Evaluating formulas on a quantum computer

Andrew Childs (Waterloo)
Andris Ambainis (Waterloo & Latvia)
Ben Reichardt (Caltech)
Robert Špalek (Google)
Shengyu Zhang (Caltech)

quant-ph/0703015
FOCS 2007
How fast can we compute the OR of \( n \) bits?

Evaluate formula: \( x_1 \ OR \ x_2 \ OR \ \ldots \ OR \ x_n \)
How fast can we compute the OR of $n$ bits?

Evaluate formula: $x_1 \text{ OR } x_2 \text{ OR } \ldots \text{ OR } x_n$

Applications:
• unstructured search
• fundamental building block for other computations
OR

How fast can we compute the OR of \( n \) bits?

Evaluate formula: \( x_1 \text{ OR } x_2 \text{ OR } \ldots \text{ OR } x_n \)

Applications:
- unstructured search
- fundamental building block for other computations

Model: Given a black box for the bits.
How fast can we compute the OR of \( n \) bits?

Evaluate formula: \( x_1 \text{ OR } x_2 \text{ OR } \ldots \text{ OR } x_n \)

Applications:
- unstructured search
- fundamental building block for other computations

Model: Given a black box for the bits.
How fast can we compute the OR of $n$ bits?

Evaluate formula: $x_1 \text{ OR } x_2 \text{ OR } \ldots \text{ OR } x_n$

Applications:
- unstructured search
- fundamental building block for other computations

Model: Given a black box for the bits.
How fast can we compute the OR of $n$ bits?

Evaluate formula: $x_1 \ OR \ x_2 \ OR \ldots \ OR \ x_n$

Applications:
- unstructured search
- fundamental building block for other computations

Model: Given a black box for the bits.
How fast can we compute the OR of $n$ bits?

Evaluate formula: $x_1 \lor x_2 \lor \ldots \lor x_n$

Applications:
- unstructured search
- fundamental building block for other computations

Model: Given a black box for the bits.
How fast can we compute the OR of $n$ bits?

Evaluate formula: $x_1 \text{ OR } x_2 \text{ OR } \ldots \text{ OR } x_n$

Applications:
- unstructured search
- fundamental building block for other computations

Model: Given a black box for the bits.
**OR**

How fast can we compute the OR of $n$ bits?

Evaluate formula: $x_1 \text{ OR } x_2 \text{ OR } \ldots \text{ OR } x_n$

Applications:
- unstructured search
- fundamental building block for other computations

Model: Given a black box for the bits.

\[
i \quad |i, x \rangle \quad \quad \quad \quad \quad \quad \quad |i \oplus x \rangle
\]

How many queries are required to evaluate OR?

Classical complexity: $\Theta(N)$

Quantum algorithm [Grover 1996]: $O(\sqrt{N})$

Quantum lower bound [BBBV 1996]: $\Omega(\sqrt{N})$
The query complexity of two-player games
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which
• player 0 goes first
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which
  • player 0 goes first
  • players alternate moves
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which

• player 0 goes first
• players alternate moves
• each player has $d$ possible moves during their turn
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which
  • player 0 goes first
  • players alternate moves
  • each player has $d$ possible moves during their turn
  • there are a total of $k$ turns
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which
- player 0 goes first
- players alternate moves
- each player has $d$ possible moves during their turn
- there are a total of $k$ turns
- the winner after any given sequence of moves ($n = d^k$ possibilities) is given by a black box function $f: \{0, 1, \ldots, d\}^k \rightarrow \{0, 1\}$
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which

• player 0 goes first
• players alternate moves
• each player has $d$ possible moves during their turn
• there are a total of $k$ turns
• the winner after any given sequence of moves ($n = d^k$ possibilities)
  is given by a black box function $f: \{0, 1, \ldots, d\}^k \rightarrow \{0, 1\}$

How many queries must we make to determine who wins the game (assuming the players play optimally)?
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which

• player 0 goes first
• players alternate moves
• each player has $d$ possible moves during their turn
• there are a total of $k$ turns
• the winner after any given sequence of moves ($n = d^k$ possibilities)
  is given by a black box function $f: \{0, 1, \ldots, d\}^k \rightarrow \{0, 1\}$

How many queries must we make to determine who wins the game
(assuming the players play optimally)?

