tilde{s}_{j'} + \sum_j |o_j|^2 \tilde{s}_j}{1 + \sum_j |c_j|^2 \tilde{s}_j} \ .$$

  We may bound $(\sum_{j: c_j \neq 0} |o_j|^2 \tilde{s}_j) / (\sum_j |c_j|^2 \tilde{s}_j) \leq \sum_{j: c_j \neq 0} \frac{|o_j|^2}{|c_j|^2}$ is at most a constant. Therefore, we obtain

  $$s_O \leq \text{const.} + \frac{\frac{1}{2}\sum_{\substack{j: c_j \neq 0 \\ j': c_{j'} \neq 0}} |d_{jj'}|^2 \tilde{s}_j \tilde{s}_{j'} + \sum_{\substack{j: c_j = 0 \\ j': c_{j'} \neq 0}} |d_{jj'}|^2 \tilde{s}_j \tilde{s}_{j'} + \sum_{j: c_j = 0} |o_j|^2 \tilde{s}_j}{1 + \sum_j |c_j|^2 \tilde{s}_j} \ ,$$

  where we have used that $d_{jj'} = 0$ if $c_j = c_{j'} = 0$. Since $d_{jj'} = o_j c_{j'}$ if $c_j = 0$, the second two terms in the numerator share a common factor of $\sum_{j: c_j = 0} |o_j|^2 \tilde{s}_j$, whence

  $$\begin{aligned}
  s_O &\leq \text{const.} + \frac{\frac{1}{2}\sum_{\substack{j: c_j \neq 0 \\ j': c_{j'} \neq 0}} |d_{jj'}|^2 \tilde{s}_j \tilde{s}_{j'}}{\sum_j |c_j|^2 \tilde{s}_j} + \sum_{j: c_j = 0} |o_j|^2 \tilde{s}_j \qquad (5.15) \\
  &= \text{const.} + \frac{\frac{1}{2}\sum_{j,j'} |d_{jj'}|^2 \tilde{s}_j \tilde{s}_{j'}}{\sum_j |c_j|^2 \tilde{s}_j} \ .
  \end{aligned}$$

It remains to maximize over $\{\tilde{s}_j\}$. Since the second term in the above sum is $\langle o(x)|(1 - |c(x)\rangle\langle c(x)|/\langle c(x)|c(x)\rangle)|o(x)\rangle$ with $|c(x)\rangle = \sum_j \sqrt{\tilde{s}_j} c_j |j\rangle$ and $|o(x)\rangle = \sum_j \sqrt{\tilde{s}_j} o_j |j\rangle$ (see Eq. (5.10)), the maximum is achieved with each $\tilde{s}_j = \tilde{S}_j$.

- Next, consider the case that there is at least one true input. We may assume that each true j has $c_j \neq 0$—otherwise, if $c_j = 0$ and $o_j \neq 0$ the output would be true. Also, if j and j' are both true, then $d_{jj'}$ must be zero. Therefore, all the vectors $\{(o_j, c_j) : j \text{ true}\}$ lie in the same one-dimensional space. We may fix some true j'' and write $(o_j, c_j) = \alpha_j (o_{j''}, c_{j''})$ for j true.

  By Lemma 5.4, both the numerator and the denominator in Eq. (5.14) are negative, so $s_O > 0$. In the numerator of Eq. (5.14), upper-bound $-\sum_j |o_j|^2 \tilde{R}_{jj} \leq \sum_{\text{true } j} |o_j|^2 \frac{1}{\lambda \tilde{s}_j}$ by dropping negative terms. Upper-bound $-\frac{1}{2\lambda} \sum_{j,j'} |d_{jj'}|^2 \tilde{R}_{jj} \tilde{R}_{j'j'} \leq \frac{1}{\lambda} \sum_{\substack{\text{true } j \\ \text{false } j'}} |d_{jj'}|^2 \frac{\tilde{s}_{j'}}{\tilde{s}_j}$. In the denominator, bound $\lambda \sum_{\text{false } j} |c_j|^2 \tilde{s}_j \leq \epsilon$. Altogether, we have after canceling factors of $1/\lambda$,

$$s_O \leq 1 + \frac{\sum_{\substack{\text{true } j \\ \text{false } j'}} |d_{jj'}|^2 \frac{\tilde{s}_{j'}}{\tilde{s}_j} + \sum_{\text{true } j} |o_j|^2 \frac{1}{\tilde{s}_j}}{\sum_{\text{true } j} |c_j|^2 \frac{1}{\tilde{s}_j} - \lambda \epsilon}$$

  Since $c_j \neq 0$ for each true j, the ratio $(\sum_{\text{true } j} |o_j|^2 \frac{1}{\tilde{s}_j}) / (\sum_{\text{true } j} |c_j|^2 \frac{1}{\tilde{s}_j}) \leq \sum_{\text{true } j} |o_j|^2 / |c_j|^2$ is a constant. Since also $\sum_{\text{true } j} |c_j|^2 \frac{1}{\tilde{s}_j} \geq \text{constant} / \max_{\text{true } j} \tilde{S}_j$, we conclude

$$\begin{aligned} s_O &\leq \text{constant} + \frac{\sum_{\substack{\text{true } j \\ \text{false } j'}} |d_{jj'}|^2 \frac{\tilde{s}_{j'}}{\tilde{s}_j}}{\sum_{\text{true } j} |c_j|^2 / \tilde{s}_j} \left(1 + \epsilon' \lambda \max_{\text{true } j} \tilde{S}_j\right) \\ &= \text{constant} + \frac{\sum_{\substack{\text{true } j \\ \text{false } j'}} |\alpha_j|^2 |d_{j''j'}|^2 \frac{\tilde{s}_{j'}}{\tilde{s}_j}}{\sum_{\text{true } j} |\alpha_j|^2 |c_{j''}|^2 / \tilde{s}_j} \left(1 + \epsilon' \lambda \max_{\text{true } j} \tilde{S}_j\right) \\ &= \text{constant} + \frac{1}{|c_{j''}|^2} \sum_{\text{false } j'} |d_{j''j'}|^2 \tilde{s}_{j'} \left(1 + \epsilon' \lambda \max_{\text{true } j} \tilde{S}_j\right) \quad , \end{aligned}$$

  by canceling a factor of $\sum_{\text{true } j} |\alpha_j|^2 / \tilde{s}_j$ in the last step. Eq. (5.7) follows by $\tilde{s}_{j'} \leq \tilde{S}_{j'}$. □

This concludes our general analysis for $\lambda \neq 0$. In following sections, we will upper-bound the right-hand sides of Eqs. (5.5), (5.6) and (5.7) by substituting specific choices of $|c\rangle$ and $|o\rangle$.

6 Quantitative eigenvalue-zero spectral analysis

We need a stronger version of Lemma 4.6, that puts quantitative lower bounds on γ_O, the achievable squared magnitude, in a unity-normalized eigenvalue-zero eigenvector, on the output node either a_O if $f_P(x) = 1$, or b_O if $f_P(x) = 0$.

In this analysis, the achievable squared magnitude γ_O will be analogous to the upper-bound S_O in the small λ analysis. In fact, it will turn out that the dependence of γ_O on the input γ_i's is identical to the dependence of $1/S_O$ on the input $1/S_i$'s in Theorem 5.6, except for different "fudge factors."

Similarly to Lemma 5.4, let us start by grouping together and renormalizing the eigenvalue-zero eigenvectors of graphs entering at a single grouped input $j \in J$. Recall Figure 2 and Eq. (5.2). The vertex a_j is a shared output node of all the inputs $i \in I_j$, which can have support if and only if all $i \in I_j$ evaluate to true.

Lemma 6.1. *If grouped input j evaluates to true, then the achievable squared magnitude on a_j in a normalized eigenvalue-zero eigenvector of the vertices above j is at least γ_j, where*

$$\frac{1}{\gamma_j} \leq 1 + \sum_{i \in I_j} \frac{1}{\gamma_i} \quad . \tag{6.1}$$

If grouped input j evaluates to false, then the achievable squared total of magnitudes on all $\{b_i : i \in I_j\}$ in a normalized eigenvalue-zero eigenvector of the vertices above j is at least γ_j, where

$$\gamma_j = \sum_{\textit{false } i \in I_j} \gamma_i \ . \tag{6.2}$$

Proof. Grouped input j evaluates to true when all $i \in I_j$ are true. Scale all the input graphs meeting at a_j so that their coefficient at a_j is 1. Then the total norm is

$$\frac{1}{\gamma_j} = 1 + \sum_{i \in I_j} \frac{1}{\gamma_i}(1 - \gamma_i) = \sum_{i \in I_j} \frac{1}{\gamma_i} - (|I_j| - 1) \leq 1 + \sum_{i \in I_j} \frac{1}{\gamma_i} \ .$$

Grouped input j evaluates to false when at least one $i \in I_j$ is false. Now a_j is zero necessarily, so the eigenvectors for the different false $i \in I_j$ can be scaled by arbitrary fractions g_i. (For true inputs $i \in I_j$, the entire subgraph attached to b_i is zeroed out.) Our goal is to maximize the sum $\sum_{\text{false } i \in I_j} g_i \sqrt{\gamma_i}$ subject to the constraint that the total norm is one: $\sum_i |g_i|^2 = 1$. The maximum is clearly achieved for $g_i = \sqrt{\gamma_i} / \sqrt{\sum_{\text{false } i'} \gamma_{i'}}$, so the squared maximum is:

$$\gamma_j = \sum_{\text{false } i \in I_j} \gamma_i \ . \qquad \square$$

Lemma 6.2. *The achievable squared magnitude γ_O on the output node in a unity-normalized eigenvalue-zero eigenvector satisfies:*

- *If the output $f_P(x)$ is true, then*

$$1/\gamma_O \leq \Big[\langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle}\Big) |o(x)\rangle\Big]^{-1} + \textit{constant} \ , \tag{6.3}$$

  where $|c(x)\rangle = \sum_{\textit{true } j} \sqrt{\gamma_j} c_j |j\rangle$ and $|o(x)\rangle = \sum_{\textit{true } j} \sqrt{\gamma_j} o_j |j\rangle$.

- *If the output is false, and all inputs j are false, then*

$$1/\gamma_O \leq \langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle}\Big) |o(x)\rangle + \textit{constant} \ , \tag{6.4}$$

  where $|c(x)\rangle = \sum_{\textit{false } j} \frac{c_j}{\sqrt{\gamma_j}} |j\rangle$ and $|o(x)\rangle = \sum_{\textit{false } j} \frac{o_j}{\sqrt{\gamma_j}} |j\rangle$.

