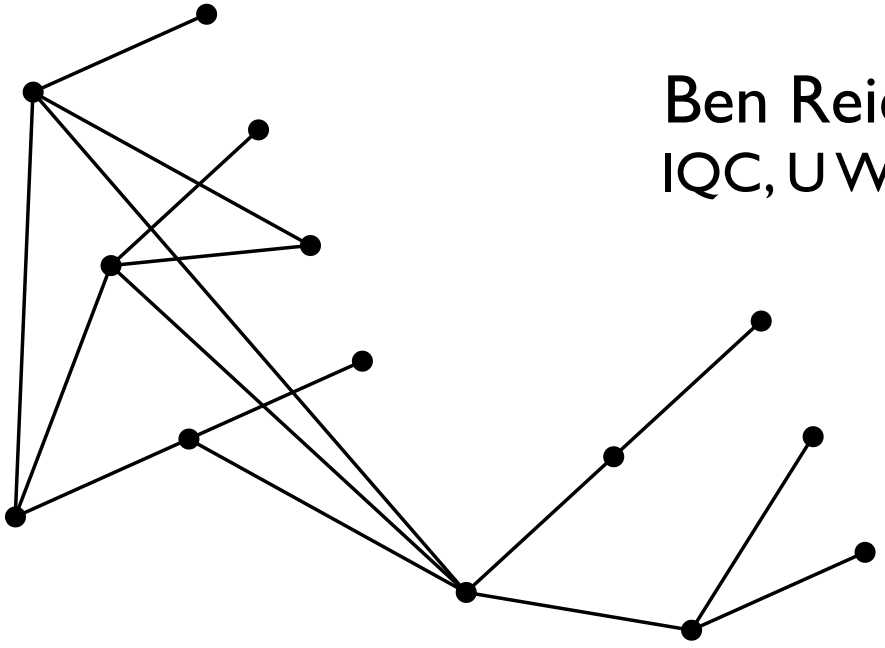
Span programs and quantum query algorithms



Ben Reichardt
IQC, U Waterloo

[arXiv:0904.2759]

Equivalent models for quantum algorithms

- (Classically controlled) quantum circuit model (with faults)
- Adiabatic quantum computing
- Anyonic models
- Quantum Turing machine
- Quantum walks
- Cluster states

Consequences

⇒ Universality of circuit model

⇒ Implementations

⇒ New algorithms (e.g., approximating Turaev-Viro, adiabatic optimization)

⇒ QMA-complete problems

Equivalent models for quantum algorithms

- (Classically controlled) quantum circuit model (with faults)
- Adiabatic quantum computing
- Anyonic models
- Quantum Turing machine
- Quantum walks
- Cluster states

Consequences

⇒ Universality of circuit model

⇒ Implementations

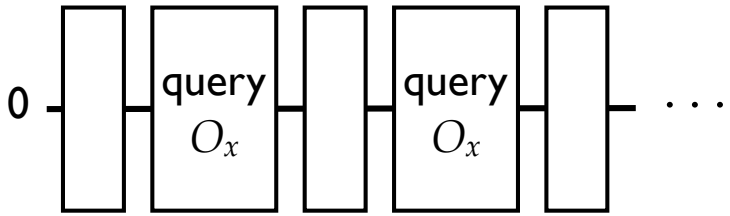
⇒ New algorithms (e.g., approximating Turaev-Viro, adiabatic optimization)

⇒ QMA-complete problems

Equivalent models for quantum query algorithms

(on input $x \in \{0,1\}^n$)

- Discrete-query model



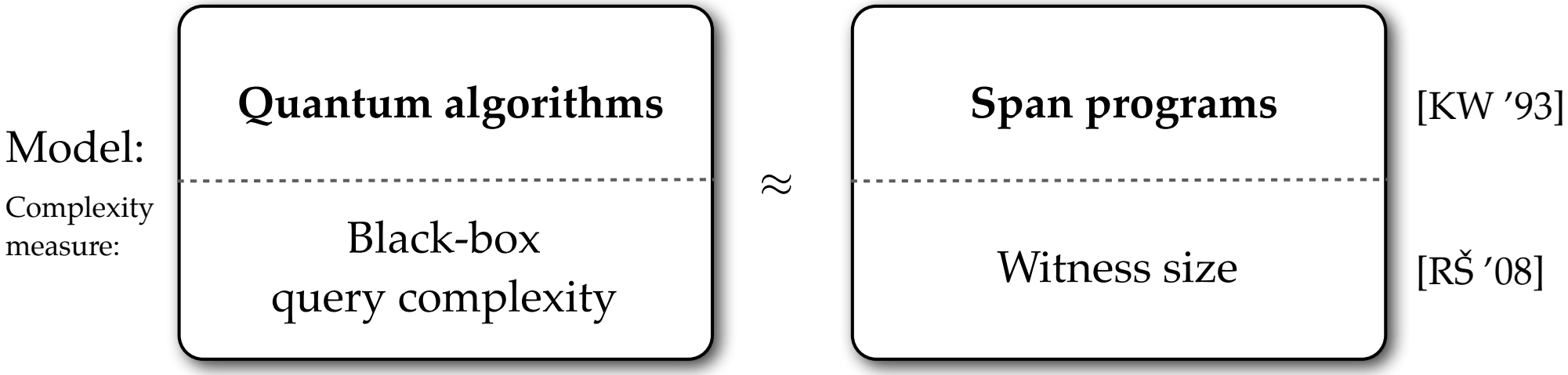
$$O_x|j,b,\alpha\rangle = |j, x_j \oplus b, \alpha\rangle$$

- Continuous-query model

Driving Hamiltonian (independent of x) and Oracle Hamiltonian

$$H_x = \sum_{j=1}^{n} x_j |j\rangle\langle j| \otimes \mathbf{1}$$

[CGMSY '09]


Model:
Complexity measure:
Quantum algorithms
Black-box query complexity
≈
Span programs
Witness size
[KW '93]
[RŠ '08]

Quantum query complexity Q(f)

- Time complexity = number of elementary gates
- Query complexity = number of input bits that must be looked at
  - e.g., Search:
    - classical query complexity (AKA "decision-tree complexity") of OR_n is $\Theta(n)$ for both deterministic & randomized algorithms
    - quantum query complexity is $Q(OR_n)=\Theta(\sqrt{n})$, by Grover search
  - Most quantum algorithms are based on good qu. query algorithms
  - *Provable* lower bounds

Two methods to lower-bound Q(f)

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

$$\mathrm{Adv}(f) = \max_{\substack{\text{adversary matrices } \Gamma: \\ \Gamma \geq 0}} \frac{\|\Gamma\|}{\max_{j \in [n]} \|\Gamma_j\|}$$

- Incomparable lower bounds:

