

A semi-definite program for quantum query complexity



Ben Reichardt

IQC Institute for
Quantum
Computing

University of Waterloo

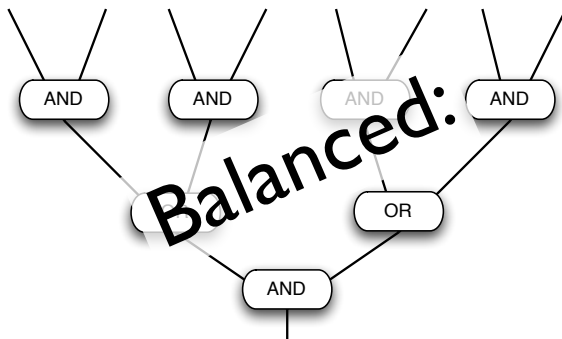
[These slides and notes are for a 30-minute talk given at the second Barriers in Computational Complexity Workshop, Princeton, August 28, 2010. I wrote the notes on the flight home. They are meant to be a rough transcription of the talk.]

This talk is about a new semi-definite program for quantum query complexity, also known as decision-tree complexity. I hope to get across that the SDP is both quite powerful and quite mathematically beautiful. It gives us some optimal quantum algorithms that are wonderfully simple, for problems that seem completely intractable in the classical model.

Composition of algorithms

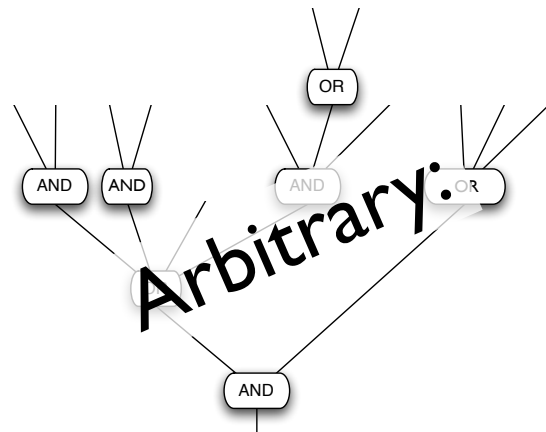
- Given: Boolean functions f, g , and optimal algorithms for each.
- To optimally evaluate $f \circ (g, \dots, g)$, can we just compose the individual algorithms?

AND-OR formulas:



Yes! $\Theta(n^{0.753\dots})$

[Snir '85] [Saks, Wigderson '86] [Santha '95]



No! $n^{0.51} \leq R(\varphi) \leq n$

[SW '86] [Heiman, Wigderson '91]

More general formulas (balanced or not): No! [SW '86] [Jayram, Kumar, Sivakumar '03]

Let me begin the talk with a bit of a teaser, a look at one of the problems we are interested in. A fundamental question in computation is the idea of recursion or composition of algorithms. How does recursion work? How does complexity behave under function composition?

A simple problem to consider is as follows. We are given boolean functions f and g , and optimal algorithms for evaluating each one. We can compose f and g by plugging a copy of g , with its own independent input, for each of f 's input bits. Now what is the complexity of the composed function? Is the optimal algorithm for f composed on g the composition of the optimal algorithms for f and for g ?

We can make the problem more complicated by considering asymptotically many levels of composition or recursion. If each subproblem has its own independent inputs, then these kinds of trees of composed functions are called read-once formulas. They have been studied intensely, but unfortunately our understanding of them remains limited. This is true even for the case of formulas where all the individual functions are just AND and OR functions.

For a balanced AND-OR formula, the answer is yes. The optimal algorithm is a recursive algorithm, and we understand its complexity very well. This is classic work due to Snir, Saks and Wigderson, and Santha. However, for general AND-OR formulas, the answer is no. We do not know what the optimal algorithm is, but we do know that it is not a recursive algorithm. Therefore, we also do not know what the complexity of the problem is, and can only impose very crude bounds. If n is the number of leaves, or inputs, then the complexity lies somewhere between n and $n^{0.51}$, almost a quadratic separation.

Needless to say, for formulas that use more than just AND and OR functions, the situation is no better. In fact, even for balanced formulas, where the individual gates have symmetrical input bits, the optimal algorithm still need not be recursive.

Composition of algorithms

- Given: Boolean functions f, g , and optimal algorithms for each.
- To optimally evaluate $f \circ (g, \dots, g)$, can we just compose the individual algorithms?

Randomized algorithm: 

Quantum algorithm: 

(with a new notion of composition)

Trivial: $Q(f(g, \dots, g)) = O(Q(f)Q(g) \log Q(f))$

Theorem: $Q(f(g, \dots, g)) = \Theta(Q(f)Q(g))$
(e.g., $f = \text{identity} \Rightarrow \text{direct-sum thm.}$)

Theorem: Query-optimal quantum algorithm for evaluating read-once formulas (over any fixed, finite gate set).

Nearly time-optimal algorithm for “constant-factor-imbalanced” formulas.

Is our limited understanding of this “quantum recursion” a barrier to the development of quantum algorithms? -R de Wolf

In contrast, in the quantum model we have come to understand these questions very well. The optimal quantum algorithm for the composed function really is a composition of the individual functions. However, this is true only with a new notion of composition. We are intuitively all very familiar with how classical computation works: you run your routine, and when it wants an input you call a subroutine, which can also call subroutines, and so on. This is not how our quantum composition works, though. The optimal algorithm does have a composed structure. But at runtime, you are somehow running the outer and inner routines all at once! It seems interesting, and we don't fully understand how it works.