- *If the output is false, and there is at least one true input j', then*

$$1/\gamma_O \leq \frac{1}{|c_{j'}|^2} \sum_{\textit{false } j} |d_{jj'}|^2 / \gamma_j + \textit{constant} \ . \tag{6.5}$$

Proof. **Case O is true:** If $f_P(x) = 1$, then we want to choose g_J so as to maximize the magnitude of $-a_O = A_{OJ} a_J = \sum_{\text{true } j} o_j^* g_j \sqrt{\gamma_j} = \langle o(x)|g\rangle$, such that $A_{CJ} a_J = \sum_{\text{true } j} c_j^* g_j \sqrt{\gamma_j} = \langle c(x)|g\rangle = 0$ and $\sum_j |g_j|^2 = \langle g|g\rangle = 1$ (Eqs. (5.2a) and (5.2b) at $\lambda = 0$). Clearly the maximum is achieved for $|g\rangle$ proportional to $|\tilde{g}\rangle = \big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle}\big)|o(x)\rangle$. Then

$$|a_O|^2 = \frac{|\langle o(x)|\tilde{g}\rangle|^2}{\langle \tilde{g}|\tilde{g}\rangle} = \langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle}\Big) |o(x)\rangle \ .$$

However, we have not fully normalized. We need to divide further by $(1 + |a_O|^2)$. We obtain

$$\frac{1}{\gamma_O} = \frac{1 + |a_O|^2}{|a_O|^2} = \frac{1}{|a_O|^2} + 1 \ .$$

Compare to Eq. (5.5).

Case O is false: Again we will scale the jth false input's eigenvector by g_j; the total norm of the eigenvector at and above the nodes a_J is $\sum_{\text{false } j} |g_j|^2 = \langle g|g\rangle$. Our goal is to choose $|g\rangle$, $\langle g|g\rangle = 1$, to maximize $|b_O|^2$, where by Eq. (5.2c) at $\lambda = 0$, for each $j \in J$ either $0 = o_j b_O + c_j b_C + g_j \sqrt{\gamma_j}$ if j is false, or $0 = o_j b_O + c_j b_C$ if j is true.

- If there is at least one true input j', then that input fixes the relationship between b_C and b_O. We must have $b_C = -\frac{o_{j'}}{c_{j'}} b_O$, with $c_{j'} \neq 0$ necessarily. Therefore, for all false j,

$$b_j = g_j \sqrt{\gamma_j} = -o_j b_O - c_j b_C = -\frac{d_{jj'}}{c_{j'}} b_O \ .$$

So, $g_j = \frac{-1}{\sqrt{\gamma_j}} \frac{d_{jj'}}{c_{j'}} b_O$, and

$$\frac{1}{\gamma_O} = \frac{\sum_{\text{false } j} |g_j|^2 + |b_O|^2 + |b_C|^2}{|b_O|^2} = \frac{\sum_{\text{false } j} |d_{jj'}|^2/\gamma_j}{|c_{j'}|^2} + 1 + \frac{|o_{j'}|^2}{|c_{j'}|^2} \ .$$

Compare to Eq. (5.7).

- If all inputs j are false, then we want to maximize b_O^2 such that $\sum_j |g_j|^2 = 1$ and for all j, $0 = o_j b_O + c_j b_C + g_j \sqrt{\gamma_j}$. There is no true j to fix the value of b_C in terms of b_O, so let $\beta = b_C / b_O$. Then $g_j = \frac{-1}{\sqrt{\gamma_j}} (o_j + \beta c_j) b_O = -b_O \langle j | (|o(x)\rangle + \beta |c(x)\rangle)$. We want to choose β to maximize

$$\frac{b_O^2}{\langle g|g\rangle} = \frac{1}{\| |o(x)\rangle + \beta |c(x)\rangle \|^2} \ .$$

It is geometrically clear that the minimum is achieved for $\beta = -\langle c(x)|o(x)\rangle / \langle c(x)|c(x)\rangle$, so

$$\frac{1}{\gamma_O} = \frac{\langle g|g\rangle + |b_O|^2 + |b_C|^2}{|b_O|^2} = \langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle}\Big) |o(x)\rangle + 1 + |\beta|^2 \ .$$

Note that $|\beta| \leq \frac{1}{\sum_j |c_j|^2/\gamma_j} \sum_j \frac{1}{\gamma_j} |c_j^* o_j| \leq \sum_j |o_j|/|c_j|$ is $O(1)$. Compare to Eq. (5.6). $\square$

Remark 6.3. *The analysis behind Eqs. (6.3) and (6.4) generalizes immediately to $|C| > 1$ if we replace the projection $|c(x)\rangle\langle c(x)| / \langle c(x)|c(x)\rangle$ with the projection onto the span of the similarly scaled rows of A_{CJ}. Eq. (6.5) also generalizes, but in a more complicated manner.*

7 Unified statement of results for $\lambda = 0$ and small $\lambda > 0$

Except for a different fudge factor, the expressions in Lemma 6.2 are identical to those in Theorem 5.6, after the substititutions $\gamma_O \to 1/S_O$ and $\gamma_i \to 1/\tilde{S}_j$.[4] Let us therefore restate the two results in a unified form, with a convenient unified notation.

Definition 7.1 (Unified notation)**.** *Consider a span program P with $|C| \leq 1$. For inputs $i \in I$, let*

$$\sigma_i = \begin{cases} S_i & \text{if } \lambda > 0 \\ 1/\gamma_i & \text{if } \lambda = 0 \end{cases} \tag{7.1}$$

If σ_i depends on the true/false evaluation of i, then use whichever bound is larger.

[4]Should this be surprising? Perhaps, one expects the $\lambda = 0$ behavior to arise as the limiting behavior taking $\lambda \to 0^+$. (Even believing this, though, one would need to compute how quickly the behavior approached its limit.) However, this is not clear, because for example the $\lambda = 0$ eigenspace is often highly degenerate while eigenspaces for $\lambda \neq 0$ are generically nondegenerate.

Similarly, let σ_O be either S_O or $1/\gamma_O$. Recall that a grouped input j evaluates to true iff all inputs in I_j are true. For grouped inputs $j \in J$ and for a given input x, let

$$\tilde{\sigma}_j = \begin{cases} \max\{\sum_{i \in I_j} \sigma_i, 1\} & \text{if } j \text{ is true} \\ (\sum_{\text{false } i \in I_j} \sigma_i^{-1})^{-1} & \text{if } j \text{ is false} \end{cases} \tag{7.2}$$

(In the first case, we take the maximum with 1 to handle the case $I_j = \emptyset$.) A convenient special case is when all σ_i are equal, in which case $\tilde{\sigma}_j$ is either $\max\{|I_j|\sigma_i, 1\}$) *or $\sigma_i/\#\{\text{false } i \in I_j\}$, if j is true or false, respectively.*

For a vector $|v\rangle = \sum_j v_j |j\rangle$, let

$$|v(x)\rangle = \begin{cases} \sum_{\text{true } j} \frac{v_j}{\sqrt{\tilde{\sigma}_j}} |j\rangle & \text{if } f_P(x) = 1 \\ \sum_{\text{false } j} \sqrt{\tilde{\sigma}_j} v_j |j\rangle & \text{if } f_P(x) = 0 \end{cases} \tag{7.3}$$

Definition 7.2 (Asymptotic notation)**.** *Let $a \lesssim b$ mean $a \leq \text{constant} + b(1 + \text{constant}' \lambda \max_i S_i)$.*

Theorem 7.3. *Consider a span program P with at most one constraint, $|C| \leq 1$. Assume that $0 \leq \lambda \leq \epsilon/S_i$ for an arbitrarily small positive constant ϵ and for all $i \in I$. Assume also that $(\max_{i \in I} \sigma_i)/(\min_{i \in I} \sigma_i) = O(1)$. Recall from Eq. (7.3) the definitions of $c(x)$ and $o(x)$ that depend on whether $f_P(x)$ is true or false. Recall also that $d_{jj'} = o_j c_{j'} - o_{j'} c_j$.*

- *If the output $f_P(x)$ is true, then*

$$\begin{aligned} \sigma_O &\lesssim \Big[\langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle} \Big) |o(x)\rangle \Big]^{-1} \\ &= \begin{cases} \frac{\sum_{\text{true } j} |c_j|^2 / \tilde{\sigma}_j}{\sum_{\text{true } j, j'} |d_{jj'}|^2 / (\tilde{\sigma}_j \tilde{\sigma}_{j'})} & \text{if } d_{jj'} \neq 0 \text{ for some true } j, j' \\ \| |o(x)\rangle \|^{-2} = (\sum_{\text{true } j} |o_j|^2 / \tilde{\sigma}_j)^{-1} & \text{if } c_j = 0 \text{ for all true } j \end{cases} \end{aligned} \tag{7.4}$$

- *If $f_P(x)$ is false with all grouped inputs j false, then*

$$\sigma_O \lesssim \langle o(x)| \Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle} \Big) |o(x)\rangle \ . \tag{7.5}$$

  The expression on the right-hand side is $\big(\sum_{\text{false } j, j'} |d_{jj'}|^2 \tilde{\sigma}_j \tilde{\sigma}_{j'} \big) / \big(\sum_{\text{false } j} |c_j|^2 \tilde{\sigma}_j \big)$ provided $|c\rangle \neq 0$.

- *If the output is false, and there is at least one true input j', then*

$$\sigma_O \lesssim \frac{1}{|c_{j'}|^2} \sum_{\text{false } j} |d_{jj'}|^2 \tilde{\sigma}_j \ . \tag{7.6}$$

Proof. Now, at $\lambda = 0$, with γ_j defined as in Lemma 6.1, $1/\gamma_j \lesssim \tilde{\sigma}_j$. For $\lambda > 0$, with $\tilde{S}_j$ defined as in Lemma 5.4, $\tilde{S}_j \lesssim \tilde{\sigma}_j$.

The claimed Eqs. (7.4), (7.5), (7.6) are the same as the equations in Lemma 6.2 and Theorem 5.6, except with $\tilde{\sigma}_j$ in place of $1/\gamma_j$ or $\tilde{S}_j$, respectively. We need to argue that each term $1/\gamma_j$ or $\tilde{S}_j$ can be replaced with $\tilde{\sigma}_j$.