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

Polynomial method $Q(f) = \Omega(\widetilde{\deg}(f))$

Adversary bound $Q(f) = \Omega(\mathrm{Adv}(f))$

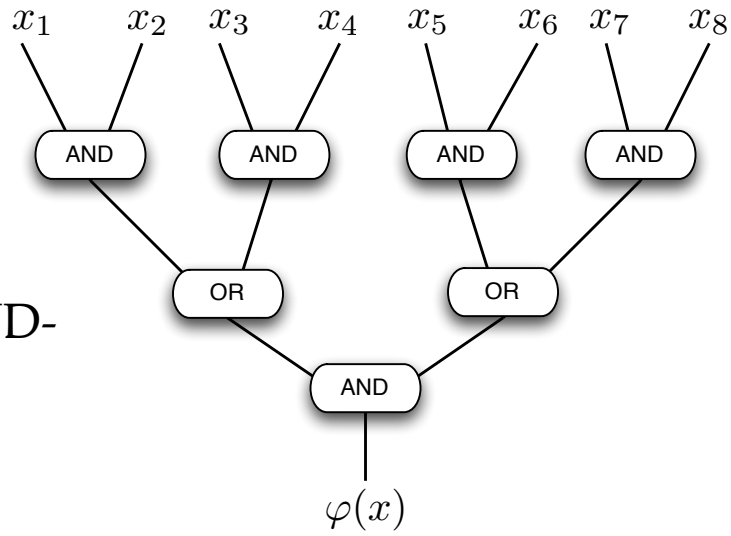
$$\mathrm{Adv}(f) = \max_{\substack{\text{adversary matrices } \Gamma: \\ \Gamma \geq 0}} \frac{\|\Gamma\|}{\max_{j \in [n]} \|\Gamma_j\|}$$

- **General adversary bound** [Høyer, Lee, Špalek '07] $Q(f) = \Omega(\mathrm{Adv}^{\pm}(f))$

$$\mathrm{Adv}^{\pm}(f) = \max_{\text{adversary matrices } \Gamma} \frac{\|\Gamma\|}{\max_{j \in [n]} \|\Gamma_j\|}$$

	$\widetilde{\deg}$	Adv	$\mathrm{Adv}^{\pm}$	Q	
Element Distinctness:	$n^{2/3}$	$n^{1/3}$	??	$n^{2/3}$	
Ambainis formula:	$\leq 2^d$	2.5^d	2.513^d	??	$(n=4^d)$

AND-OR formula-evaluation algorithms



- **Theorem** ([FGG '07]): A balanced binary AND-OR formula can be evaluated with $N^{1/2+o(1)}$ queries.

Unbalanced AND-OR

Balanced, More gates

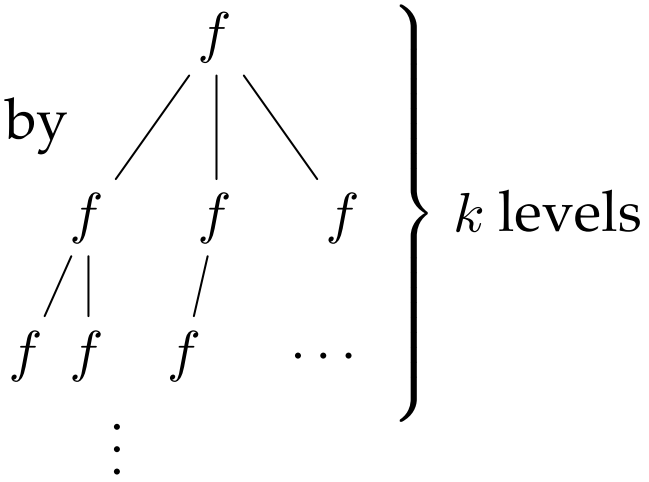
- **Theorem** ([ACRŠZ '07, R '09]):
  - An "approximately balanced" AND-OR formula can be evaluated with $O(\sqrt{N})$ queries (optimal!).
  - A general AND-OR formula can be evaluated with $N^{1/2+o(1)}$ queries.

- **Theorem** ([RŠ '08]): A balanced formula ϕ over a gate set including all three-bit gates can be evaluated in $O(Adv(\phi))$ queries (optimal!).

(Running time is poly-logarithmically slower in each case, after preprocessing.)

Span programs [Karchmer, Wigderson '93]

- **Theorem** ([R, Špalek '08]): Let $f : \{0,1\}^n \to \{0,1\}$
  Define $f^k : \{0,1\}^{n^k} \to \{0,1\}$ by



Let P be a *span program* computing f.
Then

$$Q(f^k) = O\big(\mathrm{wsize}(P)^k\big)$$

quantum query complexity

span program complexity measure

➡ Many optimal algorithms: for f any ≤3-bit function (e.g., AND, OR, PARITY, MAJ_3), and ~70 of ~200 different 4-bit functions…

Open problems:

- How can we find more good span programs?

$$m = \begin{pmatrix} \{\} & \{x_1\} & \{x_1\} & \{x_2\} & \{x_2\} & \{x_3\} & \{x_3\} & \{x_4\} & \{x_4\} & \{\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}\} & \{\overline{x_1}\} & \{\overline{x_2}\} & \{\overline{x_3}\} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w3 & 0 & 0 & 0 \\ w1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & -1 & i & -i & i & i & 0 & 0 & 0 & 0 \\ 0 & i & -i & i & i & 1 & 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ w2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix};$$

```
pevalSPC[m]
```

- What is the connection to the adversary bounds?

#	Size	Adv	Adv$^{\pm}$	Status	Comments
7128	10	2.50000	2.51353	2.77394	sorted input bits [Amb06a], $(x_1 \wedge ((x_2 \wedge x_3) \vee (\overline{x_3} \wedge \overline{x_4}))) \vee (\overline{x_1} \wedge ((\overline{x_2} \wedge \overline{x_3}) \vee (x_3 \wedge x_4)))$
863	5	2.00000	2.07136	2.22833	monotone two adjacent 1s, $(x_1 \wedge x_2) \vee (x_4 \wedge (x_1 \vee x_3))$
427	5	2.18398	2.20814	2.22833	$\#975(x_1 \wedge x_2, x_3, x_4)$
27	5	$\sqrt{5}$	–	✓ Lemma 4.12	$x_1 \wedge \#975(x_2, x_3, x_4)$
393	6	$4/\sqrt{3}$	–	✓ opt. NAND	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_4)$
383	7	2.30278	2.34406	$\sqrt{4+\sqrt{3}}$	$(x_1 \wedge x_2 \wedge x_3) \vee ((x_1 \vee x_2 \vee x_3) \wedge x_4)$
126	7	$\sqrt{11/2}$	–	✓ Lemma 4.12	$x_1 \wedge \neg \mathrm{EQUAL}_3(x_2, x_3, x_4)$
24	7	$\sqrt{11/2}$	–	✓ Lemma 4.12	$x_1 \wedge \mathrm{EQUAL}_3(x_2, x_3, x_4)$
303	6	2.35829	–	$1+\sqrt{2}$	$((x_1 \vee x_2) \wedge x_3) \vee ((\overline{x_1} \vee x_2) \wedge x_4)$, span program size 5
495	6	$1+\sqrt{2}$	–	✓ opt. NAND	$\#975(x_1, x_2, x_3 \wedge x_4)$
989	6	$1+\sqrt{2}$	–	✓ opt. gadget	$(x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge (\overline{x_2} \vee x_4))$
965	7	2.41531	2.42653	2.59234	$(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge \overline{x_3}) \vee (\overline{x_4} \wedge \overline{x_2})$

Open problems:

- How can we find more good span programs?
- What is the connection to the adversary bounds?
- Are span programs useful for developing other qu. algorithms?