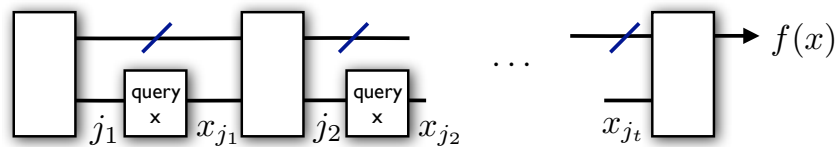
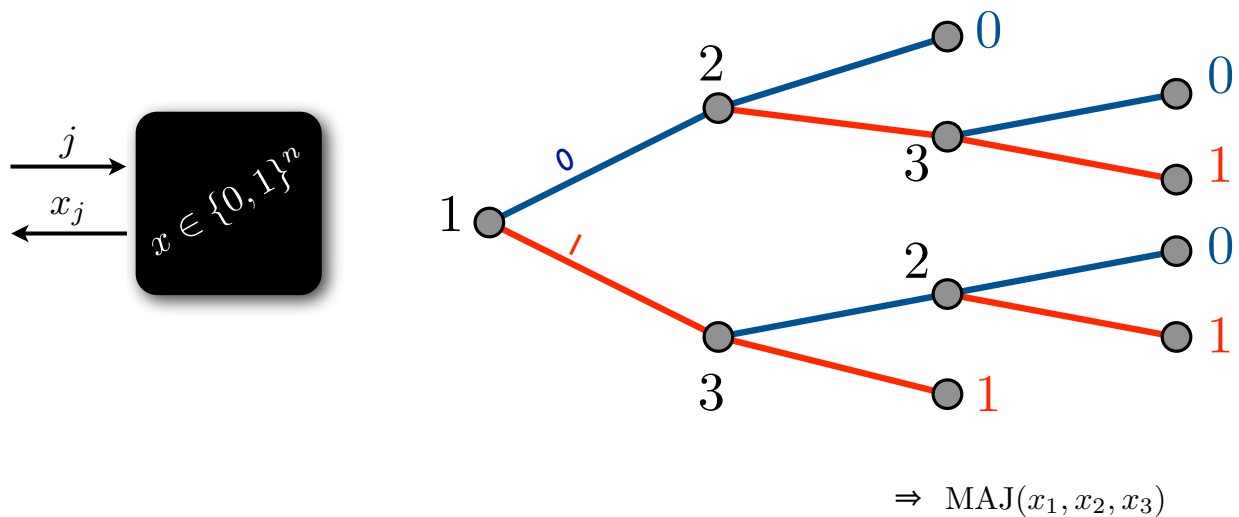
As I was saying, though, we can solve our problems quantumly. It is obvious that the complexity of evaluating f composed on g is at most the complexity of f times the complexity of g , times a logarithmic for error reduction—the log factor is there because we are working in the bounded-error model. In fact, though, we prove that the log factor is not needed, and that the complexity of f composed on g is exactly the product of the complexities of f and of g , up to a constant factor.

As a simple example, if f is the identity function, that simply concatenates its arguments, then this gives a direct-sum theorem. But you can also plug in more interesting functions.

Second, we can solve all of the read-once formula-evaluation problems quantumly. That is, for any fixed, constant-size set of functions, we can easily determine the query complexity and an optimal algorithm for evaluating arbitrary read-once formulas that use those functions. With a certain loose balance condition, we can implement the optimal algorithm with only a polylogarithmic time overhead.

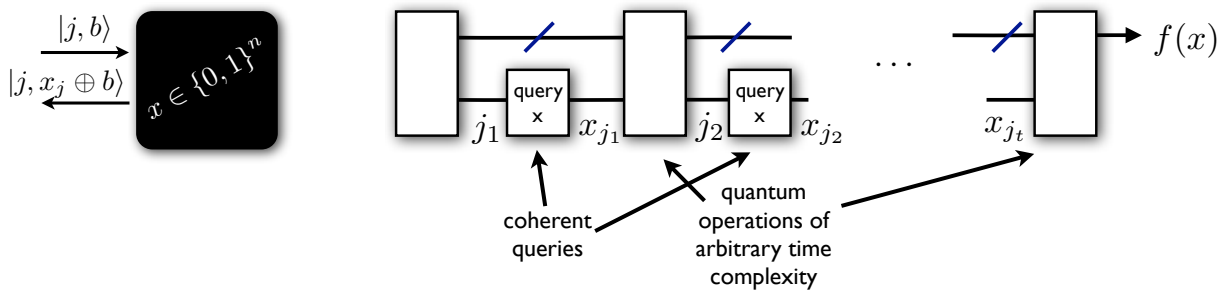
I will explain later a bit about how this new kind of quantum recursion works, but I don't know that we understand it fully. Ronald posed a question to me the other day that perhaps this limited understanding of how recursion should work is a *barrier* to developing new quantum algorithms. I am not sure, but it is an interesting thought.

Decision-tree complexity



Now let me step back and properly remind everyone of the model. I am talking about decision-tree complexity. In this model, you are given a function f , and black-box access to the input string x . Your goal is to evaluate $f(x)$ by looking at as few bits of x as possible. Between queries you can think for as long as you like. The decision-tree complexity is the number of queries you make, or in this tree representation of the computation it is the depth of the tree.

Quantum query complexity



- | | <u>Deterministic</u> | <u>Randomized</u> | <u>Quantum</u> |
|--|----------------------|-------------------|--------------------|
| • Example: $f = \text{OR}_n$ | n | $\Theta(n)$ | $\Theta(\sqrt{n})$ |
| • Note: Query complexity \leq Time complexity (# elementary gates) | | | |
| • Most quantum algorithms are based on good query algorithms | | | |

$$\widetilde{\deg}(f) \leq Q(f) \leq R(f) \leq D(f) \leq \underbrace{\widetilde{\deg}(f)}_{\text{for total functions}}^6$$

While we lose the nice tree picture, the model does generalize quite naturally quantumly. Now we allow the algorithm *coherent* access to the input string x . That is, the algorithm can make queries in superposition and the answers come back in superposition, without collapsing the quantum state. Between queries we still apply arbitrary operations independent of x , but now they are arbitrary quantum operations, or unitaries. Again, we aim to minimize the number of queries made.

