Span programs and quantum time complexity

A. J. Cornelissen[1] S. Jeffery[2] M. Ozols[1] A. Piedrafita[2]

[1]QuSoft – University of Amsterdam
[2]QuSoft – CWI

June 22nd, 2020

arXiv:2005.01323

UNIVERSITY OF AMSTERDAM

QuSoft
Research Center for Quantum Software

CWI

Overview

Overview

1. High-level discussion

Overview

1. High-level discussion
2. Technical part

Overview

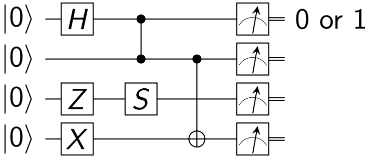
1. High-level discussion
2. Technical part
   1. Span program $\Rightarrow$ quantum algorithm
   2. Quantum algorithm $\Rightarrow$ span program

Overview

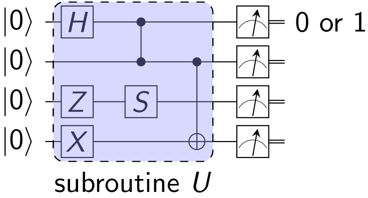
1. High-level discussion
2. Technical part
   1. Span program $\Rightarrow$ quantum algorithm
   2. Quantum algorithm $\Rightarrow$ span program
3. Application to variable-time search

Quantum query algorithms

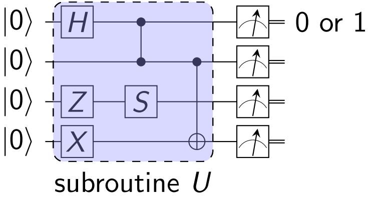
Quantum query algorithms

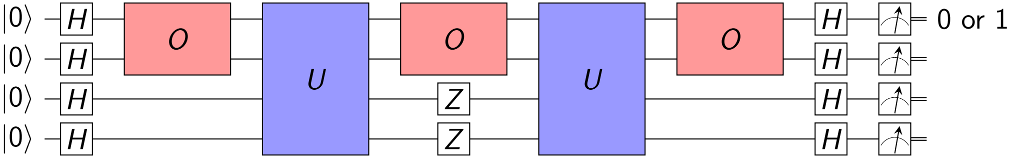


Quantum query algorithms



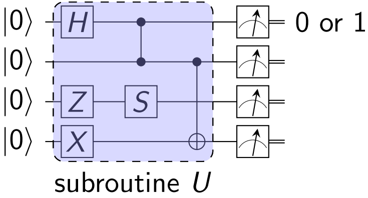
Quantum query algorithms

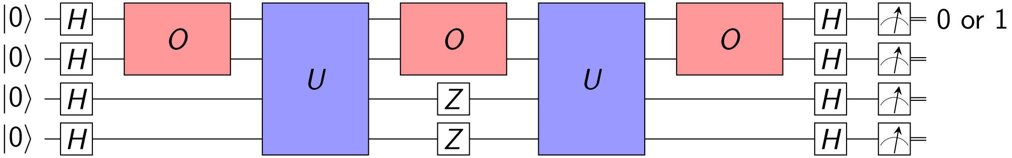




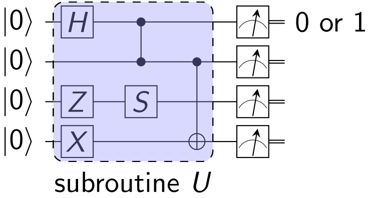
Quantum query algorithms

$$f : \{0,1\}^n \to \{0,1\}$$



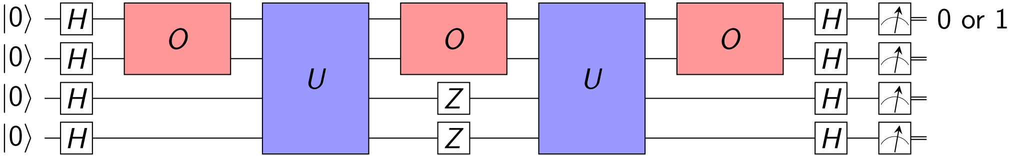


Quantum query algorithms

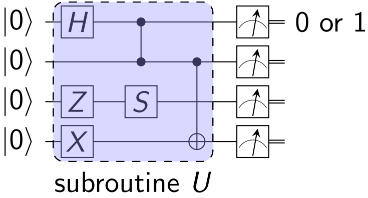


$$f : \{0,1\}^n \to \{0,1\}$$

Given $x \in \{0,1\}^n$, calculate $f(x)$.