Answers:

- **Theorem 1:** For any boolean function f,

$$\inf_{P \text{ computing } f} \text{wsize}(P) = \text{Adv}^{\pm}(f)$$

- **Theorem 2:** For any span program P computing f,

$$Q(f) = O\left(\text{wsize}(P)\frac{\log \text{wsize}(P)}{\log\log \text{wsize}(P)}\right)$$

The general adversary bound is nearly tight

- **Corollary:** For any $f : \{0,1\}^n \to \{0,1\}$

$$Q(f) = \Omega(\mathrm{Adv}^{\pm}(f)) \quad \text{[HLŠ '07]}$$

$$\text{and } Q(f) = O\left(\mathrm{Adv}^{\pm}(f)\frac{\log \mathrm{Adv}^{\pm}(f)}{\log\log \mathrm{Adv}^{\pm}(f)}\right)$$

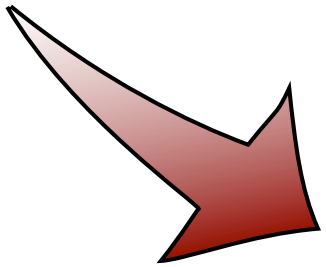
- Nearly tight characterization of quantum query complexity; the general adversary bound is always (almost) optimal

	Adv	$\widetilde{\deg}$	$\mathrm{Adv}^{\pm}$	Q	
Element Distinctness:	$n^{1/3}$	$n^{2/3}$	$\geq n^{2/3}/\log n$	$n^{2/3}$	
Ambainis formula:	2.5^d	$\leq 2^d$	2.513^d	2.513^d	$(n=4^d)$

- Simpler, "greedier" semi-definite program than [Barnum, Saks, Szegedy '03]

- **Theorem 1:** $\inf_{P \text{ computing } f} \text{wsize}(P) = \text{Adv}^{\pm}(f) = O(Q(f))$

- **Theorem 2:** If P computes f, $Q(f) = O\left(\text{wsize}(P)\frac{\log \text{wsize}(P)}{\log\log \text{wsize}(P)}\right)$



Span programs are equivalent to quantum computers!

(up to a log factor)

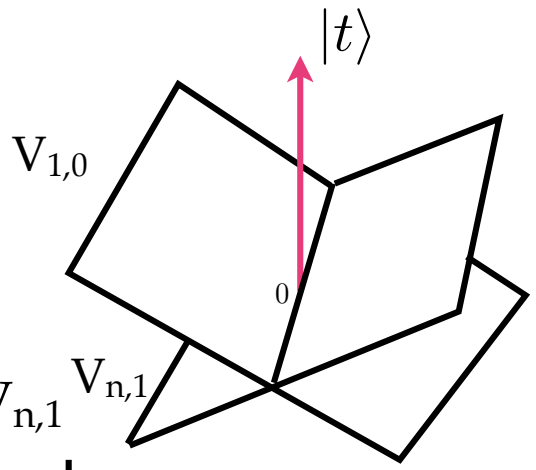
Model:

Complexity measure:

Quantum algorithms query complexity	$\approx$	Span programs witness size

Also, leads to new quantum algorithms for evaluating formulas, exact formula for the composition of the general adversary bound…

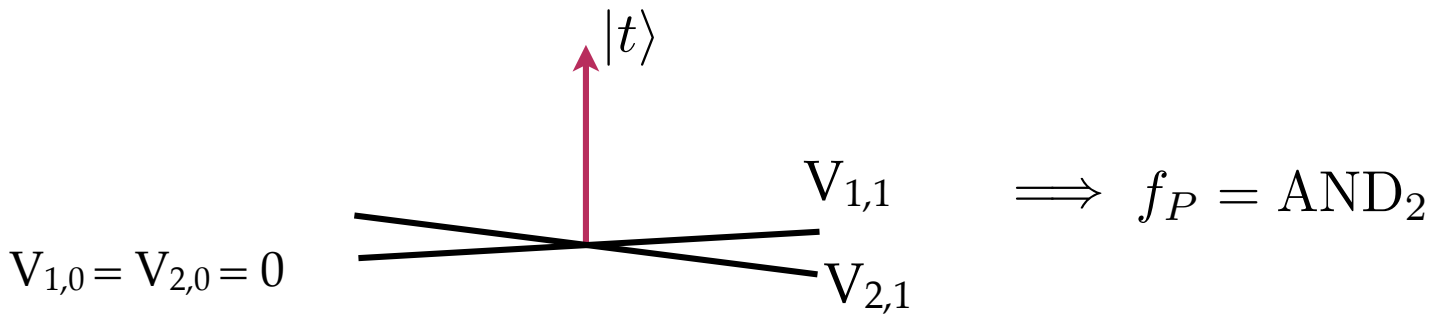
- **Definition:** **Span program** P on n bits
  - target vector $|t\rangle$ in vector space V
  - subspaces $V_{1,0}$ $V_{1,1}$ $V_{2,0}$ $V_{2,1}$ … $V_{n,0}$ $V_{n,1}$



  - P "computes" $f_P : \{0,1\}^n \to \{0,1\}$

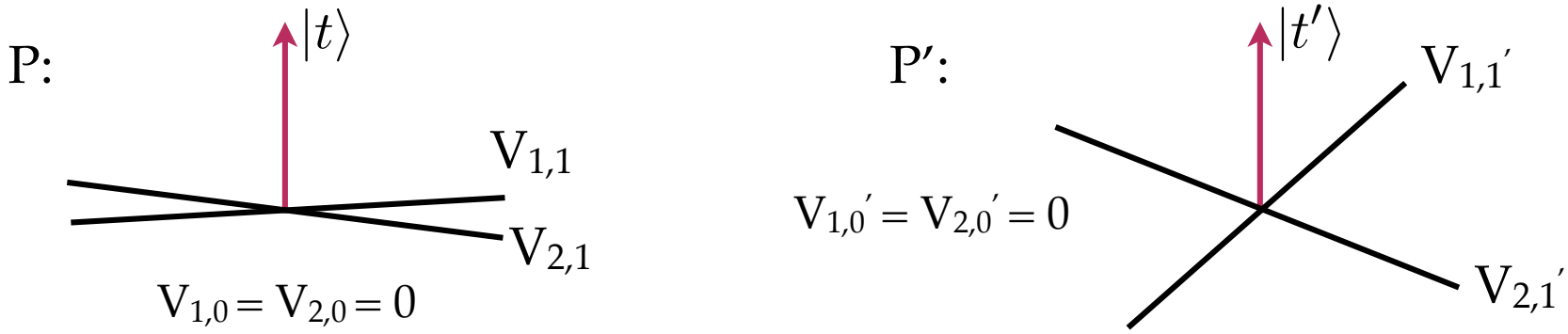
$$f_P(x) = 1 \iff |t\rangle \in \text{Span} \cup_j V_{j,x_j}$$

- **Example:** V=$\mathbf{C}^2$



- (Very simple! No qubits, ancillas or Hamiltonians…)

Example



- $f_P = f_{P'} = \mathrm{AND}_2$ but P′ seems better…

$$\mathrm{wsize}(P, 11) > \mathrm{wsize}(P', 11)$$

Span programs in coordinates

- Span program P: target $|t\rangle$

$$A = \left(\begin{array}{cccc} \overbrace{|v_{1,0,1}\rangle \cdots |v_{1,0,m}\rangle}^{V_{1,0}} & \overbrace{|v_{1,1,1}\rangle \cdots |v_{1,1,m}\rangle}^{V_{1,1}} & \cdots & \overbrace{|v_{n,0,1}\rangle \cdots |v_{n,0,m}\rangle}^{V_{n,0}} \quad \overbrace{|v_{n,1,1}\rangle \cdots |v_{n,1,m}\rangle}^{V_{n,1}} \end{array} \right)$$

$\Pi(x)$ = projection onto available columns of A

Then

$$f_P(x) = 1 \iff |t\rangle \in \text{Range}(A\Pi(x))$$

- Def.: If $f(x) = 1$, let $\text{wsize}(P, x) = \min_{|w\rangle:\, A\Pi(x)|w\rangle = |t\rangle} \||w\rangle\|^2$