As an example that most people here are probably familiar with, the deterministic and randomized query complexities of the n -bit OR function are both n . In contrast, the bounded-error quantum query complexity is only \sqrt{n} , by Grover's search algorithm.

Let me emphasize again that the operations between queries can be of arbitrary complexity. As algorithm designers, we really want to find algorithms that have a time-efficient implementation, which isn't required for query complexity. As it happens, though, good quantum query algorithms have turned out to be essential in the development of many or most good quantum algorithms. For example, Shor's factoring algorithm is at its heart a quantum query algorithm for Period Finding.

Quantum query complexity is also an interesting quantity in its own right. As we will see in a moment, there are good methods for lower-bounding it. It also sits in the zoo of complexity measures, shown here. All these measures are polynomially related for total functions, but there can be exponential gaps for partial functions.

[Barnum, Saks, Szegedy '03] SDP

Theorem: $f: \{0,1\}^n \rightarrow E$ can be evaluated with q queries and error ϵ
 $\Leftrightarrow \exists M_j^{(t)}, M^{(q)}$ p.s.d. $2^n \times 2^n$ matrices ($j=0,\dots,n, t=0,\dots,q-1$) solving

Initial condition:
$$\sum_j M_j^{(0)} = E$$

Query updates:
$$\sum_j M_j^{(t)} = \sum_j E_j \circ M_j^{(t-1)} \quad \text{for } 1 \leq t \leq q-1$$

$$M^{(q)} = \sum_j E_j \circ M_j^{(q-1)}$$

Final condition:
$$|M^{(q)}[x, y]| \leq 2\sqrt{\epsilon(1-\epsilon)} \quad \text{for all } x, y \text{ with } f(x) \neq f(y)$$

$$\left(\begin{array}{l} M_j^{(t)} \sim \text{Gram matrix of state vectors at time } t \\ \text{if bit } j \text{ is to be queried next} \end{array} \quad \begin{array}{l} E[x, y] = 1 \\ E_j[x, y] = (-1)^{x_j + y_j} \end{array} \right)$$

Now the main result of this talk will be a semi-definite program for quantum query complexity. But we already know such an SDP! Barnum, Saks and Szegedy gave one in 2003. Here I have transcribed it for you. A function f can be evaluated with q queries and error ϵ exactly when the following system of equations has a solution using positive semi-definite matrices.

It looks complicated, but actually isn't so bad. For now, just notice its form. There are matrices at each query time. There is one equation that sets up the matrices, putting an initial condition on them for time 0. Then there are updating equations, that relate matrices for time $t-1$ to those at time t . Finally, there is a condition on the last matrix, that it can tell apart the outputs of f with error at most ϵ .

So to use this SDP, you basically guess q , say $q=1000$, and plug these equations into your SDP solver. If the equations are infeasible, then you double q and try again.

In practice, this can be cumbersome. Therefore, instead of working with the dual SDP, approaches to lower-bound quantum query complexity have relied on two other methods.

Two incomparable lower-bound methods

- **Polynomial method:** $Q(f) = \Omega(\widetilde{\deg}(f))$ (for total functions $Q(f) = O(\widetilde{\deg}(f)^6)$)

- **Adversary method:** “How much can be learned from a single query?”

$$\text{Adv}(f) = \max \left\{ \frac{\|\Gamma\|}{\max_{j \in [n]} \|\Gamma \circ \Delta_j\|} : \begin{array}{l} \forall x, y, \Gamma[x, y] \geq 0 \\ \Gamma[x, y] = 0 \text{ if } f(x) \neq f(y) \\ \Delta_j[x, y] = 1 - \delta_{x_j, y_j} \end{array} \right\}$$

	$\widetilde{\deg}$		Adv
Element Distinctness:	$n^{2/3}$	$>$	$n^{1/2}$
Ambainis formula:	$\leq 2^d$	$<$	2.5^d ($n=4^d$)

These two methods are known as the polynomial and adversary methods, and each have found many applications. The first method, the polynomial method, is based on the observation I showed earlier that the quantum query complexity of a function is lower-bounded by the approximate polynomial degree—defined by Robert earlier. Thus any ways of lower-bounding the approximate polynomial degree also work to lower-bound quantum query complexity. It works also in the other direction. Quantum query algorithms give you low-degree approximating polynomials. This gives a different reason for caring about quantum query algorithms that does not depend on there being a time-efficient implementation, on quantum computers ever being implemented, or even on quantum mechanics being a correct theory of the physical universe.

The second method, known as the adversary method, is a type of quantum hybrid argument. It has various equivalent definitions one of which I have specified here. Essentially, the adversary method asks how much information can be learned from a query to the input. The numerator, the norm of the matrix Γ , is the amount of information we want to learn. In the denominator we have $\Gamma \circ \Delta_j$. Here \circ denotes entry-wise matrix multiplication. Δ_j is some fixed 0/1 matrix. Thus $\Gamma \circ \Delta_j$ is a certain submatrix of Γ . Its norm says how much we can learn from querying bit j . Therefore this ratio between how much we need to learn in total and what can be learned per query, intuitively, lower-bounds the quantum query complexity.

These two lower-bound methods are incomparable. For example, here are two problems where one bound is strictly better than the other, and where the other is strictly better than the one.