Write $\tilde{S}_j \leq \alpha_j + \tilde{\sigma}_j(1 + \beta_j \lambda \max_i S_i)$ for constants α_j and β_j. Then $\tilde{S}_j \leq \tilde{\sigma}_j(1 + \max_{j'} \frac{\alpha_{j'}}{\tilde{\sigma}_{j'}})(1 + \max_{j'} \beta_{j'} \lambda \max_i S_i)$. The factors $(1 + \max_{j'} \frac{\alpha_{j'}}{\tilde{\sigma}_{j'}})(1 + \max_{j'} \beta_{j'} \lambda \max_i S_i)$ can then be pulled out of all the expressions. Because we have assumed that $\sigma_i/\sigma_{i'} = O(1)$ for all i, i', also $\tilde{\sigma}_j/\tilde{\sigma}_{j'} = O(1)$ for all j, j', and the multiplicative factor of $(1 + \max_{j'} \frac{\alpha_{j'}}{\tilde{\sigma}_{j'}})$ becomes an additive constant.

The key point here is that adding a constant to an input only adds a constant to the output—it does not become a multiplicative constant. This can also be argued with a more detailed analysis without the assumption $\sigma_i/\sigma_{i'} = O(1)$, but this assumption suffices for our applications in this paper. □

Definition 7.4 (Span program complexity)**.** *Let P be a span program. For an input x, define the span program complexity of P on input x, $\operatorname{spc}(P,x)$, to be the minimum value b such that $\sigma_O \lesssim b \max_i S_i$, assuming that $0 \leq \lambda \max_i S_i$ is at most an arbitrarily small constant. Let the span program complexity of P be $\operatorname{spc}(P) = \max_x \operatorname{spc}(P)x$.*

8 NOT gate

The above analysis of span programs does not apply to the NOT gate, because the ability to complement inputs was assumed in Definition 4.1. 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.

Definition 8.1 (NOT gate)**.** *The NOT gate is implemented as two weight-one edges in a row. Call the three vertices b_i, $a_i = b_O$ and a_O. 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.*

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 $\gamma_O = \gamma_i$, because $a_i = b_O$ 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, $s_i \leq s_O \leq s_i(1 + 2\lambda^2 s_i)$ provided $\lambda^2 s_i \leq 1/2$.
- If the input is true, so $s_i = -1/(\lambda r_i)$, then $s_O = r_O/\lambda = s_i + 1$.

Remark 8.2. *Note that complementing a variable i barely changes s_i or γ_i. This may be useful if a span program takes inputs i and i' associated with a literal and its complement. Note also that we may assume without loss of generality that there are never two NOT gates in a row in the formula φ.*

9 Span programs for the gates in $\mathcal{S}$

In this section, we will plug specific span programs into Theorem 7.3, in order to prove:

Theorem 9.1. *Let $\mathcal{S}$ be the gate set of Definition 2.1. For $f \in \mathcal{S}$, let $A(f)$ be the nonnegative-weight adversary bound for f (Definition 2.3).*

Then for any gate $f \in \mathcal{S}$, there exists a span program P computing $f_P = f$, such that the span program complexity of P (Definition 7.4) is

$$\operatorname{spc}(P) = A(f) \ . \tag{9.1}$$

Remark 9.2. *The proof of Theorem 3.1 will imply that* $\operatorname{spc}(P) \geq A(f_P)$ *(by the adversary query lower bound Theorem 2.5), so these span programs are optimal.*

Proof of Theorem 9.1. We begin by analyzing the k-bit EQUAL gate (Claim 9.5), the MAJ$_3$ gate (Claim 9.7), 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 9.8). For each gate f, we present a span program P, with $|C| \leq 1$, that computes $f_P = f$. We then substitute into Theorem 7.3 to determine $\operatorname{spc}(P)$.

In the analyses of these gates, we will assume that the input S_I and γ_I^{-1} are all equal. If not, then replace each S_i with $\max_i S_i$, and replace each γ_i with $\min_i \gamma_i$. Since the expressions in Theorem 7.3 then scale linearly with S_i and $1/\gamma_i$, we may set $S_i = 1/\gamma_i = 1$ to compute the leading constant factor $\operatorname{spc}(P,x)$.

For all the remaining gates in $\mathcal{S}$, it suffices to analyze OR and PARITY gates on *unbalanced* inputs (Lemmas 9.10 and 9.11). 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, $A(b \oplus b') = A(b) + A(b')$, and $A(b \vee b') = \sqrt{A(b)^2 + A(b')^2}$ [BS04, HLŠ05]; we obtain matching upper bounds for span program complexity.

Remark 9.3. *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 programs are the minimum-size {AND, OR, NOT} formula expansions of f.*
2. *Next, compute for each input x the complexity* $\mathrm{spc}(P, x)$ *as a function of the variable weights of P.*
3. *Finally, optimize the nonzero weights of P in order to minimize* $\mathrm{spc}(P) = \max_x \mathrm{spc}(P, x)$. *For example, note that scaling $\| |o\rangle \|$ up helps the true cases in Theorem 7.3, and hurts the false cases. We therefore choose $\| |o\rangle \|$ to balance off the worst true case against the worst false case.*

   If the function has symmetries, we respect those symmetries during optimization. On the other hand, if two literals are not treated symmetrically by f, then we do not put those literals together in any grouped input X_j. For an example, see the proof of Claim 9.8, and also see Remark 5.9.

For ease of exposition, in some of the cases below we will directly present the optimal P.

EQUAL gate

Expanding the k-bit EQUAL function as $\mathrm{EQUAL}(x_1, \ldots, x_k) = (x_1 \wedge \ldots \wedge x_k) \vee (\overline{x_1} \wedge \ldots \wedge \overline{x_k})$ leads to a natural span program:

Definition 9.4. *Let P_{EQUAL} be the span program with $t = (1)$ and*

$$\begin{array}{cc} X_1 = \{x_1, x_2, \ldots, x_k\} & X_2 = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k}\} \\ v_1 = (w) & v_2 = (w) \end{array} \quad ,$$

where $w = (k-1)^{1/4}$ (this value comes from optimizing the calculation below).

The adversary bound for the k-bit EQUAL gate is $k/\sqrt{k-1}$.[5] We obtain a matching upper bound.

Claim 9.5. *The complexity of the k-bit* EQUAL *gate is* $\mathrm{spc}(P_{\mathrm{EQUAL}}) = k/\sqrt{k-1}$.

Proof. We substitute into Theorem 7.3. Since there are no constraints, $|c\rangle = 0$. Cases are distinguished according to the Hamming weight of the input x. Since an input x is equivalent to its bitwise complement, Hamming weight h is equivalent to weight $k - h$.

Case $x \in \{0^k, 1^k\}$: The first grouped input, $x_1 \wedge \ldots \wedge x_k$, is false, while the second grouped input, $\overline{x_1} \wedge \ldots \wedge \overline{x_k}$, is true. EQUAL$(x)$ is true. In Theorem 7.3, $|o(x)\rangle = (0, w/\sqrt{k})$, so by Eq. (7.4),

$$\mathrm{spc}(P_{\mathrm{EQUAL}}, x) = \| |o(x)\rangle \|^{-2} = k/w^2 \quad .$$

Case $|x| = h \notin \{0, k\}$: EQUAL(x) and both inputs are false. $|o(x)\rangle = (w/\sqrt{k-h}, w/\sqrt{h})$, so by Eq. (7.5),

$$\mathrm{spc}(P_{\mathrm{EQUAL}}, x) = \| |o(x)\rangle \|^2 = w^2 \big(\frac{1}{k-h} + \frac{1}{h} \big) \quad .$$

This complexity is highest for $h \in \{1, k-1\}$, for which it is $w^2 k/(k-1)$.

Set the weight $w = (k-1)^{1/4}$ to balance the two worst cases, both at $k/\sqrt{k-1}$. $\square$

[5]The optimal adversary matrix Γ comes from the $2 \times 2k$ matrix $\left(\begin{smallmatrix} 1 & 1 & \cdots & a & a & \cdots \\ a & a & \cdots & 1 & 1 & \cdots \end{smallmatrix}\right)$, where the rows correspond to inputs 0^k and 1^k, and the columns correspond to inputs of Hamming weight 1 then $k-1$, and $a = 1/(k-1)$.

Majority gate MAJ_3

Definition 9.6. *Let P_{MAJ_3} be the span program with $|o\rangle = \frac{1}{\sqrt{3}}(1,1,1)$, $|c\rangle = (1, \omega, \omega^2)$ with $\omega = e^{2\pi i/3}$, and $X_j = \{x_j\}$ for $j = 1, 2, 3$.*

As discussed in Example 4.2, P_{MAJ_3} computes the majority function, $f_{P_{\mathrm{MAJ}_3}} = \mathrm{MAJ}_3$. From Table 1, the adversary bound is $A(\mathrm{MAJ}_3) = 2$.[6]

Claim 9.7. *The complexity* $\mathrm{spc}(P_{\mathrm{MAJ}_3}) = 2 = A(\mathrm{MAJ}_3)$.

Proof. We substitute into Theorem 7.3. There are four inequivalent cases for the input $x = x_1x_2x_3$. For $j \neq j'$, $|d_{jj'}|^2 = 1$.

Case $x = 000$: $\mathrm{MAJ}_3(x) = 0$. $|o(x)\rangle = |o\rangle$, $|c(x)\rangle = |c\rangle$. Since $|o\rangle$ and $|c\rangle$ are orthogonal, by Eq. (7.5),

$$\mathrm{spc}(P_{\mathrm{MAJ}_3}, x) = \||o(x)\rangle\|^2 = 1 \ .$$

Case $x = 100$: $\mathrm{MAJ}_3(x) = 0$, but there is a true input. Therefore, by Eq. (7.6),

$$\mathrm{spc}(P_{\mathrm{MAJ}_3}, x) = \frac{1}{|c_1|^2} \sum_{j=2,3} |d_{j1}|^2 = 2 \ .$$

Case $x = 110$: $\mathrm{MAJ}_3(x) = 1$. $|o(x)\rangle = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, 0)$ and $|c(x)\rangle = (1, \omega, 0)$. By Eq. (7.4),

$$\mathrm{spc}(P_{\mathrm{MAJ}_3}, x) = \left[\||o(x)\rangle\|^2 - \frac{|\langle o(x)|c(x)\rangle|^2}{\||c(x)\rangle\|^2} \right]^{-1} = \frac{\sum_{j=1,2} |c_j|^2}{|d_{12}|^2} = 2 \ .$$

Case $x = 111$: $\mathrm{MAJ}_3(x) = 1$. $|o(x)\rangle = |o\rangle$, $|c(x)\rangle = |c\rangle$. Since $|o\rangle$ and $|c\rangle$ are orthogonal, by Eq. (7.4),

$$\mathrm{spc}(P_{\mathrm{MAJ}_3}, x) = \||o(x)\rangle\|^{-2} = 1 \ . \qquad \square$$