(intuition: want a short witness)

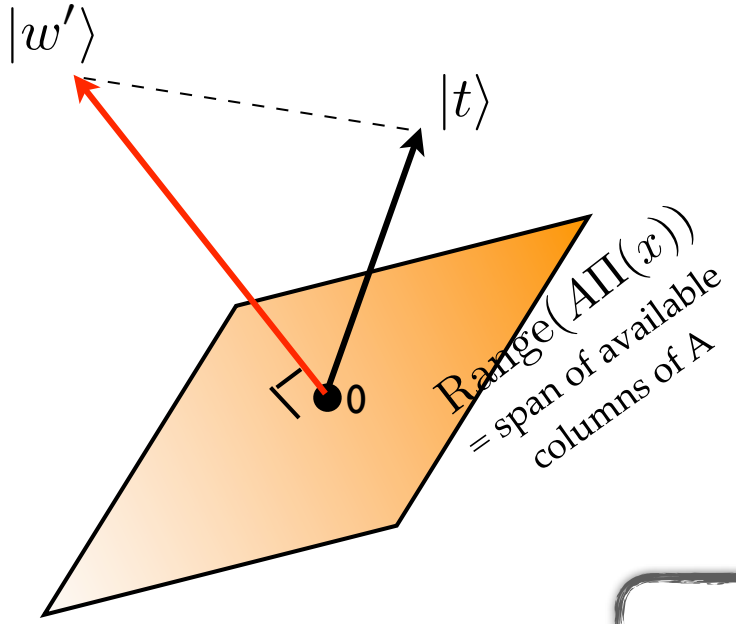
$$f_P(x) = 1 \implies |t\rangle \in \text{Range}(A\Pi(x))$$

$$\text{wsize}(P, x) = \min_{|w\rangle:\, A\Pi(x)|w\rangle = |t\rangle} \||w\rangle\|^2$$

(intuition: want a short witness)

$$f_P(x) = 0 \implies |t\rangle \notin \text{Range}(A\Pi(x))$$

$$\iff \exists\, |w'\rangle \text{ orthogonal to } \text{Range}(A\Pi(x)) \text{ with } \langle t|w'\rangle \neq 0$$



$$\text{wsize}(P, x) = \min_{\substack{|w'\rangle:\, \langle t|w'\rangle = 1 \\ \langle w'|\Pi(x)A = 0}} \|A^\dagger |w'\rangle\|^2$$

(intuition: if $|t\rangle$ is *close* to the span of available columns of A, then wsize should be *large*)

Definition: $\text{wsize}(P) = \max_x \text{wsize}(P, x)$

Example: Search (OR)

- Define a span program P as follows:
  - Vector space V = **C**
  - Target vector $|t\rangle = n^{1/4}$

$$A = \begin{pmatrix} \overbrace{0}^{V_{1,0}} & \overbrace{1}^{V_{1,1}} & \overbrace{0}^{V_{2,0}} & \overbrace{1}^{V_{2,1}} & \cdots & \overbrace{0}^{V_{n,0}} & \overbrace{1}^{V_{n,1}} \end{pmatrix}$$

➡ $f_P = \mathrm{OR}_n$

$$\mathrm{wsize}(P, 0^n) = \sqrt{n} \qquad |w'\rangle = 1/n^{1/4}$$

$$\mathrm{wsize}(P, 10\ldots0) = \sqrt{n} \qquad |w\rangle = (0, n^{1/4}, 0, \ldots, 0) \qquad \ldots$$

➡ $\mathrm{wsize}(P) = \sqrt{n}$

$$f_P(x) = 1 \implies \mathrm{wsize}(P,x) = \min_{|w\rangle:\, A\Pi(x)|w\rangle = |t\rangle} \||w\rangle\|^2$$

$$f_P(x) = 0 \implies \mathrm{wsize}(P,x) = \min_{\substack{|w'\rangle:\, \langle t|w'\rangle = 1 \\ \langle w'|\Pi(x)A = 0}} \|A^\dagger |w'\rangle\|^2$$

Definition: $\mathrm{wsize}(P) = \max_x \mathrm{wsize}(P,x)$

- Why is this the right definition?

  1. *Negating* a span program leaves wsize invariant

  2. *Composing* span programs: wsize is multiplicative

  $$f_{P\circ Q} = f_P \circ f_Q \qquad \mathrm{wsize}(P \circ Q) = \mathrm{wsize}(P)\mathrm{wsize}(Q)$$

  3. Leads to quantum *algorithms* $Q(f_P) = \tilde{O}\big(\mathrm{wsize}(P)\big)$ (Theorem 3)

Proof of Theorem 1

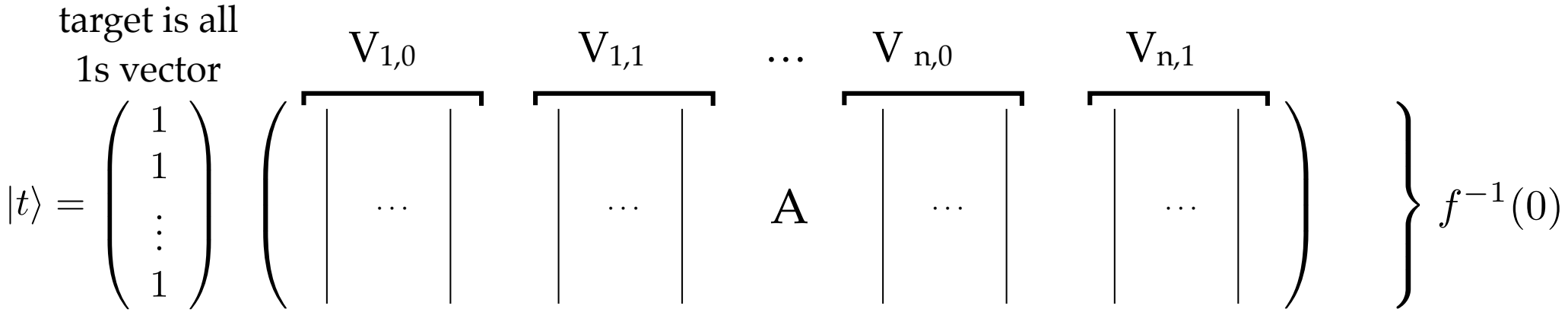
- **Theorem 1:** For any boolean function f, $\inf_{P:\, f_P = f} \mathrm{wsize}(P) \leq \mathrm{Adv}^{\pm}(f)$