We must evaluate a formula involving AND and OR gates:
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which

• player 0 goes first
• players alternate moves
• each player has $d$ possible moves during their turn
• there are a total of $k$ turns
• the winner after any given sequence of moves ($n = d^k$ possibilities)
is given by a black box function $f$: \{0, 1, ..., $d$\}$^k$ → \{0, 1\}

How many queries must we make to determine who wins the game (assuming the players play optimally)?

We must evaluate a formula involving AND and OR gates:

1-player wins if he can make any move that gives 1 (OR)
The query complexity of two-player games

Consider a two-player game (players ‘0’, ‘1’) in which

- player 0 goes first
- players alternate moves
- each player has \(d\) possible moves during their turn
- there are a total of \(k\) turns
- the winner after any given sequence of moves \((n = d^k\) possibilities\) is given by a black box function \(f: \{0, 1, \ldots, d\}^k \rightarrow \{0, 1\}\)

How many queries must we make to determine who wins the game (assuming the players play optimally)?

We must evaluate a formula involving \textbf{AND} and \textbf{OR} gates:

- 0-player wins if she can make any move that gives 0
  i.e., she only loses if all of her moves give 1 (\textbf{AND})
- 1-player wins if he can make any move that gives 1 (\textbf{OR})
Game trees

Example: $d = 2, k = 4$
Game trees

Example: \( d = 2, k = 4 \)
Game trees

Example: $d = 2, k = 4$
Example: $d = 2, k = 4$
Game trees

Example: $d = 2, k = 4$
Game trees

Example: $d = 2, k = 4$
Evaluating game trees

Classical complexity: $\Theta\left(n^{\log_d \frac{d-1+\sqrt{d^2+14d+1}}{4}}\right)$  
$d = 2: \Theta(n^{0.753})$

[Snir 85; Saks, Wigderson 86; Santha 95]
Evaluating game trees

Classical complexity: \(\Theta\left(n^{\log_d \left(\frac{d-1+\sqrt{d^2+14d+1}}{4}\right)}\right)\)  (\(d = 2: \Theta(n^{0.753})\))

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

Quantum lower bound [Barnum, Saks 02]: \(\Omega(\sqrt{n})\)
Evaluating game trees

Classical complexity: $\Theta\left(n^{\log_d \frac{d-1+\sqrt{d^2+14d+1}}{4}}\right)$  
(d = 2: $\Theta(n^{0.753})$)  
[Snir 85; Saks, Wigderson 86; Santha 95]

Quantum lower bound [Barnum, Saks 02]: $\Omega(\sqrt{n})$

Recursive Grover [Buhrman, Cleve, Wigderson 98]: $\sqrt{n} O(\log n)^{k-1}$
Evaluating game trees

Classical complexity: $\Theta\left(n^{\log_d \frac{d-1+\sqrt{d^2+14d+1}}{4}}\right)$ \quad (d = 2: $\Theta(n^{0.753})$)

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

Quantum lower bound [Barnum, Saks 02]: $\Omega(\sqrt{n})$

Recursive Grover [Buhrman, Cleve, Wigderson 98]: $\sqrt{n} \cdot O(\log n)^{k-1}$

Grover with noisy inputs [Høyer, Mosca, de Wolf 03]: $O(\sqrt{n} \cdot c^k)$
Evaluating game trees

Classical complexity: \( \Theta\left(n^{\log_d \frac{d-1 + \sqrt{d^2 + 14d + 1}}{4}}\right) \)  
\( (d = 2: \Theta(n^{0.753})) \quad [\text{Snir 85; Saks, Wigderson 86; Santha 95}] \)

Quantum lower bound [Barnum, Saks 02]: \( \Omega(\sqrt{n}) \)

Recursive Grover [Buhrman, Cleve, Wigderson 98]: \( \sqrt{n} \cdot O(\log n)^{k - 1} \)

Grover with noisy inputs [Høyer, Mosca, de Wolf 03]: \( O(\sqrt{n} \cdot c^k) \)

But these algorithms are only close to tight for \( k \) constant.