Three-bit gate $g(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$

Now consider $g(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$. From Table 1, the adversary bound is $A(g) = \sqrt{3 + \sqrt{3}}$.[7]

Claim 9.8. *Consider the span program P with $t = (1, 0)$ and*

$$\begin{array}{ccc} X_1 = \{x_1, x_2\} & X_2 = \{\overline{x_1}, \overline{x_2}\} & X_3 = \{x_3\} \\ v_1 = \left(\begin{smallmatrix} o_1 \\ c_1 \end{smallmatrix}\right) & v_2 = \left(\begin{smallmatrix} o_2 \\ 0 \end{smallmatrix}\right) & v_3 = \left(\begin{smallmatrix} 0 \\ c_3 \end{smallmatrix}\right) \end{array} \ ,$$

Then $f_P(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$ if $o_1, o_2, c_1, c_3 \neq 0$. Additionally,

$$\min_{o_1, o_2, c_1, c_3 > 0} \mathrm{spc}(P) = \sqrt{3 + \sqrt{3}} = A(f_P) \ .$$

Proof. We substitute into Theorem 7.3. There are six inequivalent cases for the input $x = x_1x_2x_3$: $010, 011, 110$ all evaluating to false, and $000, 001, 111$ all evaluating to true. Note in the calculations below that $d_{12} = -o_2c_1$, $d_{13} = o_1c_3$, $d_{23} = o_2c_3$, and since $|X_3| = 1$, $\tilde{\sigma}_3 = 1$ always.

[6]The optimal adversary matrix Γ comes from the matrix $\left(\begin{smallmatrix} 0&1&1 \\ 1&0&1 \\ 1&1&0 \end{smallmatrix}\right)$, where the rows correspond to inputs 100, 010, 001, and the columns correspond to inputs 011, 101, 110.

[7]The optimal adversary matrix Γ comes from the matrix $\left(\begin{smallmatrix} 1 & a & \sqrt{3} \\ 1 & a & \sqrt{3} \\ b & 0 & \sqrt{3} \end{smallmatrix}\right)$, where $a = \left(\frac{1}{2}(5 - \sqrt{13 - 6\sqrt{3}})\right)^{1/2}$, $b = \frac{1}{2}\left(-1 - \sqrt{3} + \sqrt{2(8 + \sqrt{3})}\right)$, and the rows correspond to inputs 011, 101, 110, the columns to inputs 000, 001, 111.

Case $x = 010$**:** All three grouped inputs, $j = 1$ $(x_1 \wedge x_2)$, $j = 2$ $(\overline{x_1} \wedge \overline{x_2})$ and $j = 3$ (x_3) evaluate to be false. We have $\tilde{\sigma}_1 = \tilde{\sigma}_2 = \tilde{\sigma}_3 = 1$, so $|o(x)\rangle = |o\rangle$ and $|c(x)\rangle = |c\rangle$, and, by Eq. (7.5),

$$\mathrm{spc}(P, x) = \langle o(x)|\Big(1 - \frac{|c(x)\rangle\langle c(x)|}{\langle c(x)|c(x)\rangle}\Big)|o(x)\rangle = \frac{d_{12}^2 + d_{13}^2 + d_{23}^2}{c_1^2 + c_3^2} \quad .$$

Case $x = 011$**:** Only input $j = 3$ is true, with $\tilde{\sigma}_1 = \tilde{\sigma}_2 = 1$, so $\mathrm{spc}(P, x) = \frac{1}{{c_3}^2}({d_{13}}^2 + {d_{23}}^2)$ (Eq. (7.6)).

Case $x = 110$**:** Only $j = 1$ is true, so $\tilde{\sigma}_2 = 1/2$ and $\mathrm{spc}(P, x) = \frac{1}{{c_1}^2}({d_{12}}^2/2 + {d_{13}}^2)$ (Eq. (7.6)).

Case $x = 001$**:** Inputs $j = 2$ and 3 are true, with $\tilde{\sigma}_2 = 2$, so $\mathrm{spc}(P, x) = \frac{c_3^2}{{d_{23}}^2/2} = 2/o_2^2$ (Eq. (7.4)).

Case $x = 111$**:** Inputs $j = 1$ and 3 are true, with $\tilde{\sigma}_1 = 2$, so $\mathrm{spc}(P, x) = \frac{c_1^2/2 + c_3^2}{d_{13}^2/2}$ (Eq. (7.4)).

Case $x = 000$**:** Only $j = 2$ is true, so $|o(x)\rangle = (0, o_2/\sqrt{2}, 0)$, $|c(x)\rangle = 0$, and $\mathrm{spc}(P, x) = 2/o_2^2$ (Eq. (7.4)).

Now $\mathrm{spc}(P) = \max_x \mathrm{spc}(P, x) = \max\{\frac{d_{12}^2 + d_{13}^2 + d_{23}^2}{c_1^2 + c_3^2}, \frac{{d_{13}}^2 + {d_{23}}^2}{c_3^2}, \frac{{d_{12}}^2/2 + {d_{13}}^2}{{c_1}^2}, \frac{2}{o_2^2}, \frac{c_1^2/2 + c_3^2}{d_{13}^2/2}, \frac{2}{o_2^2}\}$ is minimized by setting $o_1^2 = (1 + 1/\sqrt{3})^{1/2}, o_2^2 = (2 - 2/\sqrt{3})^{1/2}, c_1^2 = \sqrt{3} - 1, c_3 = 1$, so $\mathrm{spc}(P) = \sqrt{3 + \sqrt{3}}$. □

Remark 9.9. *The form of P in Claim 9.8 comes from the formula for $g(x)$. The second row of the v_J gives the* AND $(x_1 \wedge x_2) \wedge x_3$*, split into two groups because weights on the x_3 input should not be symmetrical weights on x_1 or x_2 inputs. The first row computes the* OR *with $\overline{x_1} \wedge \overline{x_2}$.*

PARITY gate, OR gate

Lemma 9.10. *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 complexities $B = \mathrm{spc}(P_b)$ and $B' = \mathrm{spc}(P_{b'})$. Then there exists a span program P for f with complexity $\mathrm{spc}(P) = B + B'$ if $f' = \mathrm{PARITY}$, or $\sqrt{B^2 + B'^2}$ if $f' = \mathrm{OR}$.

Proof. **PARITY:** For f' the PARITY gate, define a span program P_{PARITY} with $t = (1)$ and

$$\begin{array}{cc} X_1 = \{x_1, \overline{x_2}\} & X_2 = \{\overline{x_1}, x_2\} \\ v_1 = (1) & v_2 = (1) \end{array} \quad .$$

(This is the same span program as for the 2-bit EQUAL gate. In the notation of Theorem 7.3, $|c\rangle = 0$ and $|o\rangle = (1, 1)$.)

To define the span program P, feed as inputs to P_{PARITY} the outputs of P_b and $P_{b'}$. (Since P_{PARITY} works with x_1 and $\overline{x_1}$ it needs two copies of P_b, one of which has been negated with a NOT gate. Similarly, it uses two copies of $P_{b'}$.) Therefore, unlike for the previous gates, the input bounds entering P_{PARITY} differ: for the input x_1 or $\overline{x_1}$ we have

$$\sigma_1 \lesssim B \max_i S_i \quad ,$$

while for input x_2 or $\overline{x_2}$ we have

$$\sigma_2 \lesssim B' \max_i S_i \quad .$$

Here, the S_I are the ratio bounds for the inputs entering the span programs P_b and $P_{b'}$, and *not* the bounds entering P_{PARITY}. For computing the span program complexity, we may set all S_i to one.

Up to symmetry, there are only two cases to analyze: $x = 01$ or $x = 00$.

Case $x = 01$**:** The first grouped input is false, while the second grouped input is true. Let $\tilde{\sigma}_2 = B + B'$ (see Eq. (7.2)). Then $|o(x)\rangle = (0, 1/\sqrt{\tilde{\sigma}_2})$, and by Eq. (7.4),

$$\mathrm{spc}(P, x) = \frac{1}{\||o(x)\rangle\|^2} = B + B' \ .$$

A symmetrical argument holds for case $x = 10$.

Case $x = 00$**:** Neither grouped input is true, so $|o(x)\rangle = (\sqrt{\tilde{\sigma}_1}, \sqrt{\tilde{\sigma}_2})$, where $\tilde{\sigma}_1 = B'$ and $\tilde{\sigma}_2 = B$ (see Eq. (7.2)). Then by Eq. (7.5),

$$\mathrm{spc}(P, x) = \||o(x)\rangle\|^2 = B' + B \ .$$

A symmetrical argument holds for case $x = 11$.

Therefore, $\mathrm{spc}(P) = B + B'$, as claimed.

OR gate: For f' the OR gate, define a span program P_{OR} with $t = (1)$ and

$$\begin{array}{cc} X_1 = \{x_1\} & X_2 = \{x_2\} \\ v_1 = (\sqrt{B}/(B^2 + B'^2)^{1/4}) & v_2 = (\sqrt{B'}/(B^2 + B'^2)^{1/4}) \end{array} .$$

(In the notation of Theorem 7.3, $|c\rangle = 0$, $o_1 = \sqrt{B}/(B^2 + B'^2)^{1/4}$, and $o_2 = \sqrt{B'}/(B^2 + B'^2)^{1/4}$.) As for PARITY, feed as inputs to P_{OR} the outputs of P_b and $P_{b'}$ to define P (one copy of each). Again, we may take $\sigma_1 \lesssim B$ and $\sigma_2 \lesssim B'$. Since both grouped inputs are on a single literal, $\tilde{\sigma}_1 = B$ and $\tilde{\sigma}_2 = B'$ always (see Eq. (7.2)).

Case $x = 00$**:** Neither input is true. $|o(x)\rangle = (\sqrt{\tilde{\sigma}_1} o_1, \sqrt{\tilde{\sigma}_2} o_2)$, and by Eq. (7.5),

$$\mathrm{spc}(P, x) = \||o(x)\rangle\|^2 = \sqrt{B^2 + B'^2} \ .$$

Case $x = 01$**:** OR(01) is true, so $|o(x)\rangle = (0, o_2/\sqrt{\tilde{\sigma}_2})$, and by Eq. (7.4),

$$\mathrm{spc}(P, x) = \frac{1}{\||o(x)\rangle\|^2} = \sqrt{B^2 + B'^2} \ .$$

A symmetrical argument holds for case $x = 10$.