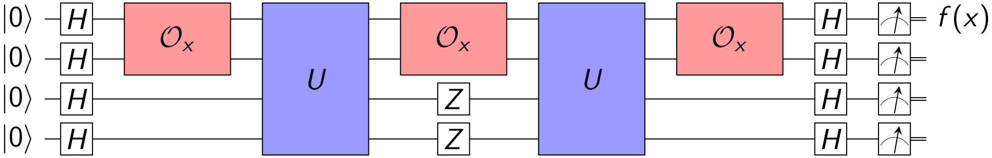


Quantum query algorithms

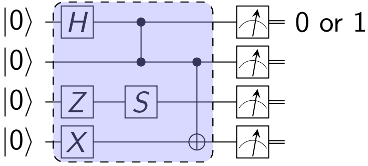


$$f : \{0,1\}^n \to \{0,1\}$$

Given $x \in \{0,1\}^n$, calculate $f(x)$.



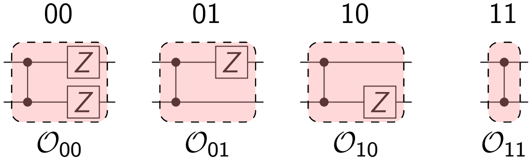
Quantum query algorithms

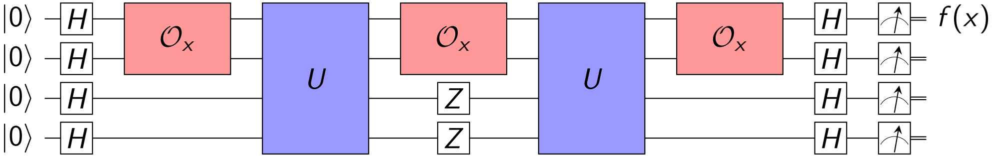


subroutine U

$$f : \{0,1\}^n \to \{0,1\}$$

Given $x \in \{0,1\}^n$, calculate $f(x)$.

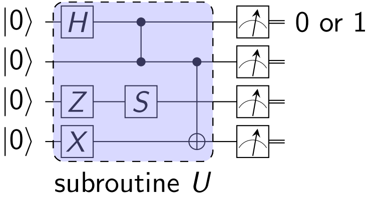


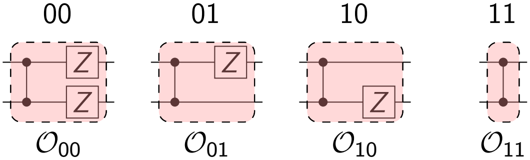


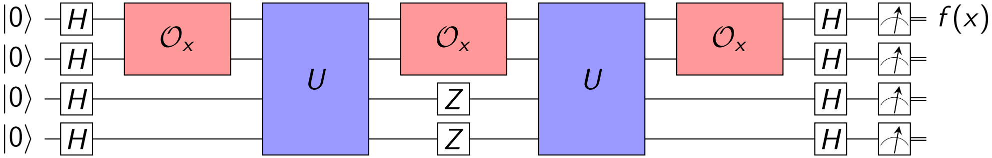
Quantum query algorithms

$$f : \{0,1\}^n \to \{0,1\}$$

Given $x \in \{0,1\}^n$, calculate $f(x)$.

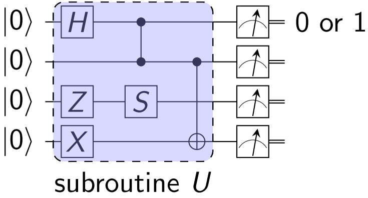






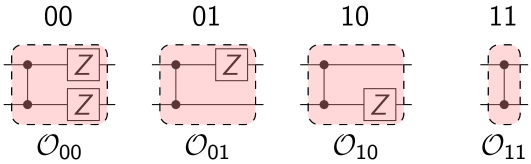
S: Query complexity no. calls to oracle circuit $\mathcal{O}_x$ $S = 3$

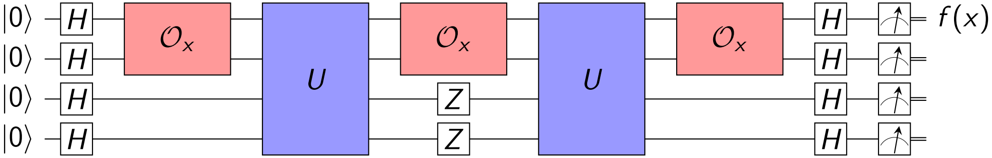
Quantum query algorithms



$$f : \{0,1\}^n \to \{0,1\}$$

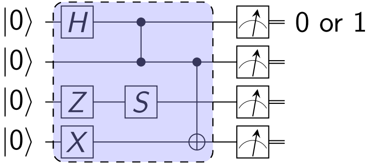
Given $x \in \{0,1\}^n$, calculate $f(x)$.