And for low degree (e.g., \( d = 2 \)), nothing better than classical was known until very recently!
Continuous-time quantum walk

Graph $G$: 

![Graph](image-url)
Continuous-time quantum walk

Graph $G$: 

$A = \begin{pmatrix}
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 \\
\end{pmatrix}$

adjacency matrix
Continuous-time quantum walk

Graph $G$:

$A = \begin{pmatrix}
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 \\
\end{pmatrix}$

$L = \begin{pmatrix}
-2 & 1 & 1 & 0 & 0 \\
1 & -3 & 0 & 1 & 1 \\
1 & 0 & -2 & 1 & 0 \\
0 & 1 & 1 & -3 & 1 \\
0 & 1 & 0 & 1 & -2 \\
\end{pmatrix}$

adjacency matrix

Laplacian
Continuous-time quantum walk

Graph $G$:

Random walk on $G$

State: Probability $p_j(t)$ of being at vertex $j$ at time $t$

Dynamics: $\frac{d}{dt} \vec{p} = -\gamma L \vec{p}$
Continuous-time quantum walk

Graph $G$:

State: Probability $p_j(t)$ of being at vertex $j$ at time $t$

Dynamics: \[ \frac{d}{dt} \vec{p} = -\gamma \vec{L} \vec{p} \]

Random walk on $G$

State: Probability $p_j(t)$ of being at vertex $j$ at time $t$

Dynamics: \[ \frac{d}{dt} \vec{p} = -\gamma \vec{L} \vec{p} \]

Quantum walk on $G$

State: Amplitude $q_j(t)$ to be at vertex $j$ at time $t$

Dynamics: \[ i \frac{d}{dt} \vec{q} = -\gamma \vec{L} \vec{q} \]

Adjacency matrix $A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$

Laplacian $L = \begin{pmatrix} -2 & 1 & 1 & 0 & 0 \\ 1 & -3 & 0 & 1 & 1 \\ 1 & 0 & -2 & 1 & 0 \\ 0 & 1 & 1 & -3 & 1 \\ 0 & 1 & 0 & 1 & -2 \end{pmatrix}$
Continuous-time quantum walk

Graph $G$:

\[
A = \begin{pmatrix}
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{pmatrix}
\]

\[
L = \begin{pmatrix}
-2 & 1 & 1 & 0 & 0 \\
1 & -3 & 0 & 1 & 1 \\
1 & 0 & -2 & 1 & 0 \\
0 & 1 & 1 & -3 & 1 \\
0 & 1 & 0 & 1 & -2
\end{pmatrix}
\]

Random walk on $G$

State: Probability $p_j(t)$ of being at vertex $j$ at time $t$

Dynamics:

\[
dt \vec{p} = -\gamma L \vec{p}
\]

Quantum walk on $G$

State: Amplitude $q_j(t)$ to be at vertex $j$ at time $t$

Dynamics:

\[
dt \vec{q} = -\gamma L \vec{q} \quad \text{(or} \quad dt \vec{q} = \gamma A \vec{q}, \text{or} \ldots\text{)}
\]
Evaluating AND-OR trees by scattering

[Farhi, Goldstone, Gutmann 07]
Evaluating AND-OR trees by scattering

[Farhi, Goldstone, Gutmann 07]
Evaluating AND-OR trees by scattering

[Farhi, Goldstone, Gutmann 07]
Evaluating AND-OR trees by scattering

[Farhi, Goldstone, Gutmann 07]
Claim: For small $k$, the wave is transmitted if the formula (translated into NAND gates) evaluates to 0, and reflected if it evaluates to 1.
Gate sets

{AND, OR, NOT} equivalent to {NAND}
Gate sets

\{\text{AND, OR, NOT}\} equivalent to \{\text{NAND}\}
Gate sets

\{\text{AND, OR, NOT}\} equivalent to \{\text{NAND}\}
Gate sets

{AND, OR, NOT} equivalent to {NAND}

\[
\begin{align*}
\text{not} & \quad = \quad \text{nand} \\
\quad x & \quad = \quad \quad x \\
\text{and} & \quad = \quad \text{nand} \\
\quad \quad x_1 & \quad \quad x_2 \\
\text{or} & \quad = \quad \text{nand} \quad \text{nand} \\
\quad \quad x_1 & \quad \quad x_2 \\
\end{align*}
\]
In a *formula* (instead of a *circuit*), the fanout of every gate is 1 (so the graph is a tree).
In a *formula* (instead of a *circuit*), the fanout of every gate is 1 (so the graph is a tree).
In a formula (instead of a circuit), the fanout of every gate is 1 (so the graph is a tree).
In a formula (instead of a circuit), the fanout of every gate is 1 (so the graph is a tree).

Consider read-once formulas: every leaf is a different input. (Equivalently, count duplicated inputs with multiplicity.)
General formulas

In a formula (instead of a circuit), the fanout of every gate is 1 (so the graph is a tree).

Consider read-once formulas: every leaf is a different input. (Equivalently, count duplicated inputs with multiplicity.)