Example: AND

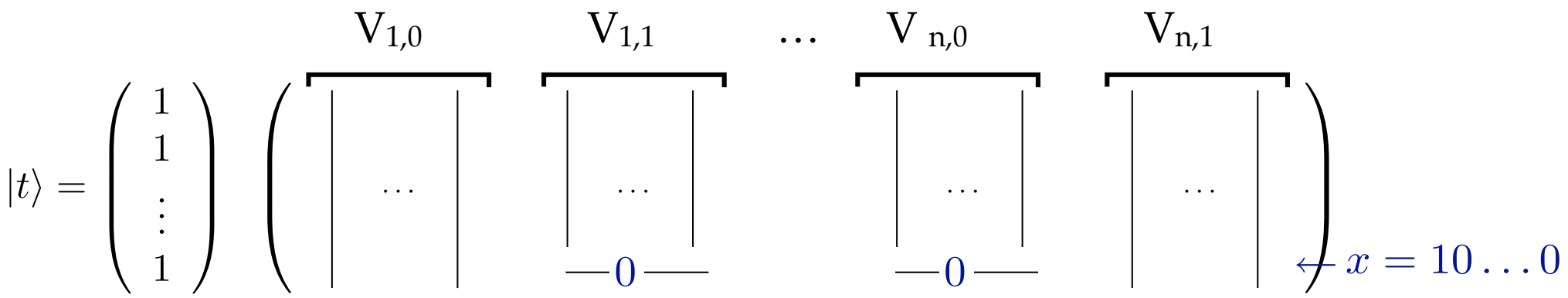
$$|t\rangle = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \begin{pmatrix} \overbrace{\begin{matrix} 1 \\ 0 \\ \vdots \\ 0 \end{matrix}}^{V_{1,1}} & \overbrace{\begin{matrix} 0 \\ 1 \\ \vdots \\ 0 \end{matrix}}^{V_{2,1}} & \cdots & \overbrace{\begin{matrix} 0 \\ 0 \\ \vdots \\ 1 \end{matrix}}^{V_{n,1}} \leftarrow x = 011\ldots1 \end{pmatrix}$$

➡ $f_P = \mathrm{AND}_n$

Consider span programs where the *rows* of A correspond to $\{x : \mathrm{f}(x) = 0\}$



Consider span programs where the *rows* of A correspond to $\{x : f(x) = 0\}$

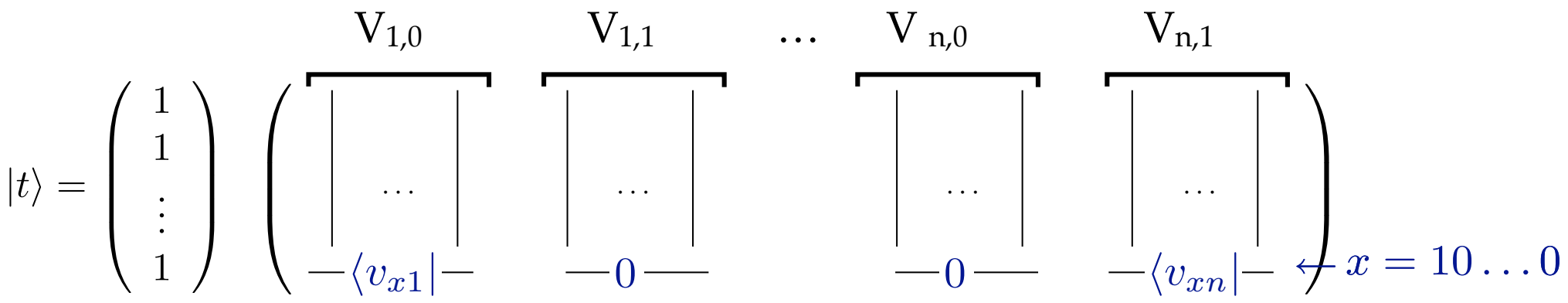


…and in the row corresponding to x,
the columns available for input x are all *zero*

(Such span programs are said to be in "canonical form" [KW'93].)

This form guarantees that $f(x) = 0 \Rightarrow f_P(x) = 0$

($|w'\rangle = |x\rangle$ itself is the witness)



in the xth row, the columns available for input x are all 0; hence

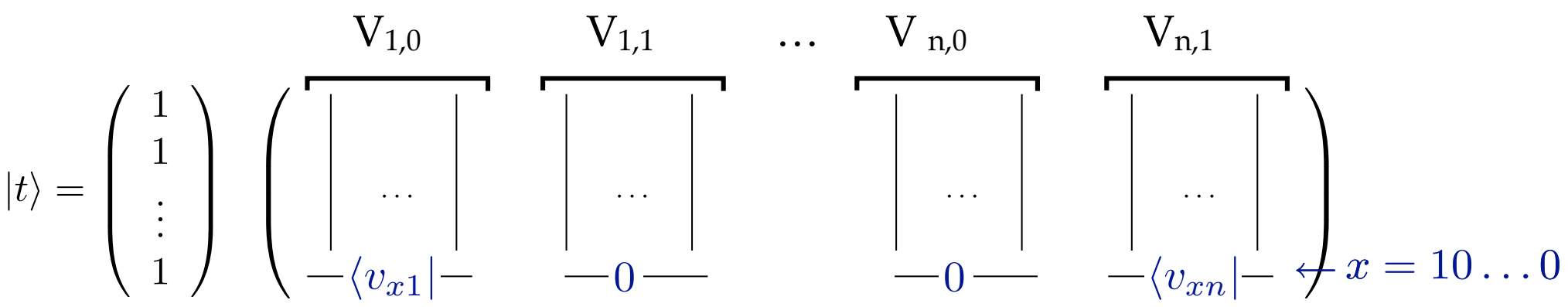
$$f(x) = 0 \;\Rightarrow\; f_P(x) = 0$$

Now consider a $y \in f^{-1}(1)$

We want to find vectors $|v_{y1}\rangle, \ldots, |v_{yn}\rangle$

such that $\forall\, x \in f^{-1}(0)\,, \quad 1 = \sum_{j:x_j \neq y_j} \langle v_{xj}|v_{yj}\rangle$

The witness size is $\max_x \sum_j \| |v_{xj}\rangle \|^2$



$$\implies \inf_{P:\, f_P = f} \mathrm{wsize}(P) \leq \inf_{\substack{\{|v_{xj}\rangle\}:\\ \text{if } f(x)\neq f(y),\, \sum_{j:x_j\neq y_j} \langle v_{xj}|v_{yj}\rangle = 1}} \max_x \sum_j \big\||v_{xj}\rangle\big\|^2$$

$$= \min_{\substack{X\succeq 0:\\ \forall (x,y)\in\Delta,\, \sum_{j:x_j\neq y_j} \langle x,j|X|y,j\rangle = 1}} \max_x \sum_j \langle x,j|X|x,j\rangle$$

(Cholesky decomposition)