Two incomparable lower-bound methods

- **Polynomial method:** $Q(f) = \Omega(\widetilde{\deg}(f))$ (for total functions $Q(f) = O(\widetilde{\deg}(f)^6)$)
- **Adversary method:** “How much can be learned from a single query?”

$$\text{Adv}(f) = \max \left\{ \frac{\|\Gamma\|}{\max_{j \in [n]} \|\Gamma \circ \Delta_j\|} : \begin{array}{l} \forall x, y, \Gamma[x, y] \geq 0 \\ \Gamma[x, y] = 0 \text{ if } f(x) = f(y) \\ \Delta_j[x, y] = 1 - \delta_{x_j, y_j} \end{array} \right\}$$

General adversary bound [Høyer, Lee, Špalek '07]

$$\text{Adv}^\pm(f) = \max \left\{ \frac{\|\Gamma\|}{\max_{j \in [n]} \|\Gamma \circ \Delta_j\|} : \begin{array}{l} \forall x, y, \Gamma[x, y] \geq 0 \\ \Gamma[x, y] = 0 \text{ if } f(x) = f(y) \end{array} \right\}$$

However, in 2007 Høyer, Lee and Špalek defined a generalization of the adversary method. It looks nearly exactly the same; in the definition the only difference is that the matrix Gamma now need not be entry-wise nonnegative. They showed that this ratio still lower-bounds quantum query complexity. This bound is certainly at least as good as the original adversary bound, and it can be strictly better.

A new SDP for quantum query complexity

Theorem: The bounded-error quantum query complexity of $f: D \rightarrow E$, $D \subseteq \{0,1\}^n$, is of order $\text{Adv}^\pm(f)$.

$$\begin{aligned} \text{Adv}^\pm(f) &:= \max_{\Gamma} \left\{ \|\Gamma \circ F\| : \forall j \in [n], \|\Gamma \circ F \circ \Delta_j\| \leq 1 \right\} \\ &= \min_{X_j, Y_j \succeq 0} \left\{ \max_{x \in \{0,1\}^n} \sum_{j \in [n]} (X_j + Y_j)[x, x] : \sum_{j \in [n]} (X_j - Y_j) \circ F \circ \Delta_j = F \right\} \\ &\quad \left(\begin{array}{l} F[x, y] = \delta_{f(x), f(y)} \\ \Delta_j[x, y] = 1 - \delta_{x_j, y_j} \end{array} \right) \end{aligned}$$

We show that the general adversary lower bound is actually **tight**. It is also an upper bound; thus it is a semi-definite program that characterizes quantum query complexity, up to constants.

This may be surprising. The original adversary lower bound can be very loose. But a small change to its definition, removing the entry-wise nonnegativity constraint, dramatically increases the power of the bound.

On this slide I have rewritten the SDP and also its dual. Compared to the Barnum/Saks/Szegedy SDP from a couple slides ago, this is a bit simpler, although that may partly be due to my condensed notation. It also has a very different flavor. In their SDP, you guess a q , and feasibility or infeasibility determines whether the query complexity is less or more than q . Their SDP quite directly gives a quantum query algorithm. In this case, though, the objective value of the SDP is itself the query complexity! The algorithm takes more work to find. On the other hand, their SDP works for arbitrary errors ϵ , so you can plug in $\epsilon = 0$ or exponentially small ϵ . The general adversary bound characterizes quantum query complexity only for constant ϵ —bounded-error query complexity.

The algorithm in one slide (for $|E|=2$)

Let $W = \text{Adv}^\pm(f)$, and $\{|v_{xj}\rangle\}$ be a Cholesky decomposition of X_j ($Y_j = 0$ w.l.o.g.).

Arrange them into another matrix

$$B = \left(\underbrace{\frac{1}{\sqrt{W}} \sum_{x \in f^{-1}(0)} |x\rangle, \sum_{x \in f^{-1}(0), j \in [n]} |x\rangle\langle j, \bar{x}_j| \otimes |v_{xj}\rangle}_{1 + 2 \sum_j \text{rank}(X_j) \text{ columns}} \right) \left\{ |f^{-1}(0)| \text{ rows} \right.$$

1. Start on the first column of B .
2. Pick $T \in_R [200W]$. T times reflect about the kernel of B and then query the input.
3. Output 1 if the system is back in the first column.

Originally, the algorithm was a bit complicated. Now, though, it is simple enough to fit on a single slide. So I did. It works starting from an optimal solution to the dual SDP. For simplicity consider the case of boolean codomain; in this case the matrices Y_j may be taken to be zero. Let's start by taking a Cholesky decomposition of the matrices X_j , giving the vectors $v_{\{xj\}}$. That is, X_j is the Gram matrix of the vectors $v_{\{xj\}}$ for different inputs x .

Now put these vectors back together, very carefully, to form a big matrix B . Here is the expression for B . Essentially what is happening is that we are putting the vectors down as the rows of B , placed with care. I'll say a bit more about where this expression comes from later.

The algorithm works on the space of columns of B . Begin in the first column. Now pick a time T uniformly at random from 0 to up to maybe 200 times the adversary bound. T times repeat the following two operations. Reflect around the kernel of B . Then query the input, input bit j if you are in a column labeled j . Then reflect about the kernel of B . Then query the input. Finally, look at the column and output 1 if you are back in the first column. Otherwise output 0.

So the algorithm is very simple, once you have constructed B . Reflecting about the kernel of B may be a complicated operation, but the number of queries made is certainly of order the general adversary bound.