Quantum lower bound [Barnum-Saks 02]: $\Omega(\sqrt{n})$
Results

[Farhi, Goldstone, Gutmann 07] + [C., Cleve, Jordan, Yeung 07]

• $\sqrt{n^{1+o(1)}}$ time (and query) quantum algorithm for evaluating the balanced, binary NAND formula with $n$ inputs
Results

[Farhi, Goldstone, Gutmann 07] + [C., Cleve, Jordan, Yeung 07]

- $\sqrt{n^{1+o(1)}}$ time (and query) quantum algorithm for evaluating the balanced, binary NAND formula with $n$ inputs

Conjecture [Laplante, Lee, Szegedy 05]: Formula size is lower bounded by the square of the bounded-error quantum query complexity.
Results

[Farhi, Goldstone, Gutmann 07] + [C., Cleve, Jordan, Yeung 07]

- $\sqrt{n^{1+o(1)}}$ time (and query) quantum algorithm for evaluating the balanced, binary NAND formula with $n$ inputs

Conjecture [Laplante, Lee, Szegedy 05]: Formula size is lower bounded by the square of the bounded-error quantum query complexity.

This talk:

- $O(\sqrt{n})$ query quantum algorithm for evaluating “approximately balanced” NAND formulas (optimal!)
Results

[Farhi, Goldstone, Gutmann 07] + [C., Cleve, Jordan, Yeung 07]

- $\sqrt{n^{1+o(1)}}$ time (and query) quantum algorithm for evaluating the balanced, binary NAND formula with $n$ inputs

Conjecture [Laplante, Lee, Szegedy 05]: Formula size is lower bounded by the square of the bounded-error quantum query complexity.

This talk:

- $O(\sqrt{n})$ query quantum algorithm for evaluating “approximately balanced” NAND formulas (optimal!)
- $\sqrt{n^{1+o(1)}}$ time (and query) quantum algorithm for evaluating arbitrary NAND formulas
The algorithm

1. Start at the root of the tree
2. Perform phase estimation with precision $\approx 1/\sqrt{n}$ on a discrete-time quantum walk on the tree
3. If the estimated phase is 0 or $\pi$, then output 1; otherwise output 0
The algorithm

1. Start at the root of the tree
2. Perform phase estimation with precision $\approx 1/\sqrt{n}$ on a discrete-time quantum walk on the tree
3. If the estimated phase is 0 or $\pi$, then output 1; otherwise output 0

Outline

- Scattering $\rightarrow$ phase estimation
- Hamiltonian for a continuous-time quantum walk (with non-uniform edge weights)
- Low-energy eigenstates “compute NAND”
- Continuous time $\rightarrow$ discrete time (gives a small speedup)
- Formula rebalancing
From scattering to phase estimation

To do scattering calculations, we compute a complete basis of eigenstates:

Left: \( e^{ikx} + R(k) e^{-ikx} \)  
Right: \( \bar{T}(k) e^{-ikx} \)  
Bound: \( e^{\kappa x} \)

\[ G \]

\[ e^{ikx} + R(k) e^{-ikx} \]
\[ \bar{T}(k) e^{-ikx} \]
\[ e^{\kappa x} \]
From scattering to phase estimation

To do scattering calculations, we compute a complete basis of eigenstates:

Left: \( e^{ikx} + R(k) e^{-ikx} \)

Right: \( \bar{T}(k) e^{-ikx} \)

Bound: \( e^{\kappa x} \)

Instead, we can just look at eigenstates of the graph itself.

Phase estimation: Given \( U \) and an eigenstate \( |\varphi\rangle \) with \( U |\varphi\rangle = e^{i\varphi} |\varphi\rangle \), we can estimate \( \varphi \) to precision \( \delta \) in \( O(1/\delta) \) steps. (Equivalent to measuring \( H = i \log U \).)
The Hamiltonian

Graph: Tree representing the NAND formula, with edges added to 1 inputs (so that all leaves evaluate to 0).
The Hamiltonian

Graph: Tree representing the NAND formula, with edges added to 1 inputs (so that all leaves evaluate to 0).

$$H|v\rangle = h_{pv}|p\rangle + \sum_c h_{vc}|c\rangle$$
The Hamiltonian

**Graph:** Tree representing the NAND formula, with edges added to 1 inputs (so that all leaves evaluate to 0).

\[ H |v\rangle = h_{pv} |p\rangle + \sum_c h_{vc} |c\rangle \]