Case $x = 11$**:** OR(11) is true, so $|o(x)\rangle = (o_1/\sqrt{\tilde{\sigma}_1}, o_2/\sqrt{\tilde{\sigma}_2})$, and by Eq. (7.4),

$$\mathrm{spc}(P, x) = \frac{1}{\||o(x)\rangle\|^2} = \tfrac{1}{2}\sqrt{B^2 + B'^2} \ .$$

Therefore, $\mathrm{spc}(P) = \sqrt{B^2 + B'^2}$, as claimed. □

Arbitrary three-bit gates, and all gates in $\mathcal{S}$

Lemma 9.11. *For any gate $f \in \mathcal{S}' = \{$arbitrary two- or three- bit gates, $O(1)$-fan-in EQUAL gates$\}$ (Definition 2.1), there exists a span program P computing $f_P = f$, such that*

$$\mathrm{spc}(P) = A(f) \ . \tag{9.2}$$

Proof. So far, we have verified this claim for the EQUAL gate, and for six of the fourteen inequivalent binary functions on at most three bits, listed in Table 1: 0 and x_1 (both trivial), $x_1 \wedge x_2 = \neg(\overline{x_1} \vee \overline{x_2})$, $x_1 \oplus x_2$, $\mathrm{MAJ}_3(x_1, x_2, x_3)$, $\mathrm{EQUAL}(x_1, x_2, x_3)$, and $g(x) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$.

By applying Lemma 9.10, we can finish off all the other gates as well. For example, the function $\mathrm{MAJ}_3(x_1, x_2, x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3})$ has adversary bound $\sqrt{7} = \sqrt{A(\mathrm{MAJ}_3)^2 + A(\mathrm{AND}_3)^2}$. □

In fact, this completes the proof of Theorem 9.1, by applying Lemma 9.10 repeatedly to handle $O(1)$-size {AND, OR, NOT, PARITY} formulas on inputs possibly themselves gates from $\mathcal{S}'$. $\square$

Example 9.12. For example, Lemma 9.10 implies that any {AND, OR, NOT} formula of bounded size has a span program with complexity the square root of the sum of the squares of the input complexities. We conjecture that this holds even for formulas with size $\omega(1)$; see [Amb06b, ACR+07] for special cases.

Remark 9.13. *In the proof of Theorem 9.1, we used separate analyses for* EQUAL, MAJ_3 *and g because the upper bounds from Lemma 9.10 for these functions do not match the adversary lower bounds. E.g., from Table 1 the smallest {AND, OR, NOT} formula for* MAJ_3 *has five inputs. Lemma 9.10 therefore gives a span program P for* MAJ_3 *with complexity* $\mathrm{spc}(P) = \sqrt{5}$. *This does not match* $A(\mathrm{MAJ}_3) = 2$, *though.*

For EQUAL *and for g, the span programs of Definition 9.4 and Claim 9.8 are based on optimal-size {AND, OR, NOT} formulas. E.g., $g(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2})$. The span program complexity is less than the naive $\sqrt{5}$ for g (or $\sqrt{2k}$ for k-bit* EQUAL*) essentially because the worst-case inputs to the formula $(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5)$ do not arise when the fourth input x_4 is promised to be $\overline{x_1}$, and x_5 is promised to be $\overline{x_2}$. This allowed further optimization of the span program weights.*

For the MAJ_3 *gate, optimizing the span program associated to the minimum-size formula over inputs of the promised form also beats $\sqrt{5}$; it gives a span program with complexity $\sqrt{3+\sqrt{2}}$. However, this is still worse than the span program P_{MAJ_3} of Definition 9.6.*

10 Span program spectral analysis of the formula φ

Theorem 10.1. *Consider an adversary-balanced formula φ on the gateset $\mathcal{S}$. Let P be the span program computing $f_P = \varphi$, obtained by composing the span programs for the individual gates following the rules of Theorem 9.1 and Definition 8.1. For an input $x \in \{0,1\}^N$, recall the definition of the weighted graph $G_P(x)$ from Lemma 4.6; if $x_i = 1$, delete from G_P the edge to the ith input.*

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 $w = \epsilon_w / \sqrt{A(\varphi)}$ (instead of weight one), where $\epsilon_w > 0$ is a constant that will be determined in the proof. Then,

- *If $\varphi(x) = 1$, there exists a normalized eigenvalue-zero eigenstate 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 eigenstates supported on a_O or b_O for $|\lambda| \leq \epsilon / A(\varphi)$.*

Proof. The proof of Theorem 10.1 has two parts. First, we will prove by induction that $\sigma_g = O(A(\varphi_g))$. 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 expressions for $\tilde{S}_j$ in Lemma 5.4, and for γ_j in Lemma 6.1, 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 $|\lambda| \leq \epsilon / A(\varphi)$, 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 9.1, the output bound σ_g satisfies

$$\sigma_g \lesssim A(g) \max_i \sigma_{h_i} \ , \tag{10.1}$$

or equivalently

$$\sigma_g \leq \text{constant} + A(g)(\max_i \sigma_{h_i})(1 + \text{constant'} \cdot |\lambda| A(\varphi_g)) \tag{10.2}$$

for certain constants. Different kinds of gates give different constants in Eq. (10.2), but since the gate set is finite, all constants are uniformly $O(1)$.

Since $|\lambda| \leq \epsilon/A(\varphi)$, the recurrence Eq. (10.2) has solution

$$\sigma_g \leq O\Bigg(\max_{\chi} \prod_{h \in \chi} A(h) \Big(1 + \epsilon' \frac{A(\varphi_h)}{A(\varphi)} \Big) \Bigg) \ ,$$

where the maximum is over the choice of χ a non-self-intersecting path from g up to an input. Because φ is by assumption adversary balanced (Definition 2.8), $\prod_{h \in \chi} A(h) = A(\varphi_g)$ (Theorem 2.9). Also, $\prod_{h \in \chi}(1 + \epsilon' \frac{A(\varphi_h)}{A(\varphi)}) = O(1)$. Therefore, the solution satisfies

$$\sigma_g = O(A(\varphi_g)) \ . \tag{10.3}$$

Remark 10.2. *To explain Eq. (10.3), let us recall that the notation σ_g has meanings both for $\lambda = 0$ and for $\lambda \neq 0$ (Definition 7.1). Let (a_g, b_g) be the output edge for g. Then Eq. (10.3) implies that there exists a normalized vector, satisfying the eigenvector equations at b_g and all higher vertices for $\lambda = 0$, that has squared amplitude on either a_g (if $\varphi_g(x) = 1$) or b_g (if $\varphi_g(x) = 0$) at least $\gamma_g \geq 1/O(A(\varphi_g))$. (By Lemma 4.6, there does not exist any eigenvalue-zero eigenvector supported on b_g if $\varphi_g(x) = 1$, or on a_g if $\varphi_g(x) = 0$.) Eq. (10.3) also implies that for nonzero λ, the ratio bound S_g of Definition 5.1 is $O(\varphi_g)$.*

Final amplification step: Assume $\varphi(x) = 1$. Then by Eq. (10.3), there exists a normalized eigenvalue-zero eigenstate of the graph $G_P(x)$ with squared amplitude $|a_O|^2 \geq \gamma_O = 1/O(A(\varphi))$. Recall that $w = \epsilon_w/\sqrt{A(\varphi)}$ is the weight of the output edge (a_O, b_O) of P in $\tilde{G}_P(x)$, and let $\tilde{a}_O = w a_O$. The $\lambda = 0$ eigenvector equations for $\tilde{G}_P(x)$ are the same as those for $G_P(x)$, except with $\tilde{a}_O$ in place of a_O. Therefore, we may take $|\tilde{a}_O|^2 = 1/\Omega(A(\varphi))$, so for a normalized eigenvalue-zero eigenstate 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 Lemma 4.6, there does not exist any eigenvalue-($\lambda = 0$) eigenvector supported on a_O. Also $b_O = 0$ at $\lambda = 0$ by the constraint $\lambda a_O = w b_O$. For $\lambda \neq 0$, $|\lambda| \leq \epsilon/A(\varphi)$, Eq. (10.3) implies that in any eigenvalue-λ eigenstate for $\tilde{G}_P(x)$, either $\tilde{a}_O = b_O = 0$ or the ratio $|\tilde{a}_O/b_O| \leq |\lambda| \cdot O(A(\varphi))$, so

$$|a_O/b_O| \leq \text{constant} \cdot \frac{|\lambda|}{w} A(\varphi) \tag{10.4}$$

for some constant that does not depend on w. We have not yet used the eigenvector equation at a_O. This gives $\lambda a_O = w b_O$, so

$$a_O/b_O = w/\lambda \ . \tag{10.5}$$

Combining Eqs. (10.4) and (10.5), we get $w^2 \leq \text{constant} \cdot \lambda^2 A(\varphi) \leq \text{constant} \cdot \epsilon^2/A(\varphi)$. Substituting $w = \epsilon_w/\sqrt{A(\varphi)}$, 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 eigenstate supported on a_O or b_O. $\square$

11 Formula evaluation algorithm

In this section, we apply Theorem 10.1 and the Szegedy correspondence between discrete- and continuous-time quantum walks [Sze04] to design the optimal quantum algorithm $\mathcal{A}$ needed to prove Theorem 3.1. The approach is very similar to that used for the NAND formula evaluation algorithm of [CRŠZ07], with only a minor technical difference. The main idea is to construct a quantum walk $U_x = O_x U_{0^N}$ whose spectrum and eigenvectors correspond exactly to those of $A_{\tilde{G}_P(x)}$. Starting at the output node a_O, run phase estimation on U_x to distinguish $\varphi(x) = 0$ from $\varphi(x) = 1$ by using Theorem 10.1. Essentially, since the phase-estimation precision needed to avoid false positives is $1/O(A(\varphi))$, the query complexity will be $O(A(\varphi))$.

This approach is slightly indirect. To motivate it, we begin by briefly considering in Section 11.1 a more direct algorithm $\mathcal{A}'$, that runs phase estimation directly on $\exp(iA_{\tilde{G}_P(x)})$. $\mathcal{A}'$ is analogous to the algorithm described by Cleve et al. [CCJY07] soon after the original NAND formula evaluation paper [FGG07]. Algorithm $\mathcal{A}'$ is nearly optimal, but not quite. The operator $\exp(iA_{\tilde{G}_P(x)})$ is a continuous-time quantum walk, and the overhead can be thought of as coming from simulating continuous-time quantum dynamics with a discrete computational model, in particular with discrete oracle queries. To avoid this overhead, we would like to work with a discrete-time quantum walk, which we do to prove Theorem 3.1 in Section 11.2.