A tiny fraction of **The analysis in one slide**

1. Consider a related matrix, find nice eigenvalue-zero eigenvectors.
2. For a bipartite matrix, nice eigenvalue-zero eigenvectors imply an (effective) spectral gap around zero.
3. Spectral gaps give quantum walk algorithms that lose a log factor [CGMSY'09]. Hand it over to the linear-algebra elves for simpler, optimal algorithms.

Lemma: Let G be a bipartite graph with biadjacency matrix $B_G \in L(V, W)$. Assume that for some $t, \psi \in W$, $B_G^\dagger \psi = 0$ and

$$\frac{|\langle t | \psi \rangle|^2}{\|\psi\|^2} \geq \delta > 0.$$

Let G' have biadjacency matrix $B_{G'} = (t \ B_G)$. Then for all $\Lambda \geq 0$,

$$\sum_{\lambda: |\lambda| \leq \Lambda} |\langle v | \lambda \rangle|^2 \leq \frac{8\Lambda^2}{\delta}.$$

e.g., $\Lambda = 1/(800\sqrt{\delta})$

Unfortunately, as we simplified the algorithm, the analysis became more and more complicated. It doesn't come close to fitting on a single slide. Still, here is a very high-level outline. We start by considering a related matrix to B and finding certain vectors in the kernel, based on the SDP constraints. Then we turn these eigenvalue-zero eigenvectors into a spectral gap centered on zero. Once we have a spectral gap, finding an algorithm is relatively straightforward. The same is true classically if we are running a random walk. Quantumly we can run a quantum walk very easily. To get the algorithm into a really simple form takes a bit more work but is not too difficult.

The reason I am showing you this slide is to highlight the second step, where we turn properties of eigenvalue-zero eigenvectors into a spectral gap, an argument that works for bipartite matrices or bipartite graphs. The main technical lemma is shown here. I think it might have other applications.

Consequences

1

The general adversary lower bound is tight!
(Up to a constant, or up to a log factor if the input alphabet is non-boolean.)
 $\text{Adv}^\pm/\text{wsize}$ SDPs characterize quantum query complexity.

The adversary bound has very nice properties...

$$\text{Adv}^\pm(f \circ (g, \dots, g)) = \text{Adv}^\pm(f) \text{Adv}^\pm(g)$$

What does this result mean? First of all, as already mentioned, it says that the general adversary lower bound is a semi-definite program characterizing quantum query complexity for any function. The bound is tight up to a constant factor, or up to a logarithmic factor if the input alphabet is non-boolean.

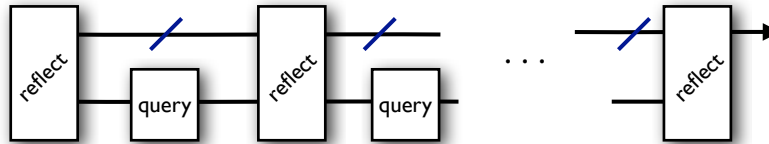
This is interesting both because the SDP has a simple form, compared to the Barnum/Saks/Szegedy SDP, and because the general adversary bound is very mathematically clean. For example, the adversary bound of f composed on g is exactly the adversary bound of f times the adversary bound of g . This means that quantum query complexities also compose multiplicatively, up to only constant factors. Moreover, since this is an exact equality, without losing even a constant, we can repeat the composition as many times as we like, and therefore compute the general adversary bound for any read-once formula.

Consequences

- 1 The general adversary lower bound is tight!
(Up to a constant, or up to a log factor if the input alphabet is non-boolean.)
 $\text{Adv}^\pm/\text{wsize}$ SDPs characterize quantum query complexity.

$$\text{Adv}^\pm(f \circ (g, \dots, g)) = \text{Adv}^\pm(f) \text{Adv}^\pm(g)$$

- 2 Any function can be evaluated optimally by alternating two fixed reflections.



Second, notice that the algorithm we constructed has a quite simple form. It goes back and forth between a certain fixed reflection (about the kernel of B) and queries to the input string. Actually, input queries are also reflections, so the algorithm just alternates two reflections.

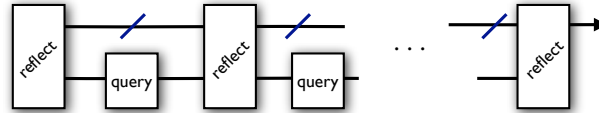
As the product of two reflections is a rotation, the quantum state is actually rotating at a steady speed through space. In terms of our intuition for the adversary bound, this algorithm accumulates information about $f(x)$ at a steady rate.

This two-reflections form of the algorithm is not a trivial fact. But while it is aesthetically pleasing, its practical consequences are unclear.

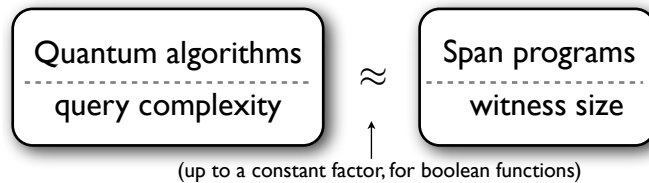
Consequences

- 1 The general adversary lower bound is tight!
(Up to a constant, or up to a log factor if the input alphabet is non-boolean.)
Adv[±]/wsize SDPs characterize quantum query complexity.

- 2 Any function can be evaluated optimally by alternating two fixed reflections.



- 3 Quantum computers are equivalent to span programs.



- 4 Span programs/SDP solutions compose well \Rightarrow New notion of algorithm recursion.

Trivially: $Q(f(g, \dots, g)) = O(Q(f)Q(g) \log Q(f))$

Theorem: $Q(f(g, \dots, g)) = \Theta(Q(f)Q(g))$

Query-optimal (and nearly time-optimal) quantum algorithms for evaluating formulas.