**Edge weights:** \( h_{pv} \approx 4 \sqrt{\frac{s_v}{s_p}} \)  
\( s_v = \# \text{ of inputs in subformula under } v \)
The Hamiltonian

**Graph:** Tree representing the NAND formula, with edges added to 1 inputs (so that all leaves evaluate to 0).

\[ H |v\rangle = h_{pv} |p\rangle + \sum_c h_{vc} |c\rangle \]

**Edge weights:** \( h_{pv} \approx 4 \sqrt{\frac{s_v}{s_p}} \) \( s_v = \# \text{ of inputs in subformula under } v \)

(Also, add two NOT gates to the root and use different weights there.)
The Hamiltonian

Graph: Tree representing the NAND formula, with edges added to 1 inputs (so that all leaves evaluate to 0).

\[ H|v\rangle = h_{pv}|p\rangle + \sum_c h_{vc}|c\rangle \]

Edge weights: \( h_{pv} \approx 4 \frac{s_v}{s_p} \quad s_v = \# \text{ of inputs in subformula under } v \)
(Also, add two NOT gates to the root and use different weights there.)

Eigenstates: \( H|E\rangle = E|E\rangle \)
The Hamiltonian

Graph: Tree representing the NAND formula, with edges added to 1 inputs (so that all leaves evaluate to 0).

\[
H |v\rangle = h_{pv} |p\rangle + \sum_c h_{vc} |c\rangle
\]

Edge weights: \( h_{pv} \approx 4 \sqrt{\frac{s_v}{s_p}} \) \( s_v = \# \) of inputs in subformula under \( v \)

(Also, add two NOT gates to the root and use different weights there.)

Eigenstates: \( H |E\rangle = E |E\rangle \) \( h_{pv} \langle p |E\rangle + \sum_c h_{vc} \langle c |E\rangle = E \langle v |E\rangle \)
The Hamiltonian

Graph: Tree representing the NAND formula, with edges added to 1 inputs (so that all leaves evaluate to 0).

\[ H|v\rangle = h_{pv}|p\rangle + \sum_c h_{vc}|c\rangle \]

Edge weights: \( h_{pv} \approx 4\sqrt{s_v/s_p} \) \( s_v = \# \) of inputs in subformula under \( v \)

(Also, add two NOT gates to the root and use different weights there.)

Eigenstates: \( H|E\rangle = E|E\rangle \) \( h_{pv}\langle p|E\rangle + \sum_c h_{vc}\langle c|E\rangle = E\langle v|E\rangle \)

For \( E = 0 \): \( \langle p|\Psi\rangle = -\sum_c \frac{h_{vc}}{h_{pv}} \langle c|\Psi\rangle \)
Zero-energy eigenstates evaluate NAND: Qualitative version

Let $\text{NAND}(p)$ denote the value of the NAND subformula under $p$. Let $r = \text{root of the tree}$. 
Zero-energy eigenstates evaluate NAND: Qualitative version

Let \( \text{NAND}(p) \) denote the value of the NAND subformula under \( p \).
Let \( r = \text{root of the tree} \).

**Theorem.**

If \( \text{NAND}(p) = 1 \), then \( \langle p | \Psi \rangle = 0 \) for any \( | \Psi \rangle \) with \( H | \Psi \rangle = 0 \).
If \( \text{NAND}(r) = 0 \), then \( | \langle r | \Psi \rangle | > 0 \) for some \( | \Psi \rangle \) with \( H | \Psi \rangle = 0 \).
\( \text{NAND} = 1 \)

If \( \text{NAND}(p) = 1 \), then \( \langle p | \Psi \rangle = 0 \) for any \( | \Psi \rangle \) with \( H | \Psi \rangle = 0 \).
If $\text{NAND}(p) = 1$, then $\langle p | \Psi \rangle = 0$ for any $| \Psi \rangle$ with $H | \Psi \rangle = 0$.

**Base case:** Some child $v$ of $p$ is a leaf.

\[ h_{pv} \langle p | \Psi \rangle = 0 \]
**NAND = 1**

If $\text{NAND}(p) = 1$, then $\langle p|\Psi \rangle = 0$ for any $|\Psi \rangle$ with $H|\Psi \rangle = 0$.