The approach in Section 11.1 is optional motivation, and the reader may choose to skip directly to Section 11.2.

11.1 Intuition: Continuous-time quantum walk algorithm

Theorem 10.1 immediately suggests the basic form of a quantum algorithm for evaluating $\varphi(x)$:

```
Algorithm $\mathcal{A}'$: Input $x \in \{0,1\}^N$, Output true/false.
    1. Prepare an initial state on the output node, $|a_O\rangle$.
    2. Run phase estimation, with precision $\delta_p \geq \epsilon/A(\varphi)$ and small enough constant error
       rate $\delta_e$, on the unitary $V = \exp(iA_{\tilde{G}_P(x)}/\|A_{\tilde{G}_P(0^N)}\|)$.
    3. Output true if and only if the phase estimation output is $\lambda = 0$.
```

The idea of the second step is to "measure the Hamiltonian $A_{\tilde{G}_P(x)}$." In this step, we have normalized $A_{\tilde{G}_P(x)}$ by $\|A_{\tilde{G}_P(0^N)}\| \geq \|A_{\tilde{G}_P(x)}\|$ instead of by $\|A_{\tilde{G}_P(x)}\|$, in order to minimize dependence on the input x. This norm is $O(1)$ since the graph $\tilde{G}_P(0^N)$ has vertex degrees and edge weights all $O(1)$.

The above procedure evaluates $\varphi(x)$ correctly, with a constant gap between its completeness and soundness:

- Theorem 10.1 implies that if the formula evaluates to true, then $A_{\tilde{G}_P(x)}$ has an eigenvalue-zero eigenstate with squared support $|a_O|^2 = \Omega(1)$ on a_O. Therefore, the phase estimation outcome is $\lambda = 0$ with probability at least $|a_O|^2 - \delta_e = \Omega(1)$ (the completeness parameter).
- On the other hand, if the formula evaluates to false, then Theorem 10.1 implies that $A_{\tilde{G}_P(x)}$ has no eigenvalue-λ eigenstates supported on a_O with $|\lambda| < \delta_p$. Therefore, the measured outcome will be $\lambda = 0$ only if there is an error in the phase estimation. By choosing δ_e a small enough constant, the soundness error δ_e will be bounded away from the completeness parameter.

Therefore, the above procedure is correct. Its efficiency also seems promising. Phase estimation of V with precision δ_p and error rate δ_e requires $O(1/(\delta_p\delta_e))$ calls to V [CEMM98]. Therefore, the second step requires only $O(A(\varphi))$ calls to V.

However, to complete the description of the algorithm $\mathcal{A}'$, we need to explain how to implement V. This is important because $A_{\tilde{G}_P(x)}$ depends on the input x. Therefore, implementing V requires querying the x. If each call to V requires many queries to the input oracle O_x (Definition 2.4), then $\mathcal{A}'$'s overall query efficiency will be poor.

Note now that only the input edges of $G_P(x)$ depend on the input x. Therefore, $A_{\tilde{G}_P(x)}$ can be split up into two terms: (input edges) + (all other edges). The first term can be exponentiated with only two queries to the input oracle O_x, while exponentiating the second term requires no input queries. The two terms do not commute, but the exponential of their sum can still be computed to sufficient precision by using a Lie product decomposition (these are more quantitative versions of identities like $e^{A+B} = \lim_{n\to\infty}(e^{A/n}e^{B/n})^n$). For more details, see [CCJY07].

Unfortunately, implementing the exponential of $A_{\tilde{G}_P(x)}$ will require $\omega(1)$ input queries. By using higher-order Lie product formulas, the overhead can be reduced to $\exp(O(\sqrt{\log|x|}))$, which is $N^{o(1)}$. However, this is still a super-constant overhead, so it appears that this approach cannot yield an optimal formula evaluation algorithm—the best we can hope for is $O(A(\varphi)) \cdot N^{o(1)}$ queries.

11.2 Proof of Theorem 3.1: Discrete-time quantum walk algorithm

Therefore, we turn to the approach used in the NAND formula evaluation algorithm of [CRŠZ07]. Instead of running phase estimation on the exponential of $A_{\tilde{G}_P(x)}$, we construct a discrete-time, or "coined," quantum walk $U_x = O_x U_{0^N}$ that has spectrum and eigenvectors corresponding in a precise way to those of $A_{\tilde{G}_P(x)}$. Then we run phase estimation on U_x. Each call to U_x requires exactly one oracle query, so there is no query overhead.

11.2.1 Construction of the coined quantum walk U_x

The first step in constructing U_x is to decompose $\tilde{A}_{G_P(0^N)}$ into (constant)$\cdot \Delta^\dagger \circ \Delta$, where Δ is a square matrix with row norms one, and $\circ$ denotes the entrywise matrix product. We follow [CRŠZ07]. One minor technical difference, though, is that for us, $A_{\tilde{G}_P(x)}$ is a Hermitian matrix with possibly complex entries. In [CRŠZ07], the analogous weighted adjacency matrix, for the NAND formula φ, is a real symmetric matrix. Therefore, we need to slightly modify the construction of P to obtain the correct phases for the entries of $\tilde{A}_{G_P(0^N)}$.

Definition 11.1. *For notational convenience, let $A = A_{\tilde{G}_P(0^N)}$ be the weighted adjacency matrix for $\tilde{G}_P(0^N)$. (Recall from Theorem 10.1 that $\tilde{G}_P(0^N)$ is the same as G_P except with the edge weight on the output edge (a_O, b_O) reduced.) $A = \sum_{v,w} A_{v,w} |v\rangle\langle w|$ is a Hermitian matrix.*

$\tilde{G}_P(0^N)$ is a bipartite graph, so we may color each vertex red or black, such that every edge is between one red vertex and one black vertex.

Claim 11.2. *Let $A' = \sum_{v,w} |A_{v,w}| |v\rangle\langle w|$ be the entrywise absolute value of A. A' is a real symmetric matrix. Let $\|A'\|$ be the largest magnitude eigenvalue of A'. Let $|\delta\rangle$ be the principle eigenvector of A', with $\langle v|\delta\rangle = \delta_v > 0$ for every v, and let*

$$\Delta = \frac{1}{\sqrt{\|A'\|}} \sum_{\substack{\text{black } v \\ \text{red } w}} \left(\left(\sqrt{A_{v,w}}\right)^* \sqrt{\frac{\delta_w}{\delta_v}} |v\rangle\langle w| + \sqrt{A_{v,w}} \sqrt{\frac{\delta_v}{\delta_w}} |w\rangle\langle v| \right) \quad . \tag{11.1}$$

Then Δ has all row norms one, and $A = \|A'\| \cdot \Delta^\dagger \circ \Delta$.

Proof. Since A' has nonnegative entries, the principal eigenvector $|\delta\rangle$ is also nonnegative. Since $\tilde{G}_P(0^N)$ is a connected graph, $\delta_v > 0$ for every v. Hence Δ is well defined up to choice of sign of the square root, which doesn't matter.

By construction, for all v and w, $\Delta^*_{v,w} \Delta_{w,v} = A_{v,w}/\|A'\|$, i.e., $\|A'\| \cdot \Delta^\dagger \circ \Delta = A$. Furthermore, the squared norm of the v-th row of Δ is $\sum_w |P_{v,w}|^2 = \frac{1}{\delta_v} \sum_w A'_{v,w} \delta_w = \frac{(A'\delta)_v}{\delta_v} = \|A'\|$. □

Remark 11.3. *In defining Δ, we have evenly divided the complex phases of entries of A between red-black and black-red terms. However, any division of the phases would have worked. For example, Claim 11.2 would also hold with Eq. (11.1) replaced by $\Delta = \frac{1}{\sqrt{\|A'\|}} \sum_{\substack{\text{black } v \\ \text{red } w}} \left(\frac{A^*_{v,w}}{\sqrt{|A_{v,w}|}} \sqrt{\frac{\delta_w}{\delta_v}} |v\rangle\langle w| + \sqrt{|A_{v,w}| \frac{\delta_v}{\delta_w}} |w\rangle\langle v| \right)$.*

We can now apply Szegedy's correspondence theorem [Sze04] to relate the spectrum of $\tilde{A}_{G_P}$ to that of a discrete-time coined quantum walk unitary.

Theorem 11.4 ([Sze04])**.** *Let $\{|v\rangle : v \in V\}$ be an orthonormal basis for $\mathcal{H}_V$. For each $v \in V$, let $|\tilde{v}\rangle = |v\rangle \otimes \sum_{w \in V} \delta_{vw} |w\rangle \in \mathcal{H}_V \otimes \mathcal{H}_V$, where $\langle \tilde{v}|\tilde{v}\rangle = \sum_w |\delta_{vw}|^2 = 1$. Let $T = \sum_v |\tilde{v}\rangle\langle v|$ and $\Pi = TT^\dagger = \sum_v |\tilde{v}\rangle\langle \tilde{v}|$ be the projection onto the span of the $|\tilde{v}\rangle$s. Let $S = \sum_{v,w} |v,w\rangle\langle w,v|$, a swap.*

Let $U = (2\Pi - 1)S$, a swap followed by reflection about the span of the $|\tilde{v}\rangle$s. Let $M = T^\dagger S T = \sum_{v,w} \delta^*_{vw} \delta_{wv} |v\rangle\langle w|$.

Then the spectral decomposition of U corresponds to that of M as follows: Take $\{|\lambda_\alpha\rangle\}$ a complete set of orthonormal eigenvectors of the Hermitian matrix M with respective eigenvalues λ_α. Let $R_\alpha = \operatorname{span}\{T|\lambda_\alpha\rangle, ST|\lambda_\alpha\rangle\}$. Then $R_a \perp R_{\alpha'}$ for $\alpha \neq \alpha'$; let $R = \oplus_\alpha R_\alpha$. U fixes the spaces R_a and is $-S$ on $R^\perp$.

The eigenvalues and eigenvectors of U within R_a are given by $\beta_{\alpha,\pm} = -\lambda_\alpha \pm i\sqrt{1-\lambda_\alpha^2}$ and $(1+\beta_{\alpha,\pm}S)T|\lambda_\alpha\rangle$, respectively.

(A proof of Theorem 11.4 in the above form is given in [CRŠZ07].)