S:	Query complexity	no. calls to oracle circuit $\mathcal{O}_x$	$S = 3$
T:	Time complexity	no. elementary gates	$T = 10 + 2\,\mathrm{TC}(U) + 3\,\mathrm{TC}(\mathcal{O}_x)$

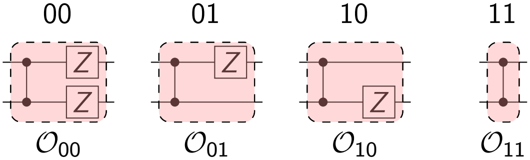
Quantum query algorithms

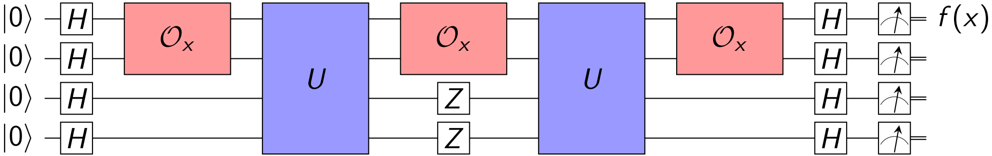


subroutine U

$$f : \{0,1\}^n \to \{0,1\}$$

Given $x \in \{0,1\}^n$, calculate $f(x)$.





S:	Query complexity	no. calls to oracle circuit $\mathcal{O}_x$	$S = 3$
T:	Time complexity	no. elementary gates	$T = 10 + 2\,\mathrm{TC}(U) + 3\,\mathrm{TC}(\mathcal{O}_x)$
k:	Space complexity	no. qubits	$k = 4$

Frameworks

Frameworks

A framework has two essential properties:

1. Encode f into several mathematical objects.
2. A quantum algorithm is generated from these.

Frameworks

A framework has two essential properties:

1. Encode f into several mathematical objects.
2. A quantum algorithm is generated from these.

Examples:

1. Quantum random walks (Ambainis, '03)

Frameworks

A framework has two essential properties:

1. Encode f into several mathematical objects.
2. A quantum algorithm is generated from these.

Examples:

1. Quantum random walks (Ambainis, '03)
2. Quantum singular value transformations (Gilyén, Su, Low, Wiebe, '18)

Frameworks

A framework has two essential properties:

1. Encode f into several mathematical objects.
2. A quantum algorithm is generated from these.

Examples:

1. Quantum random walks (Ambainis, '03)
2. Quantum singular value transformations (Gilyén, Su, Low, Wiebe, '18)
3. Completely bounded forms (Arunachalam, Briët, Palazuelos, '18)

Frameworks

A framework has two essential properties:

1. Encode f into several mathematical objects.
2. A quantum algorithm is generated from these.

Examples:

1. Quantum random walks (Ambainis, '03)
2. Quantum singular value transformations (Gilyén, Su, Low, Wiebe, '18)
3. Completely bounded forms (Arunachalam, Briët, Palazuelos, '18)
4. Span programs (Reichardt, '09)

Frameworks

A framework has two essential properties:

1. Encode f into several mathematical objects.
2. A quantum algorithm is generated from these.

Examples:

1. Quantum random walks (Ambainis, '03)
2. Quantum singular value transformations (Gilyén, Su, Low, Wiebe, '18)
3. Completely bounded forms (Arunachalam, Briët, Palazuelos, '18)
4. Span programs (Reichardt, '09)
5. ...

Span program framework

Span program framework

Has been used to design:

1. Quantum algorithms from solutions to the dual adversary bound (Reichardt, '09).

Span program framework

Has been used to design:

1. Quantum algorithms from solutions to the dual adversary bound (Reichardt, '09).
2. Quantum algorithm for k-element distinctness (Belovs, '12).

Span program framework

Has been used to design:

1. Quantum algorithms from solutions to the dual adversary bound (Reichardt, '09).
2. Quantum algorithm for k-element distinctness (Belovs, '12).
3. Quantum algorithm for formula evaluation (Reichardt, Spalek, '12; Jeffery, Kimmel, '17).

Span program framework

Has been used to design:

1. Quantum algorithms from solutions to the dual adversary bound (Reichardt, '09).
2. Quantum algorithm for k-element distinctness (Belovs, '12).
3. Quantum algorithm for formula evaluation (Reichardt, Spalek, '12; Jeffery, Kimmel, '17).
4. Quantum algorithms for graph problems such as:
    1. Bipartiteness testing (Āriņš, '15; Beigi, Taghavi, '20)
    2. Cycle detection (Cade, Montanaro, Belovs, '16; Beigi, Taghavi, '20)
    3. st-connectivity (Jeffery, Kimmel, '17; Beigi, Taghavi, '20)

Span program framework

Has been used to design:

1. Quantum algorithms from solutions to the dual adversary bound (Reichardt, '09).
2. Quantum algorithm for k-element distinctness (Belovs, '12).
3. Quantum algorithm for formula evaluation (Reichardt, Spalek, '12; Jeffery, Kimmel, '17).
4. Quantum algorithms for graph problems such as:
    1. Bipartiteness testing (Āriņš, '15; Beigi, Taghavi, '20)
    2. Cycle detection (Cade, Montanaro, Belovs, '16; Beigi, Taghavi, '20)
    3. st-connectivity (Jeffery, Kimmel, '17; Beigi, Taghavi, '20)
5. ...