**Base case:** Some child $v$ of $p$ is a leaf.

$$h_{pv}\langle p|\Psi \rangle = 0$$

**Induction:** Some child $v$ of $p$ has $\text{NAND}(v) = 0$; all its children $c$ have $\text{NAND}(c) = 1$. 
If \( \text{NAND}(p) = 1 \), then \( \langle p | \Psi \rangle = 0 \) for any \( |\Psi\rangle \) with \( H |\Psi\rangle = 0 \).

**Base case:** Some child \( v \) of \( p \) is a leaf.

\[
h_{pv} \langle p | \Psi \rangle = 0
\]

**Induction:** Some child \( v \) of \( p \) has \( \text{NAND}(v) = 0 \); all its children \( c \) have \( \text{NAND}(c) = 1 \).

\[
h_{pv} \langle p | \Psi \rangle = - \sum_c h_{vc} \langle c | \Psi \rangle = 0
\]
If \( \text{NAND}(r) = 0 \), then \( \langle r | \Psi \rangle > 0 \) for some \( |\Psi\rangle \) with \( H |\Psi\rangle = 0 \).
If $\text{NAND}(r) = 0$, then $|\langle r | \Psi \rangle| > 0$ for some $|\Psi\rangle$ with $H |\Psi\rangle = 0$.

Base case: A single leaf.

\[
\begin{array}{c}
\begin{array}{c}
 r \\
 0
\end{array}
\end{array}
\]

$|\Psi\rangle = |r\rangle$
If \( \text{NAND}(r) = 0 \), then \( |\langle r|\Psi\rangle| > 0 \) for some \( |\Psi\rangle \) with \( H|\Psi\rangle = 0 \).

**Base case:** A single leaf.

\[
\begin{array}{c}
\text{r}\ 0 \\
\end{array}
\]

\( |\Psi\rangle = |r\rangle \)

**Induction:** All children \( v \) of \( r \) have \( \text{NAND}(v) = 1 \);

some child \( c \) of \( v \) has \( \text{NAND}(c) = 0 \).
**NAND = 0**

If $\text{NAND}(r) = 0$, then $|\langle r | \Psi \rangle| > 0$ for some $|\Psi\rangle$ with $H |\Psi\rangle = 0$.

**Base case:** A single leaf.

\[
\begin{array}{c}
\begin{array}{c}
\text{r} \\
0
\end{array}
\end{array}
\]

$|\Psi\rangle = |r\rangle$

**Induction:** All children $v$ of $r$ have $\text{NAND}(v) = 1$; some child $c$ of $v$ has $\text{NAND}(c) = 0$.

put a state here with $|\langle c | \Psi \rangle| > 0$
\( \text{NAND} = 0 \)

If \( \text{NAND}(r) = 0 \), then \( \langle r | \Psi \rangle > 0 \) for some \( |\Psi\rangle \) with \( H |\Psi\rangle = 0 \).

**Base case:** A single leaf.

\[
\begin{align*}
    r & \quad 0 \\
    |\Psi\rangle & = |r\rangle
\end{align*}
\]

**Induction:** All children \( v \) of \( r \) have \( \text{NAND}(v) = 1 \);
some child \( c \) of \( v \) has \( \text{NAND}(c) = 0 \).

\[
    h_{rv} \langle r | \Psi \rangle = -h_{vc} \langle c | \Psi \rangle \neq 0
\]

put a state here with \( |\langle c | \Psi \rangle| > 0 \)
Zero-energy eigenstates evaluate NAND: 
Quantitative version

Theorem (qualitative).
If \( \text{NAND}(p) = 1 \), then \( \langle p | \Psi \rangle = 0 \) for any \( |\Psi\rangle \) with \( H |\Psi\rangle = 0 \).
If \( \text{NAND}(r) = 0 \), then \( |\langle r | \Psi \rangle| > 0 \) for some \( |\Psi\rangle \) with \( H |\Psi\rangle = 0 \).

Theorem (quantitative). For approximately balanced formulas:
If \( \text{NAND}(r) = 1 \), then eigenstates \( |E\rangle \) with \( |E| < O\left(\frac{1}{\sqrt{n}}\right) \)
have \( \langle r | E \rangle = 0 \).
If \( \text{NAND}(r) = 0 \), then \( |\langle r | \Psi \rangle| > \Omega(1) \) for some \( |\Psi\rangle \) with \( H |\Psi\rangle = 0 \).
Simulating the quantum walk

We could perform phase estimation directly on the dynamics of this Hamiltonian (i.e., measure the energy).

But this would require simulating the dynamics by a sequence of quantum gates, using the black box to simulate the walk near the leaves, and combining that simulation with the input-independent part.