A third consequence is that quantum computers are equivalent to span programs. More precisely, quantum algorithms under the query complexity measure are equivalent to span programs, under the witness size measure, up to constant factors. Span programs are a classical, linear-algebraic model of computation introduced in the early 90s. This connection means that we can develop new quantum algorithms by finding good span programs, and can view previous quantum algorithms as span programs. So if you find the quantum computer model too complicated, with the definitions of qubits, tensor products, unitaries, measurements, projections, queries—instead you can just think about span programs, which have a much simpler geometric definition. I'll explain it in a moment.

Finally, there is a fourth corollary that I already mentioned for the composition question. Span programs compose very easily and intuitively, corresponding to composition of adversary bound dual solutions. This gives us our new notion of algorithm recursion. Don't compose quantum algorithms. Instead, start with the span programs, or convert your algorithms to span programs, then compose those span programs by plugging them together, then convert back to a quantum algorithm. This implies the results mentioned before, query-optimal quantum algorithms for evaluating read-once formulas.

Span programs

- **Def.** [Karchmer, Wigderson '93]: A monotone, n -bit **span program** P is:
 - A **target vector** t in vector space V over \mathbf{C} , and n **subspaces** V_1, \dots, V_n

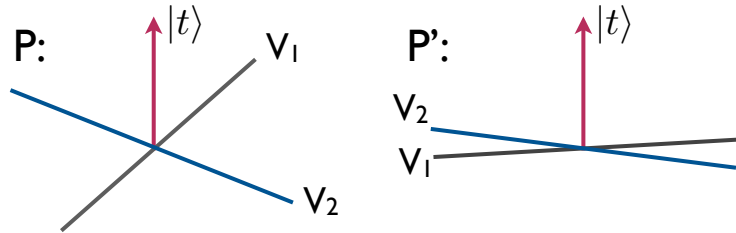
P “computes” $f_P: \{0, 1\}^n \rightarrow \{0, 1\}$,

$$f_P(x) = 1 \Leftrightarrow t \text{ lies in the span of subspaces } \{V_j : x_j = 1\}$$

- **Previous applications:**

- Showing $SL \subseteq \oplus L$ [KW '93]
- Crypto.: Equiv. to linear secret sharing schemes
- Span program size lower bounds give lower-bounds on formula size, branching program size, LSS share length

- **Examples:**



both compute the AND_2 function, but P' has larger “witness size”

Canonical span program [KW'93]:

$$B = \left(\sum_{x \in f^{-1}(0)} |x\rangle, \sum_{x \in f^{-1}(0), j \in [n]} |x\rangle\langle j, \bar{x}_j| \otimes \langle v_{xj}| \right)$$

Now what is a span program? I have given Karchmer and Wigderson’s definition here, but it may be easiest seen by an example. Fix a vector, called the target vector, in some vector space, here in two dimensions. Now put in two subspaces V_1 and V_2 . Here they are one-dimensional but in general they are arbitrary. This setup defines a function according to which subspaces are needed to span the target vector. That’s where the name span program comes from. For example, here I can’t reach the target using only V_1 and I can’t reach it using only V_2 , but adding up a vector in V_1 with a vector in V_2 I can reach the target. Since I need both subspaces, the function computed is the two-bit AND function. If I added a third subspace, that would give the threshold two-of-three function.

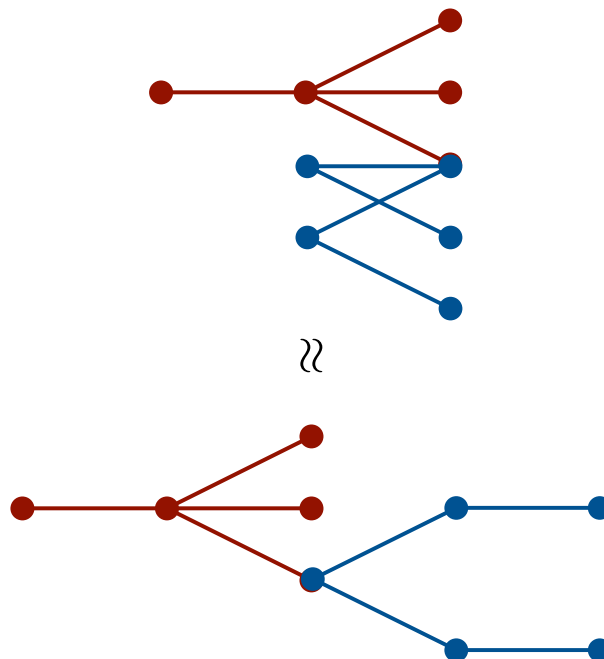
Karchmer and Wigderson also defined “canonical span programs” of a special form. I won’t repeat their definition now, but if you are familiar with it then you can see that the matrix B I used in the algorithm is exactly a canonical span program. So it is actually a very natural object and didn’t just come from nowhere.

Span program composition [R, Špalek '08]

- Given span programs Q and P_j , get $Q \circ \vec{P}$
 - computing $f_{Q \circ \vec{P}} = f_Q \circ \vec{f}_{P_j}$
 - If $\text{wsiz}(P_j) = \alpha_j \Rightarrow \text{wsiz}(Q \circ \vec{P}) \leq \text{wsiz}_{\vec{\alpha}}(Q)$

- Example: $\text{OR}(x_1, x_2, \text{AND}(x_3, x_4))$:

t	x_1	x_2	1	x_3	x_4
1	1	1	1	0	0
0	0	0	1	1	0
0	0	0	1	0	1




As I mentioned before, span programs compose very naturally. Here is a quick example. You can compute the OR function with a span program by using the same one-dimensional space for the target and all input spaces. That way you span the target if and only if the input Hamming weight is at least one. In the red portion of this matrix I have written down this span program, by writing bases for the subspaces in the different columns. In the blue portion of the matrix, I have written out a span program for the AND function. You both of the columns (1,0) and (0,1) in order to span the column (1,1).