$$= \mathrm{Adv}^{\pm}(f)$$

(SDP duality) □

Proof of Theorem 2

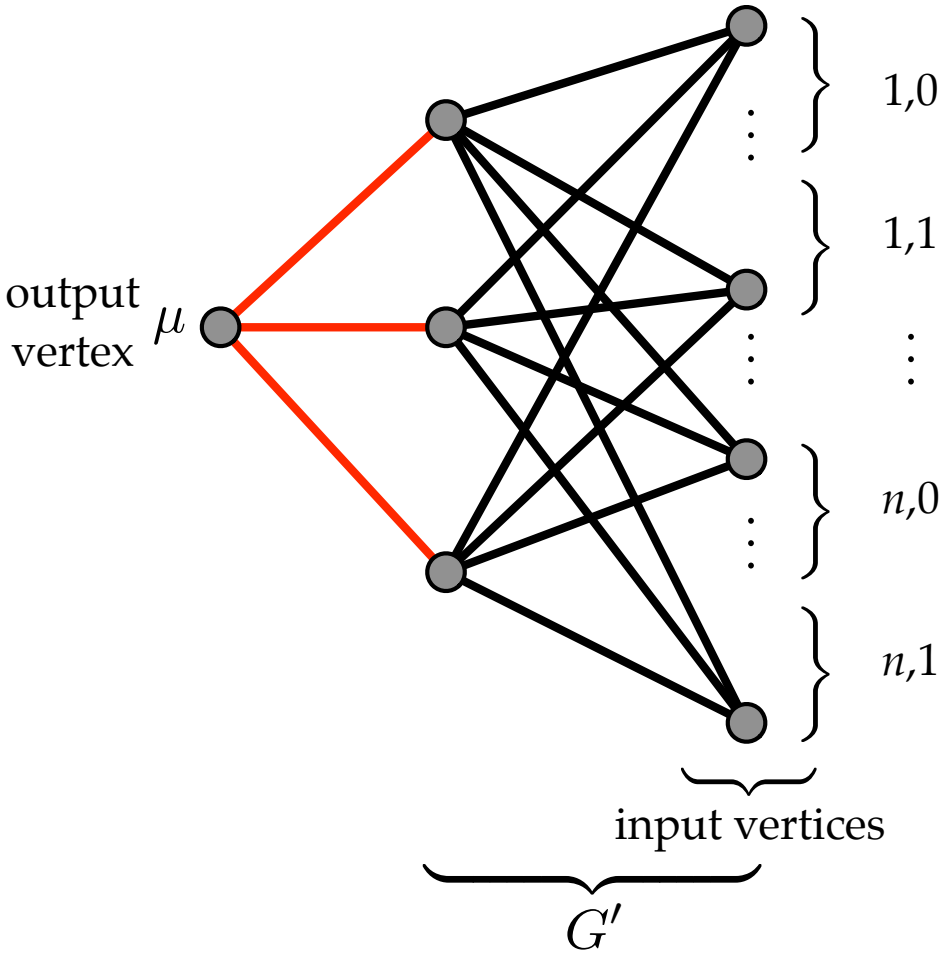
- **Theorem 2:** For any span program P, $Q(f_P) = O\left(\text{wsize}(P)\frac{\log \text{wsize}(P)}{\log\log \text{wsize}(P)}\right)$

1. Correspondence between P and bipartite graphs $G_P(x)$

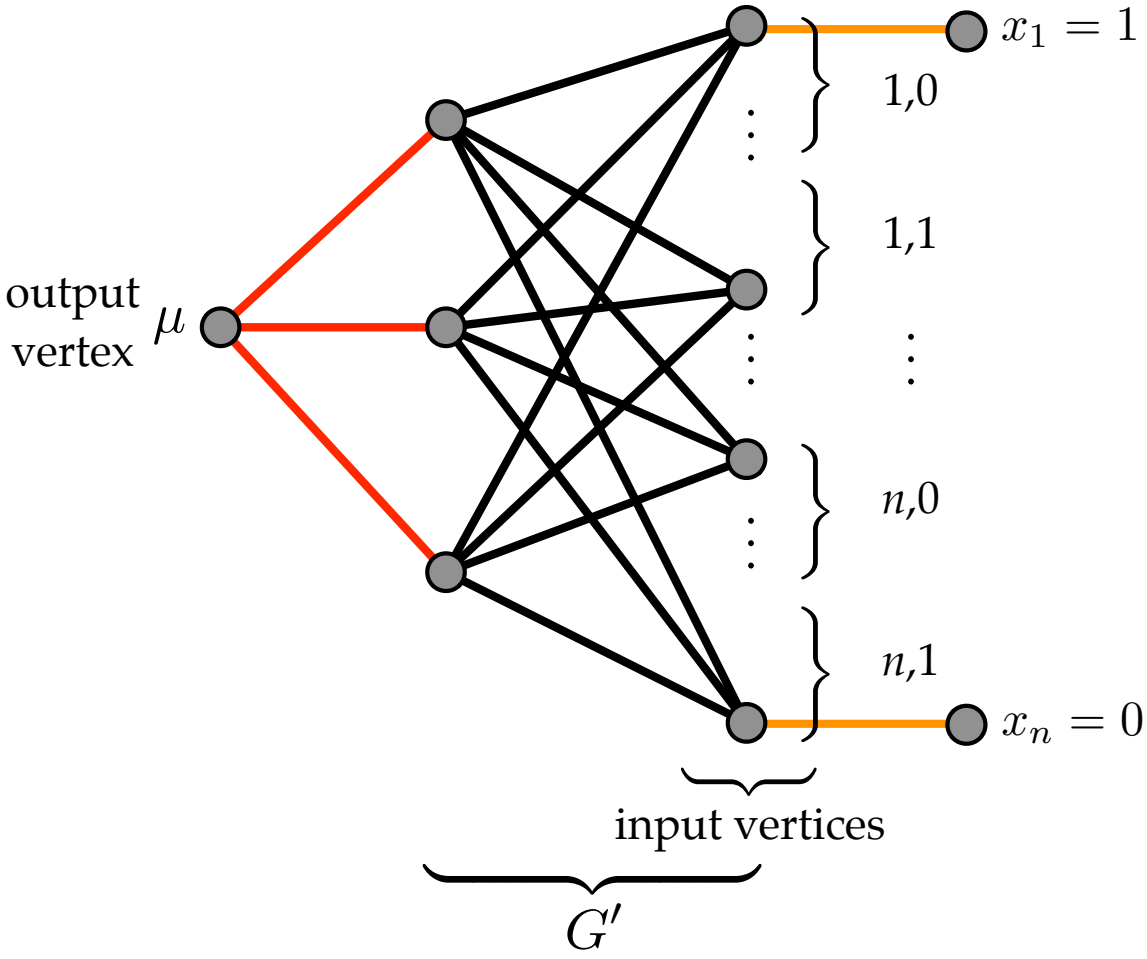
2. Eigenvalue-zero eigenvectors imply an "effective" spectral gap around zero

3. Quantum algorithm for detecting eigenvectors of structured graphs

- **Theorem:** Let G be a weighted bipartite graph.