[C., Cleve, Jordan, Yeung 07]
Simulating the quantum walk

We could perform phase estimation directly on the dynamics of this Hamiltonian (i.e., measure the energy).

But this would require simulating the dynamics by a sequence of quantum gates, using the black box to simulate the walk near the leaves, and combining that simulation with the input-independent part.

\[ e^{i(A+B)} \approx (e^{iA/m} e^{iB/m})^m \]
Simulating the quantum walk

We could perform phase estimation directly on the dynamics of this Hamiltonian (i.e., measure the energy).

But this would require *simulating* the dynamics by a sequence of quantum gates, using the black box to simulate the walk near the leaves, and combining that simulation with the input-independent part.

\[ e^{i(A+B)} \approx (e^{iA/m}e^{iB/m})^m \]

simulation steps (run time)^2
Simulating the quantum walk

We could perform phase estimation directly on the dynamics of this Hamiltonian (i.e., measure the energy).

But this would require simulating the dynamics by a sequence of quantum gates, using the black box to simulate the walk near the leaves, and combining that simulation with the input-independent part.

\[ e^{i(A+B)} \approx (e^{iA/m} e^{iB/m})^m \]
\[ \approx (e^{iA/2m} e^{iB/m} e^{iA/2m})^m \]

Simulation steps

(\text{run time})^2

(\text{run time})^{3/2}
Simulating the quantum walk

We could perform phase estimation directly on the dynamics of this Hamiltonian (i.e., measure the energy).

But this would require simulating the dynamics by a sequence of quantum gates, using the black box to simulate the walk near the leaves, and combining that simulation with the input-independent part.

\[
e^{i(A+B)} \approx (e^{iA/m} e^{iB/m})^m \\
\approx (e^{iA/2m} e^{iB/m} e^{iA/2m})^m \\
\vdots \\
\approx (e^{iA/2m} e^{iB/m} e^{iA/2m})^m \\
\vdots \\
\approx (\text{run time})^{1+o(1)}
\]

\[
\approx (\text{run time})^{2} \\
\approx (\text{run time})^{3/2} \\
\vdots \\
\approx (\text{run time})^{1+o(1)}
\]
Simulating the quantum walk

[C., Cleve, Jordan, Yeung 07]

We could perform phase estimation directly on the dynamics of this Hamiltonian (i.e., measure the energy).

But this would require simulating the dynamics by a sequence of quantum gates, using the black box to simulate the walk near the leaves, and combining that simulation with the input-independent part.

\[ e^{i(A+B)} \approx (e^{iA/m} e^{iB/m})^m \]
\[ \approx (e^{iA/2m} e^{iB/m} e^{iA/2m})^m \]
\[ \vdots \]
\[ \approx (e^{iA/2m} e^{iB/m} e^{iA/2m})^m \]

Instead, we can avoid the \( o(1) \) by using a discrete-time quantum walk.
Continuous time $\rightarrow$ discrete time

Szegedy quantization of classical Markov chains:
Continuous time $\rightarrow$ discrete time

Szegedy quantization of classical Markov chains:

- Classical random walk
- Stochastic matrix $P$
Continuous time $\rightarrow$ discrete time

Szegedy quantization of classical Markov chains:

- Classical random walk
  - Stochastic matrix $P$
- Quantum walk
  - Unitary operator $U$ derived from $P$
  - (locality of $P \rightarrow$ locality of $U$)
Continuous time $\rightarrow$ discrete time

Szegedy quantization of classical Markov chains:

**Classical random walk**
- Stochastic matrix $P$
- Eigenvalues of $\sqrt{P \circ P^T}$: $\lambda_j$

**Quantum walk**
- Unitary operator $U$ derived from $P$
- (locality of $P \rightarrow$ locality of $U$)
Continuous time → discrete time

Szegedy quantization of classical Markov chains:

<table>
<thead>
<tr>
<th>Classical random walk</th>
<th>Quantum walk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stochastic matrix $P$</td>
<td>Unitary operator $U$ derived from $P$ (locality of $P$ → locality of $U$)</td>
</tr>
<tr>
<td>Eigenvalues of $\sqrt{P \circ P^T}$: $\lambda_j$</td>
<td>Eigenvalues of $U$: $e^{\pm i \arcsin \lambda_j}$</td>
</tr>
</tbody>
</table>
Continuous time $\rightarrow$ discrete time

Szegedy quantization of classical Markov chains:

Classical random walk

Stochastic matrix $P$

Eigenvalues of $\sqrt{P \circ P^T}$: $\lambda_j$

Quantum walk

Unitary operator $U$ derived from $P$

(locality of $P \rightarrow$ locality of $U$)

Eigenvalues of $U$: $e^{\pm i \arcsin \lambda_j}$

Claim: Any symmetric matrix $H$ with positive entries can be factorized as $H = \sqrt{P \circ P^T}$ for some stochastic matrix $P$. (use Perron vector)

(note that locality of $H \rightarrow$ locality of $P$)
Continuous time $\rightarrow$ discrete time

Szegedy quantization of classical Markov chains:

**Classical random walk**
- Stochastic matrix $P$
- Eigenvalues of $\sqrt{P \circ P^T}$: $\lambda_j$

**Quantum walk**
- Unitary operator $U$ derived from $P$
- Eigenvalues of $U$: $e^{\pm i \arcsin \lambda_j}$

**Claim:** Any symmetric matrix $H$ with positive entries can be factorized as $H = \sqrt{P \circ P^T}$ for some stochastic matrix $P$. (use Perron vector)

(Note that locality of $H \rightarrow$ locality of $P$)

This gives a general way to relate continuous- and discrete-time quantum walk. Small eigenphases of $e^{-iH}$ and $U$ are equal up to third order.
A quantum walk algorithm clearly cannot work for highly unbalanced formulas:
Formula rebalancing

A quantum walk algorithm clearly cannot work for highly unbalanced formulas:

But we can apply

**Theorem [Bshouty, Cleve, Eberly 91]**: Any NAND formula of size $n$ can be rewritten as an equivalent NAND formula of depth $O(\log n)$ and size $n^{1+o(1)}$. 
Applications to recursive functions

Recursive “all equal” function [Ambainis 03]

\[ f(x, y, z) = \begin{cases} 
1 & x = y = z \\
0 & \text{otherwise} 
\end{cases} \]

recurse \( k \) times

Polynomial degree: \( 2^k \)

Q. query complexity: \( \Omega((\frac{3}{\sqrt{2}})^k) = \Omega(2.12^k) \) (adversary method)
\[ O(\sqrt{6^k}) = O(2.45^k) \] (NAND of 6)

Recursive majority function [Boppana 86]

\[ f(x, y, z) = \begin{cases} 
1 & x + y + z \geq 2 \\
0 & \text{otherwise} 
\end{cases} \]

recurse \( k \) times

C. query complexity [JKS 03]: \( \Omega((\frac{7}{3})^k) = \Omega(2.33^k) \)
\[ o((\frac{8}{3})^k) = o(2.67^k) \]

Q. query complexity: \( \Omega(2^k) \) (adversary method)
\[ O(\sqrt{5^k}) = O(2.24^k) \] (NAND of 5)
Closed problems

This also resolves a conjecture of [O’Donnell-Servedio 03]:
Any NAND formula of size $n$ can be approximated by a polynomial of degree $\sqrt{n^{1+o(1)}}$.

Hence formulas are (classically!) PAC learnable in time $2\sqrt{n^{1+o(1)}}$. 
Closed problems

This also resolves a conjecture of [O’Donnell-Servedio 03]: Any NAND formula of size $n$ can be approximated by a polynomial of degree $\sqrt{n^{1+o(1)}}$. Hence formulas are (classically!) PAC learnable in time $2\sqrt{n^{1+o(1)}}$.

[Reichardt, Špalek 07]: Generalization to formulas built from other gates, using new gate widgets, as well as the concept of span programs. Gives optimal (or nearly optimal) algorithms for many other functions, including an optimal algorithm ($O(2^k)$) for recursive ternary majority.
Closed problems

This also resolves a conjecture of [O’Donnell-Servedio 03]:
Any NAND formula of size $n$ can be approximated by a polynomial of degree $\sqrt{n^{1+o(1)}}$.
Hence formulas are (classically!) PAC learnable in time $2^{\sqrt{n^{1+o(1)}}}$.

[Reichardt, Špalek 07]: Generalization to formulas built from other gates, using new gate widgets, as well as the concept of span programs. Gives optimal (or nearly optimal) algorithms for many other functions, including an optimal algorithm ($O(2^k)$) for recursive ternary majority.

Open problems

• Formulas with yet more general gates?
• Similar algorithm for circuits?
• Can we compute a certificate for the value of a formula?
• Improved formula rebalancing?