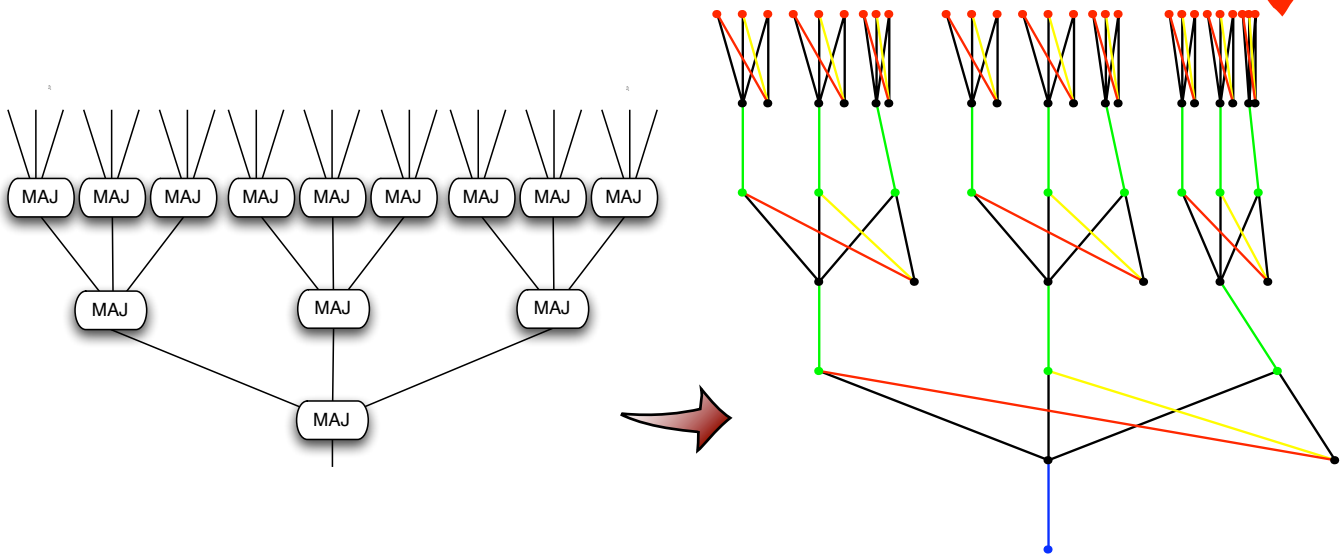
By plugging the matrices together in this way, I obtain a span program for the OR of the AND function.

You can visualize this by considering the matrix as the biadjacency matrix for a bipartite graph, i.e., as giving edge weights from row vertices to column vertices. Then this composition corresponds to plugging graphs together, as shown on the right.

There are other ways of composing span programs, and this seems very natural.

Span program composition

- Given: Boolean functions f, g , and optimal algorithms for each function.
- To optimally evaluate $f \circ (g, \dots, g)$, can we just compose the individual algorithms? 



$$\text{Adv}^{\pm}(f \circ (g, \dots, g)) = \text{Adv}^{\pm}(f) \text{Adv}^{\pm}(g)$$

For example, to evaluate a large read-once formula, we start with span programs for the individual gates and compose them all together. In terms of graphs, you plug together little graph gadgets for each function in the formula, as you can see on the right. This is what is really going on in our notion of quantum composition.

Open problems & Barriers

1

The general adversary lower bound is tight!
(Up to a constant, or up to a log factor if the input alphabet is non-boolean.)
 $\text{ADV}_{\pm}/\text{wsize}$ SDPs characterize quantum query complexity.

(The adversary bound has very nice properties...)

Open problems:

1. Use this to find more new algorithms, even time-efficient algorithms, such as for:

- Triangle Finding, Graph Collision

2. Understand connections to other complexity measures, such as:

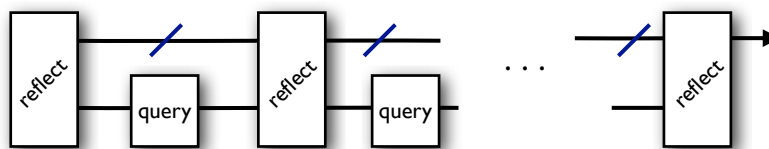
- $\text{Adv}(f) \leq \text{Adv}^{\pm}(f)$
- Sensitivity
- Classical query complexities. For total functions, largest known gap is quadratic, but best bound is $D(f) \leq Q(f)^6$.

Let me now turn to some open problems and obstacles. Regarding the semi-definite program, we would like to use this to derive new quantum query algorithms. Some candidate problems include Triangle Finding, which is useful in boolean matrix multiplication, and Graph Collision, which is useful in Triangle Finding. Also, perhaps we can use the general adversary bound to better understand some of the other function complexity measures. For example, for total functions, it is known that the deterministic query complexity is at most the sixth power of the quantum query complexity, but the largest known gap is quadratic. To narrow this gap, we would probably want somehow to use the clean composition properties of the adversary bound.

Open problems & Barriers

2

Any function can be evaluated optimally by alternating two fixed reflections.



Open problems:

Conjecture: Quantum query complexity is unchanged with a coherent bounded-error input oracle.

(Classically, an extra logarithmic factor is sometimes, but not always, needed [Feige, Raghavan, Peleg, Upfal '94].)

Conjecture known for OR_n and almost every function.