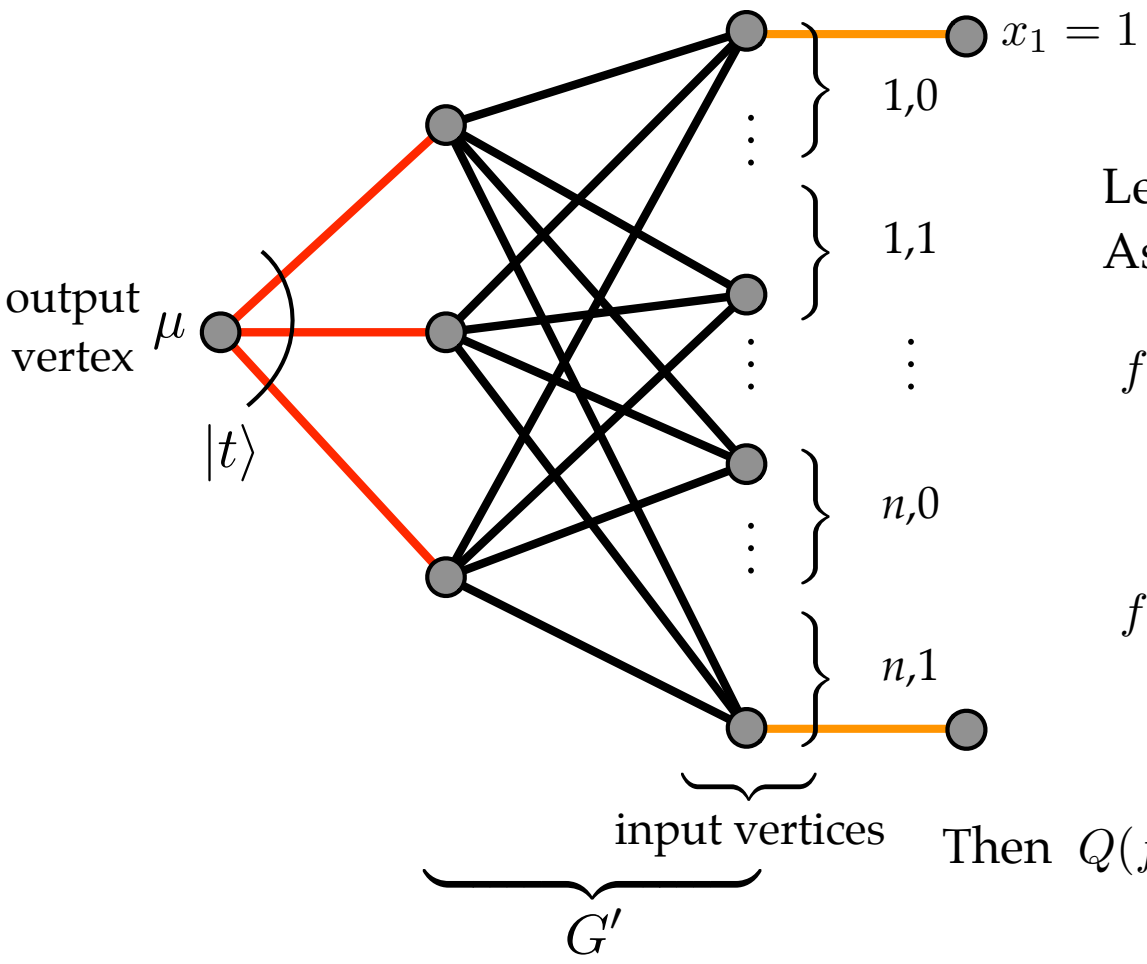


- **Theorem:** Let G be a weighted bipartite graph.



For an input x, add weight-one dangling edges to vertices in $\cup_j V_{j,\overline{x_j}}$ to define graphs G(x), G′(x).

- **Theorem:** Let G be a weighted bipartite graph.



For an input x, add weight-one dangling edges to vertices in $\cup_j V_{j,\overline{x_j}}$ to define graphs G(x), G′(x).

Let $f : \{0,1\}^n \to \{0,1\}$, $\delta > 0$.
Assume that for all x,

$f(x) = 1 \Rightarrow$ G(x) has an eigenvalue-zero eigenvector $|\psi\rangle$ with
$$|\langle\psi|\mu\rangle|^2 \geq \delta \||\psi\rangle\|^2$$

$f(x) = 0 \Rightarrow$ G′(x) has an eigenvalue-zero eigenvector $|\psi\rangle$ with
$$|\langle\psi|t\rangle|^2 \geq \delta \||\psi\rangle\|^2$$

Then $Q(f) = O\left(\min\left\{ \frac{\|\mathrm{abs}(A_G)\|}{\delta}, \frac{1}{\delta} \frac{\log \frac{1}{\delta}}{\log\log \frac{1}{\delta}} \right\} \right)$.

$$A_{G(x)} = \left(\begin{array}{cc|cc} & & & \text{output} \quad \text{inputs} \\ & 0 & |t\rangle & A \\ & & 0 & \mathbf{1} - \Pi(x) \\ \hline \langle t| & 0 & & \\ A^\dagger & \mathbf{1} - \Pi(x) & & 0 \end{array}\right)$$

Summary

- **Theorem 1:** For any boolean function f,

$$\inf_{P \text{ computing } f} \text{wsize}(P) = \text{Adv}^{\pm}(f)$$

- **Theorem 2:** For any span program P,

$$Q(f) = O\left(\text{wsize}(P)\frac{\log \text{wsize}(P)}{\log\log \text{wsize}(P)}\right)$$

Main corollaries

1. The general adversary bound is (almost) optimal for every total or partial function

$$f : \{0,1\}^n \to \{0,1\}^{\text{poly}(\log n)}$$

2. Span programs are (almost) equivalent to quantum query algorithms

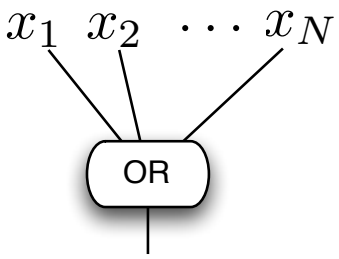
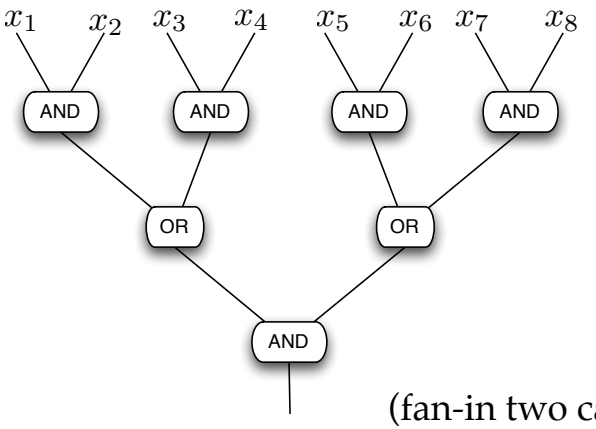
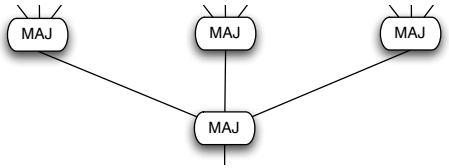
3. Optimal quantum algorithm for evaluating balanced formulas over *any finite* gate set

- **Theorem 1:** $\inf_{P \text{ computing } f} \mathrm{wsize}(P) = \mathrm{Adv}^{\pm}(f) = O(Q(f))$

- **Theorem 2:** If P computes f, $Q(f) = O\left(\mathrm{wsize}(P)\frac{\log \mathrm{wsize}(P)}{\log\log \mathrm{wsize}(P)}\right)$

- **Thm.** [RŠ '08]:
  $Q(f_P^k) = O\big(\mathrm{wsize}(P)^k\big)$
- **Thm.** [HLŠ '07, R '09]:
  $\mathrm{Adv}^{\pm}(f^k) = O\big(\mathrm{Adv}^{\pm}(f)^k\big)$

Using Theorem 2, implies optimal qu. algorithm for evaluating balanced formulas over *any finite* gate set

	Classical	Quantum
	$\Theta(n)$	$\Theta(\sqrt{n})$ [Grover '96]
Balanced AND-OR  (fan-in two case)	$\Theta(n^{0.753\ldots})$ [Snir '85] [Saks, Wigderson '86] [Santha '95]	$\Theta(\sqrt{n})$ [Farhi, Goldstone, Gutmann '07] [Ambainis, Childs, R, Špalek, Zhang '07]
General read-once AND-OR ⋮	$\Omega(n^{0.51})$ [Heiman, Wigderson '91]	$\Omega(\sqrt{n})$, [Barnum, Saks '04] $\sqrt{n} \cdot 2^{O(\sqrt{\log n})}$ [ACRŠZ '07]
Balanced MAJ_3 	$\Omega(2.333^d)$, $O(2.654^d)$ [Jayram, Kumar, Sivakumar '03]	$\Theta(2^d)$ ⋮ [RŠ '08]