Remark 11.5 (Coined quantum walks)**.** *The operator $U = (2\Pi - 1)S$ in Theorem 11.4 is known as a "coined quantum walk." S is known as the "step operator," and the reflection $(2\Pi - 1)$ is the "coin-flip operator." On the space R, $(2\Pi - 1)$ decomposes as $\sum_v |v\rangle\langle v| \otimes$ (reflection about $\sum_w \delta_{vw}|w\rangle$).*

In a classical random walk on a graph, a coin is flipped between each step to decide which adjacent vertex to step to next. In a coined quantum walk, on the other hand, the coin is maintained as part of the coherent quantum state, and is reflected between steps (also known as "diffusion").

Remark 11.6. *Theorem 11.4 can be viewed as giving a correspondence between coined quantum walks and classical random walks; in the special case that each $\delta_{vw} = \delta_{wv} \geq 0$, M is a classical random walk transition matrix. For general δ_{vw}, Theorem 11.4 can be viewed a correspondence between coined quantum walks and continuous-time quantum walks. We use the theorem in the latter sense.*

Lemma 11.7. *For Δ defined by Eq. (11.1) and with $\delta_{vw} = \Delta_{v,w}$, let U_{0^N} be the coined quantum walk operator $U_{0^N} = iU = i(2\Pi - 1)S$ in the notation of Theorem 11.4. U_{0^N} acts on $\mathcal{H}_V \otimes \mathcal{H}_V$, where V is the vertex set of G_P. For $x \in \{0,1\}^N$, let $U_x = (O_x \otimes I)U_{0^N}$, where O_x acts on $\mathcal{H}_V$ by applying a phase $(-1)^{x_i}$ to input vertex b_i and otherwise does nothing. Then,*

- *If $\varphi(x) = 1$, there exist eigenvalue 1 and eigenvalue -1 normalized eigenstates of U_x each with $\Omega(1)$ support on $|a_O, b_O\rangle$.*
- *If $\varphi(x) = 0$, then U_x does not have any eigenstates supported on $|a_O, b_O\rangle$ with eigenvalues $\pm e^{i\lambda}$ for $|\lambda| \leq \mathrm{constant}/A(\varphi)$.*

Proof. Note that for an input vertex b_i on a span program input edge (a_i, b_i), the b_ith row of Δ is $\langle b_i|\Delta = \langle a_i|$. Define $\Delta(x)$ as follows: If $x_i = 1$, then let $\langle b_i|\Delta(x) = \langle b_i|$, and let the other rows of $\Delta(x)$ be the same as those of $\Delta = \Delta(0^N)$.

In the notation of Theorem 11.4 with each δ_{vw} set to the (v,w) entry of $\Delta(x)$, the vectors $|\tilde{v}\rangle$ do not depend on x if $v \notin \{b_i\}_{i \in I}$, whereas

$$|\tilde{b}_i\rangle = \begin{cases} |b_i, a_i\rangle & \text{if } x_i = 0 \\ |b_i, b_i\rangle & \text{if } x_i = 1 \end{cases}$$

Therefore, in $M = \Delta(x)^\dagger \circ \Delta(x)$, entries (a_i, b_i) and (b_i, a_i) are zeroed out when $x_i = 1$, while other entries are unchanged: so $M = \frac{1}{\|A'\|} A_{\tilde{G}_P(x)}$. Also, on R, $U = (2\Pi - 1)S$ is the same as U_x. So Theorem 11.4 implies that the spectrum of $U_x = (O_x \otimes I)U_{0^N}$ corresponds exactly to that of $A_{\tilde{G}_P(x)}/\|A'\|$. If the eigenvalues of $A_{\tilde{G}_P(x)}/\|A'\|$ are $\{\lambda_\alpha\}$, then the eigenvalues of U_x are given by $\pm\sqrt{1-\lambda_\alpha^2} + i\lambda_\alpha$, i.e., $e^{i \arcsin \lambda_\alpha}$ and $-e^{-i \arcsin \lambda_\alpha}$.

In case $\varphi(x) = 1$, Theorem 10.1 promises that $A_{\tilde{G}_P(x)}$ has an eigenvalue-zero eigenstate with $\Omega(1)$ support on a_O. Denote this eigenstate by $|\lambda_\alpha = 0\rangle$. By Theorem 11.4, $(1 \pm iS)T|\lambda_\alpha = 0\rangle$ are eigenstates of U_x with eigenvalues ± 1. Since $T|a_O\rangle = |a_O, b_O\rangle$, the eigenvectors $(1 \pm iS)T|\lambda_\alpha = 0\rangle$ each have $\Omega(1)$ support on $|a_O, b_O\rangle$. Moreover, this remains true even after renormalizing: T is an isometry, while the swap S is unitary, so $\|(1 \pm iS)T|\lambda_\alpha = 0\rangle\| \leq 2$.

The claim also follows for the case $\varphi(x) = 0$ by Theorems 10.1 and 11.4. Every eigenstate of U with support on $|a_O, b_O\rangle = T|a_O\rangle$ must be of the form $(1 + \beta_{\alpha,\pm}S)T|\lambda_\alpha\rangle = (1 + (-\lambda_\alpha \pm i\sqrt{1-\lambda_\alpha^2})S)T|\lambda_\alpha\rangle$. The terms which can overlap $T|a_O\rangle$ are either $\langle a_O|\lambda_\alpha\rangle$ (via T) or $\langle b_O|\lambda_\alpha\rangle$ (via ST). But by Theorem 10.1, both coefficients must be zero. Note that $\|A'\| = O(1)$ since the graph $\tilde{G}_P(0^N)$ has vertex degrees and edge weights all $O(1)$. Therefore, the spectral gap from zero of $A_{\tilde{G}_P(x)}/\|A'\|$ is only a constant factor worse than that of $A_{\tilde{G}_P(x)}$. $\square$

11.3 Algorithm $\mathcal{A}$, correctness, and query and time complexity

```
Algorithm $\mathcal{A}$: Input $x \in \{0,1\}^N$, Output true/false.
    1. Prepare an initial state on the output edge $|a_O, b_O\rangle$.
    2. Run phase estimation on $U_x$, with precision $\delta_p \geq$ constant/$A(\varphi)$ and small enough
       constant error rate $\delta_e$.
    3. Output true if the measured phase is 0 or $\pi$. Otherwise output false.
```

Correctness: Lemma 11.7 implies that $\mathcal{A}$ is both complete and sound:

- If $\varphi(x) = 1$, then $U(x)$ has eigenvalue-(± 1) eigenstates each with $\Omega(1)$ squared support on $|a_O, b_O\rangle$. The completeness parameter is at least this squared support minus the phase estimation error rate δ_e. For small enough constant δ_e, the completeness is $\Omega(1)$.
- If $\varphi(x) = 0$, then since the precision parameter δ_p is smaller than the promised gap away from ± 1 in Lemma 11.7, phase estimation will output 0 or π only if there is an error. By choosing the error rate δ_e a small enough constant, the soundness error δ_e will be bounded away from the completeness parameter.

Therefore, algorithm $\mathcal{A}$ is correct. The constant gap between its completeness and soundness parameters can be amplified as usual.

Query and time complexity: Phase estimation of U_x with precision δ_p and error rate δ_e requires $O(1/(\delta_p \delta_e))$ calls to $U_x = (O_x \otimes I)U_{0^N}$ [CEMM98]. Therefore, $\mathcal{A}$ makes $O(A(\varphi))$ queries to the input oracle O_x.

The time-efficiency claim of Theorem 3.1 is slightly more complicated. Here, we need to allow a preprocessing phase in which the algorithm can compute $A_{\tilde{G}_P(0^N)}$ and in particular (approximations to) the coin diffusion operators in U_{0^N}. This preprocessing depending on φ, but not x, takes poly(N) time. The algorithm then needs coherent access to the precomputed information in order to apply efficiently the coin diffusion operators. For further details, see [CRŠZ07].

This completes the proof of Theorem 3.1. □

12 Extensions

We briefly discuss some directions in which our result, Theorem 3.1, can be extended.

12.1 Four-bit gates

Up to symmetries, there are 208 inequivalent binary functions that depend on exactly four input bits $x_1, \ldots, x_4$. We have considered so far only a minority of these functions. Results for all the functions we have considered are listed at the webpage [RŠ07]. To summarize,

- Thirty functions can be written as a PARITY or OR of two subformulas on disjoint inputs. These functions are included in the gate set $\mathcal{S}$, with an analysis by Lemma 9.10.
- For 13 additional functions, we have found a span program matching the adversary lower bound. The arguments for these functions are similar to the analyses in Section 9, although most use span programs with $|C| > 1$ constraints. These functions could be added to the gate set $\mathcal{S}$ without breaking Theorem 3.1.
- For 20 of the remaining functions, we have found a span program with complexity beating the square-root of the minimum {AND, OR, NOT} formula size, but not matching the adversary lower bound.

See the open problems in Section 13 for further discussion. 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.

12.2 Eliminating the preprocessing

In many cases for φ—perhaps even all cases?—the preprocessing step of algorithm $\mathcal{A}$ can be eliminated. Because φ is an adversary-balanced formula on a known gateset, a decomposition through Theorem 11.4 can be computed separately for each gate of $\mathcal{S}$ and then put together at runtime. This decomposition is *not* the decomposition of Claim 11.2, which involved global properties of φ like $\|A'\|$. For an example, see the exactly balanced NAND tree algorithm in [CRŠZ07].

The decompositions can be combined because all the weights of gate input/output edges are one. This is quite different from the case of unbalanced NAND trees considered by [CRŠZ07], in which the weight of an input edge depends on the subformula entering it.

12.3 Arbitrary {AND, OR, NOT, PARITY} formulas

Some of the conditions on the gates in $\mathcal{S}$ (Definition 2.1) can be loosened. For example, $\mathcal{S}$ includes as single gates $O(1)$-size {AND, OR, NOT, PARITY} formulas on inputs that are themselves possibly elements of $\mathcal{S}'$. Let f be such a gate, $f = g \circ (h_1, \ldots, h_k)$ with g an {AND, OR, NOT, PARITY} formula of size $O(1)$, and each h_i either the identity or a gate from $\mathcal{S}'$. We have assumed that all the inputs to f have equal adversary bounds. However, the proof we have given works equally well only provided that the adversary bounds entering each h_i are equal. Also, the adversary bounds entering h_i and entering $h_{i'}$ should differ by at most a constant factor (although as remarked in Theorem 7.3, this assumption can be eliminated).

We believe that the assumption that g be of size $O(1)$ can also be significantly weakened. A stronger analysis like that of [CRŠZ07] for "approximately balanced" {AND, OR, NOT} formulas can presumably also be applied with PARITY gates. We have avoided this analysis to simplify the proofs, and to focus on the main novelty of this paper, the extended gate sets.