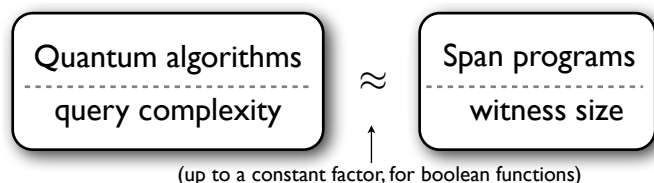
Reflection form may allow for adapting the analysis of [Høyer, Mosca, de Wolf '04].

May also allow for better, more direct understanding of quantum recursion. Instead of composing span programs, compose algorithms in reflection form.

Second, regarding the very clean form of the algorithm, as alternating two fixed reflections—this seems like it could be useful, but it is not clear for what. One potential application is to query complexity with faulty oracles. Say that when we ask for bit j , we get x_j with 90% probability and the complementary bit with 10% probability. How does query complexity change? Clearly, with log steps of repetition we can get rid of the error. Classically, this repetition is sometimes needed and sometimes not. Quantumly, if we say that the error is coherent, then the log overhead is never known to be needed. For the OR function and almost every function, it is known not to be needed. For the parity function, which should be the most sensitive to noise, it is not needed. I conjecture that a constant-factor overhead always suffices in the quantum model. Since our algorithm uses two reflections, just like Grover's search algorithm, it may be possible to adapt the analysis of Hoyer, Mosca and de Wolf from the search problem to every problem. This reflection form may also allow for a different understanding of optimal recursion. Instead of plugging together span programs, it may be possible to plug together quantum algorithms more directly, so long as the algorithms have been converted into this reflection standard form.

Open problems & Barriers

- 1 The general adversary lower bound is tight!
(Up to a constant, or up to a log factor if the input alphabet is non-boolean.)
ADV \pm /wsize SDPs characterize quantum query complexity.
- 2 Any function can be evaluated optimally by alternating two fixed reflections.
- 3 Quantum computers are equivalent to span programs.



Open problems:

- “Correct” definition of span programs for non-boolean functions?
- Use span programs to design new quantum algorithms.

Regarding span programs, the most interesting problem is to use span programs to develop new quantum algorithms. We can also re-derive old algorithms. A simpler problem is to figure out the right definition of span programs for non-boolean functions. The definition I gave works only for boolean output. Either you can span the target vector or you can't. In the non-boolean case, we know what the right primal and dual SDPs are, which basically tells us how “canonical” span programs should generalize. But we don't know what the right definition for general span programs is. Getting this definition right could be useful for developing new quantum algorithms, and could even have applications in the classical theory of span programs.

Open problems & Barriers

4

Span programs/SDP solutions compose well \Rightarrow New notion of algorithm recursion.

Trivially: $Q(f(g, \dots, g)) = O(Q(f)Q(g) \log Q(f))$

Theorem: $Q(f(g, \dots, g)) = \Theta(Q(f)Q(g))$

Query-optimal (and nearly time-optimal) quantum algorithms for evaluating formulas.

Open problems:

- Fully understand this notion of quantum recursion & applications.
- Determine the quantum query complexity of read-once formulas over non-boolean gate sets.

Hard example: $f \circ (g, \dots, g)$ where

$$f : \{0, 1, 2\}^n \rightarrow \{0, 1\}$$

$$g(x) = \begin{cases} 2 & \text{if } x_1 = 0 \\ \tilde{g}(x) \in \{0, 1\} & \text{otherwise} \end{cases}$$

Lastly, I think this new notion of recursion needs more study to really understand it. Understanding optimal recursion should help us develop new quantum algorithms. One difficult problem might be to try to understand the query complexity of read-once formulas that use non-boolean intermediate functions. Classically, it would be crazy to consider this problem, since the boolean case is already so hard. Quantumly, though, the problem may be approachable.

This example might show why non-boolean function recursion is so much more complicated. Let the outer function f have ternary input alphabet. Let $g(x)$ be 2 if the first bit is 0 and otherwise be some complicated boolean function. Thus it is easy to tell whether $g(x)$ is 2 or not, but it could be difficult to tell a 0 output from a 1. The complexity of the composed function will depend on how f distinguishes between 0, 1 and 2. If f treats 0 and 1 as the same, then the composed function is easy to evaluate. I think this example shows that a complexity measure for non-boolean functions will need to be more than just a single scalar, like the adversary bound, if we want to be able to compose lower bounds well.

References:

AND-OR formulas:

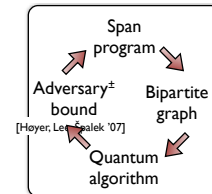
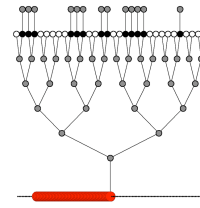
- Farhi, Goldstone, Gutmann quant-ph/0702144
- Ambainis, Childs, R, Špalek, Zhang '10, R 0907.1623

General adversary bound:

- Høyer, Lee, Špalek quant-ph/0611054

Span programs and quantum query complexity:

- R, Špalek 0710.2630: Span programs for evaluating formulas
- R 0907.1622: Formula-evaluation extensions
- R 1005.1601: Reflection-based optimal algorithm
- Lee, Mittal, R, Špalek '10?: Non-boolean function evaluation
- R 0904.2759: Optimal span programs and quantum algorithm
 - Portions revised into ECCC TR10-075 and TR10-110



Thank you!

Finally, here are a few references. I mostly would like to acknowledge my coauthors, Robert Spalek and, for the work on evaluating non-boolean functions, Troy Lee, Rajat Mittal and Robert.

Also, if you are interested in this area, there are a number of papers now, some of which are quite long, but these two ECCC technical reports I hope give a good introduction and summary.

Thank you.