	Classical	Quantum
OR_n (Search)	$\Theta(n)$	$\Theta(\sqrt{n})$
Balanced AND-OR	$\Theta(n^{0.753\ldots})$	$\Theta(\sqrt{n})$
General read-once AND-OR	$\Omega(n^{0.51})$	$\Omega(\sqrt{n}), O(\sqrt{n} \cdot \log n)$ [R '09]
Balanced MAJ_3 …	$\Omega(2.333^d), O(2.654^d)$	$\Theta(2^d)$
"Almost-balanced" formula over an arbitrary finite gate set	???	$\Theta(Adv\pm(f))$ [R '09]
Unbalanced formulas		Query complexity now understood, but not time-complexity

Recipe for finding optimal quantum query algorithms

- Find a solution to: $\mathrm{Adv}^{\pm}(f) = \min_{\substack{\{X_j \succeq 0\}: \\ \forall (x,y)\in\Delta, \sum_{j:x_j\neq y_j}\langle x|X_j|y\rangle = 1}} \max_x \sum_j \langle x|X_j|x\rangle \quad (*)$

- Take the Cholesky decomposition: $\{|v_{xj}\rangle\} : \langle v_{xj}|v_{yj}\rangle = \langle x|X_j|y\rangle$

- Use the entries of the vectors to weight the edges of a graph, and run phase estimation on the quantum walk…

$$B_G = \left(\begin{array}{c|c} 1 & \\ \vdots & \sum_{x:f(x)=0} \sum_{j=1}^{n} |x\rangle\langle \overline{x_j}| \otimes \langle\, v_{xj}| \\ 1 & \end{array} \right)$$

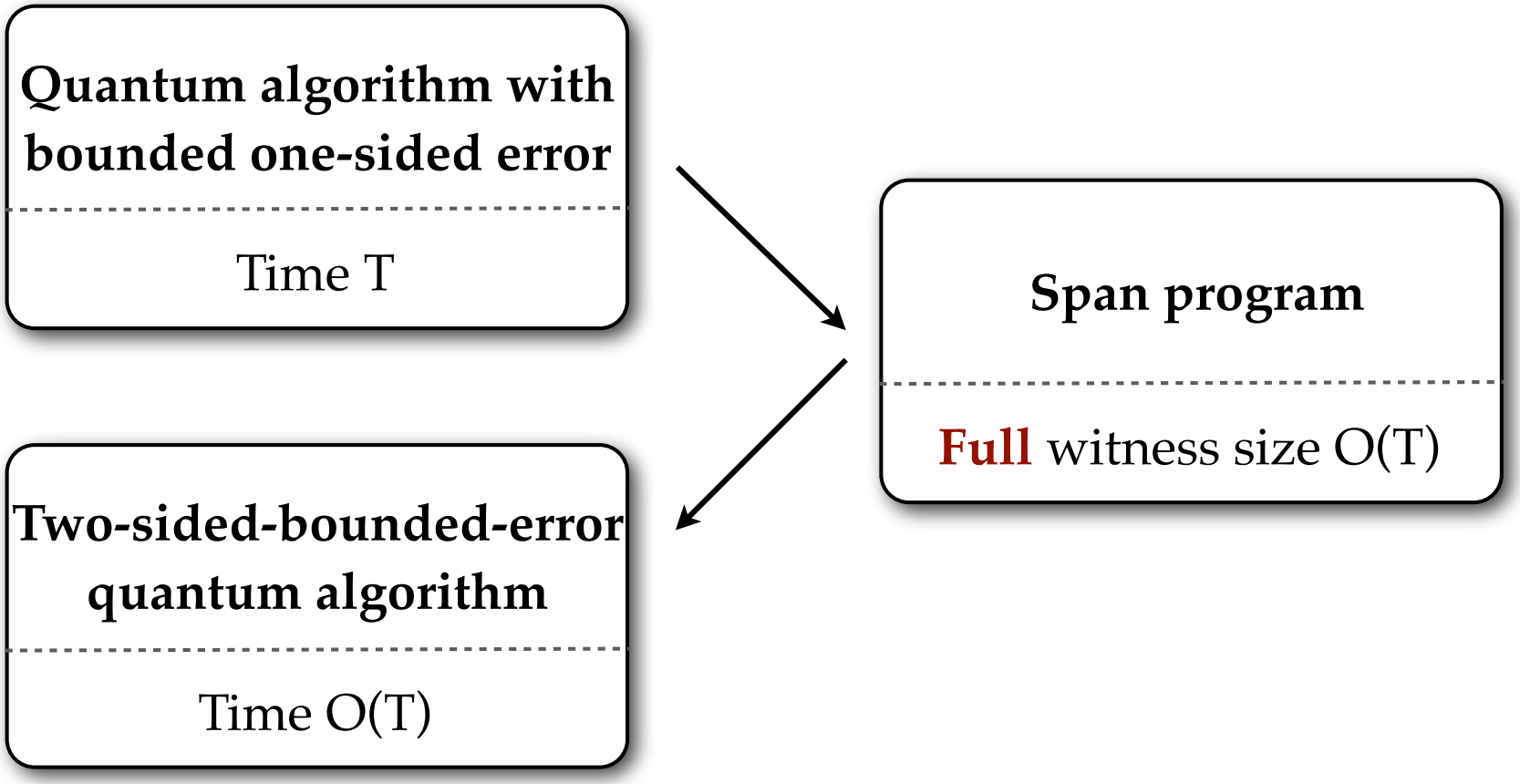
- But how can we find good solutions to $(*)$?

Open problems (1)

- Functions with non-binary domains? $f : \{1, 2, \ldots, k\}^n \to \{0, 1\}$
  - Formulas over non-boolean gates? Composition of the adversary bound?
- Can the logarithmic overhead be removed?
- Is there a good *classical* algorithm for evaluating span programs?
  - Our results apply to both total and *partial* functions, though.
- Robust evaluation of span programs? See [Høyer, Mosca, de Wolf '03].

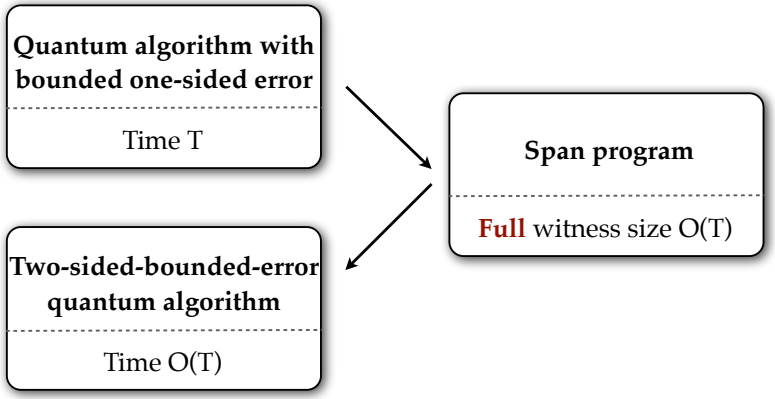
Open problems (2)

- Relationship of span programs to quantum algorithms, under *time* complexity, instead of query complexity?
  - Time-efficient algorithms can be based on sparse span programs with constant norm, and small "full witness size" [0904.2759, 0907.1622].



Open problems (3)

- Relationship of span programs to quantum algorithms, under *time* complexity, instead of query complexity?
  - Time-efficient algorithms can be based on sparse span programs with constant norm, and small "full witness size" [0904.2759, 0907.1622].



★ More explicit and *time-efficient* algorithms.

- So far: Almost-balanced formulas over arbitrary finite gate sets [0907.1622], arbitrary AND-OR formulas [0907.1623].
- Solve the $\mathrm{Adv}^{\pm}$ dual SDP.
- Find nontrivial rederivations of known algorithms.
- Take advantage of the optimal algorithm's time-independent, greedy structure?