For {AND, OR, NOT, PARITY} formulas that are not "approximately balanced," rebalancing will typically be required. We have not investigated how the formula rebalancing procedure of Bshouty, Cleve and Eberly [BCE91, BB94] affects the formula's adversary bound. In [CRŠZ07], it sufficed to consider the effect on the formula size, because the adversary bound for any {AND, OR, NOT} formula on N inputs is always $\sqrt{N}$.

13 Open problems

We have begun the development of a new framework for quantum algorithms based on span programs. There are many open problems in this area. All of the open problems from [CRŠZ07] remain unresolved. We would like to raise several other questions.

1. The technical analysis can be improved. Theorem 5.6 and Lemma 6.2 hold in the case that there is at most one constraint, $|C| \leq 1$. Lemma 6.2 does generalize to $|C| > 1$, but we only have Eq. (5.3) in the $\lambda \neq 0$ analysis. Analysis for $|C| > 1$ can still be run on a case-by-case basis, but general statements would be preferable. See the conjectures in Remark 5.9.

2. It is natural to try to extend the algorithm to work optimally for a larger gate set. Needless to say, there are many candidates. Optimal span programs for most four-bit functions are open—see the list of functions tabulated at [RŠ07]. One function of particular interest, and not on that list, is the six-bit Kushilevitz function [HLŠ07, Amb06a]. An interesting class of functions are the Hamming-weight threshold functions $\text{Threshold}_{h,k} : \{0,1\}^k \to \{0,1\}$ defined by

$$\text{Threshold}_{h,k}(x) = \begin{cases} 1 & \text{if } |x| \geq h \\ 0 & \text{if } |x| < h \end{cases}$$

There are obvious span programs for threshold functions, but of them we only have *optimal* span programs for the special cases of OR, AND and MAJ_3.

3. We have focused on the adversary bound with nonnegative weights $A(f)$. In fact, Eq. (2.2) still provides a lower bound on the quantum query complexity even when one removes the restriction that the entries of Γ be nonnegative [HLŠ07]. This more general adversary bound, $A^{\pm}(f)$ is clearly at least $A(f)$. For every three-bit function f, no advantage is gained by allowing negative weights: $A^{\pm}(f) = A(f)$. For most four-bit functions f, though, $A(f) < A^{\pm}(f)$, so this improvement is likely generic. However, for no function f such that $A(f) < A^{\pm}(f)$ do we have an upper bound that matches $A^{\pm}(f)$. (There are many fewer functions with $A(f) = A^{\pm}(f)$ for which we do not have an upper bound.) Might the $A^{\pm}$ bound be loose for many of the functions for which $A^{\pm}(f) > A(f)$? Theorem 2.9 is not known to hold for $A^{\pm}$; under the conditions of the theorem, only $A^{\pm}(f) \geq A^{\pm}(g)A^{\pm}(h_1)$ is known. Could the inequality be sometimes strict?

   Remark 13.1. *It seems there might be a connection between the nonnegative-weight adversary lower bound and the span program upper bound: in the cases we have checked, the span program's "hard" inputs, i.e., those x for which $\mathrm{spc}(P,x) = \mathrm{spc}(P)$, are also the "hard" inputs for the adversary bound, i.e., those x for which Γ has nonzero entries in a row or column.*

4. 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 9.10. (In the lemma, we assumed that the highest input adversary bound was at most a constant factor times the lowest input adversary bound, but this assumption can be removed; see Theorem 7.3.) For the MAJ_3 gate, we have an optimal span program for the case in which only two of the inputs are balanced:

   Lemma 13.2. *Consider $f(x,x',x'') = \mathrm{MAJ}_3(b(x), b'(x'), b''(x''))$ with b, b', b'' functions on $O(1)$ bits computed by span programs $P_b, P_{b'}, P_{b''}$ with complexities $B = \mathrm{spc}(P_b) = \mathrm{spc}(P_{b'})$ and $B'' = \mathrm{spc}(P_{b''})$. Let $\beta = B''/B$. Then there exists a span program P for f with complexity $\mathrm{spc}(P) = \frac{1}{2}\big(\sqrt{8+\beta^2}+\beta\big)B = A(f)$. P has target vector $t = (1,0)$ and*[8]

$$\begin{array}{lll} X_1 = \{x_1\} & X_2 = \{x_2\} & X_3 = \{x_3\} \\ v_1 = \big(w, i\big) & v_2 = \big(w, -i\big) & v_3 = \big((\frac{1}{2} + \beta w^2)^{1/2}, 2w\big) \end{array}$$

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

   For example, the gates $\mathrm{MAJ}_3(x_1, x_2, x_3 \wedge x_4)$ and $\mathrm{MAJ}_3(x_1, x_2, x_3 \oplus x_4)$ are among the four-bit functions of Section 12.1 for which we have an optimal span program. However, we do not have an understanding of MAJ_3 when all three input complexities differ. In this case, the formula for the adversary lower bound is substantially more complicated, and we do not have a matching span program.

   For other gates, with the exception of PARITY and OR, we know similarly little. For a highly unbalanced formula with large depth, there is the further problem of whether the formula can be rebalanced without increasing its adversary lower bound too much.

5. Having run the algorithm to evaluate the formula, one is left with an eigenvalue-zero eigenstate. In Grover's search algorithm, which evaluates the fan-in N OR gate, the eigenstate gives the index of a 1 bit, a "witness," if $\mathrm{OR}(x)$ is true. For more general formulas, is the eigenvalue-zero eigenstate also useful, perhaps for extracting witness information?

6. In this paper, we have only plugged together bounded-size span programs evaluating $O(1)$-size functions. An intriguing question is, do there exist interesting quantum algorithms based directly on asymptotically large span programs? Some candidate problems may be found in [BGW99, BGP96], although note that we care about span programs over $\mathbf{C}$ that need not be monotone.

[8] In the balanced case $\mathrm{spc}(P_b) = \mathrm{spc}(P_{b'}) = \mathrm{spc}(P_{b''})$, this span program differs from the that of Claim 9.7, but the choice of phases $0, \pm\pi/2$ is more convenient to work with than $0, \pm 2\pi/3$.

14 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.

References

[ABO99] Eric Allender, Robert Beals, and Mitsunori Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8:99–126, 1999. Preliminary version in *Proc. 28th ACM STOC*, 1996.

[ACR$^+$07] Andris Ambainis, Andrew M. Childs, Ben W. Reichardt, Robert Špalek, and Shengyu Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In *Proc. 48th IEEE FOCS*, 2007.

[Amb06a] Andris Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Syst. Sci.*, 72(2):220–238, 2006. Preliminary version in *Proc. 44th IEEE FOCS*, 2003.

[Amb06b] Andris Ambainis. Quantum search with variable times. arXiv:quant-ph/0609168, 2006.

[Amb07] Andris Ambainis. A nearly optimal discrete query quantum algorithm for evaluating NAND formulas. arXiv:0704.3628 [quant-ph], 2007.

[BB94] Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for Boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

[BCE91] Nader H. Bshouty, Richard Cleve, and Wayne Eberly. Size-depth tradeoffs for algebraic formulae. In *Proc. 32nd IEEE FOCS*, pages 334–341, 1991.

[BGP96] Amos Beimel, Anna Gál, and Mike Paterson. Lower bounds for monotone span programs. *Computational Complexity*, 6:29–45, 1996. Preliminary version in *Proc. 36th IEEE FOCS*, 1995.

[BGW99] László Babai, Anna Gál, and Avi Wigderson. Superpolynomial lower bounds for monotone span programs. *Combinatorica*, 19(3):301–319, 1999. Preliminary version in *Proc. 28th ACM STOC*, 1996.

[BS04] Howard Barnum and Michael Saks. A lower bound on the quantum query complexity of read-once functions. *J. Comput. Syst. Sci.*, 69(2):244–258, 2004.

[BSS03] Howard Barnum, Michael Saks, and Mario Szegedy. Quantum decision trees and semidefinite programming. In *Proc. 18th IEEE Complexity*, pages 179–193, 2003.

[CCJY07] Andrew M. Childs, Richard Cleve, Stephen P. Jordan, and David Yeung. Discrete-query quantum algorithm for NAND trees. arXiv:quant-ph/0702160, 2007.

[CEMM98] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proc. R. Soc. London A*, 454(1969):339–354, 1998.

[CF02] Ronald Cramer and Serge Fehr. Optimal black-box secret sharing over arbitrary Abelian groups. In *Proc. CRYPTO 2002*, LNCS vol. 2442, pages 272–287. Springer-Verlag, 2002.

[CRŠZ07] Andrew M. Childs, Ben W. Reichardt, Robert Špalek, and Shengyu Zhang. Every NAND formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. arXiv:quant-ph/0703015, 2007.

[FGG07] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum algorithm for the Hamiltonian NAND tree. arXiv:quant-ph/0702144, 2007.

[GP03] Anna Gál and Pavel Pudlák. A note on monotone complexity and the rank of matrices. *Information Processing Letters*, 87(6):321–326, 2003.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th ACM STOC*, pages 212–219, 1996.

[Gro02] Lov K. Grover. Tradeoffs in the quantum search algorithm. arXiv:quant-ph/0201152, 2002.

[GV96] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins, Baltimore, 3rd edition, 1996.

[HLŠ05] Peter Høyer, Troy Lee, and Robert Špalek. Tight adversary bounds for composite functions. arXiv:quant-ph/0509067, 2005.

[HLŠ06] Peter Høyer, Troy Lee, and Robert Špalek. Source codes of semidefinite programs for ADV$^{\pm}$. `http://www.ucw.cz/~robert/papers/adv/`, 2006.

[HLŠ07] Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proc. 39th ACM STOC*, pages 526–535, 2007.

[JKS03] T. S. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complexity. In *Proc. 35th ACM STOC*, 2003.

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *Proc. 8th IEEE Symp. Structure in Complexity Theory*, 1993.

[NNP05] Ventzislav Nikov, Svetla Nikova, and Bart Preneel. On the size of monotone span programs. In *Proc. SCN 2004*, LNCS vol. 3352, pages 249–262, 2005.

[RŠ07] Ben W. Reichardt and Robert Špalek. Quantum query complexity of up to 4-bit functions. `http://www.ucw.cz/~robert/papers/gadgets/`, 2007.

[San95] Miklos 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. 6th IEEE Structure in Complexity Theory*, 1991.

[Sni85] Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[SW86] Michael Saks and Avi Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. 27th IEEE FOCS*, pages 29–38, 1986.

[Sze04] Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proc. 45th IEEE FOCS*, pages 32–41, 2004.