Span program framework

Has been used to design:

1. Quantum algorithms from solutions to the dual adversary bound (Reichardt, '09).
2. Quantum algorithm for k-element distinctness (Belovs, '12).
3. Quantum algorithm for formula evaluation (Reichardt, Spalek, '12; Jeffery, Kimmel, '17).
4. Quantum algorithms for graph problems such as:
    1. Bipartiteness testing (Āriņš, '15; Beigi, Taghavi, '20)
    2. Cycle detection (Cade, Montanaro, Belovs, '16; Beigi, Taghavi, '20)
    3. *st*-connectivity (Jeffery, Kimmel, '17; Beigi, Taghavi, '20)
5. ...

Analysis of these algorithms:

1. Query complexity: easy.
2. Time complexity: hard.

Span program framework

Has been used to design:

1. Quantum algorithms from solutions to the dual adversary bound (Reichardt, '09).
2. Quantum algorithm for k-element distinctness (Belovs, '12).
3. Quantum algorithm for formula evaluation (Reichardt, Spalek, '12; Jeffery, Kimmel, '17).
4. Quantum algorithms for graph problems such as:
    1. Bipartiteness testing (Āriņš, '15; Beigi, Taghavi, '20)
    2. Cycle detection (Cade, Montanaro, Belovs, '16; Beigi, Taghavi, '20)
    3. *st*-connectivity (Jeffery, Kimmel, '17; Beigi, Taghavi, '20)
5. ...

Analysis of these algorithms:

1. Query complexity: easy.
2. Time complexity: hard.

Motivation 1: We wish to learn more about the time complexity of span program algorithms.

Interconvertibility between span programs and quantum algorithms

Interconvertibility between span programs and quantum algorithms

Span program $\longrightarrow$ Quantum algorithm $\mathcal{B}$

Interconvertibility between span programs and quantum algorithms

General construction (Reichardt, '09; Jeffery, '20):

Quantum algorithm $\mathcal{A}$ $\longrightarrow$ Span program $\longrightarrow$ Quantum algorithm $\mathcal{B}$

Interconvertibility between span programs and quantum algorithms

General construction (Reichardt, ’09; Jeffery, ’20):

Quantum algorithm $\mathcal{A}$ → Span program → Quantum algorithm $\mathcal{B}$

with the following properties:

	Quantum algorithm $\mathcal{A}$	Quantum algorithm $\mathcal{B}$
Query complexity	S	$\mathcal{O}(S)$

Interconvertibility between span programs and quantum algorithms

General construction (Reichardt, '09; Jeffery, '20):

Quantum algorithm $\mathcal{A}$ → Span program → Quantum algorithm $\mathcal{B}$

with the following properties:

	Quantum algorithm $\mathcal{A}$	Quantum algorithm $\mathcal{B}$
Query complexity	S	$\mathcal{O}(S)$
Time complexity	T	

Motivation 2: Can we do the same with time complexity?

Interconvertibility between span programs and quantum algorithms

General construction (Reichardt, '09; Jeffery, '20):

$$\boxed{\text{Quantum algorithm } \mathcal{A}} \longrightarrow \boxed{\text{Span program}} \longrightarrow \boxed{\text{Quantum algorithm } \mathcal{B}}$$

with the following properties:

	Quantum algorithm $\mathcal{A}$	Quantum algorithm $\mathcal{B}$
Query complexity	S	$\mathcal{O}(S)$
Time complexity	T	$\mathcal{O}(T \,\mathrm{polylog}(T))$*

Motivation 2: Can we do the same with time complexity?

*if we have efficient uniform access to $\mathcal{A}$.

Interconvertibility between span programs and quantum algorithms

General construction (Reichardt, '09; Jeffery, '20):

Quantum algorithm $\mathcal{A}$ → Span program → Quantum algorithm $\mathcal{B}$

with the following properties:

	Quantum algorithm $\mathcal{A}$	Quantum algorithm $\mathcal{B}$
Query complexity	S	$\mathcal{O}(S)$
Time complexity	T	$\mathcal{O}(T \text{ polylog}(T))$*
Space complexity	k	$k + \mathcal{O}(\text{polylog}(T))$*

Motivation 2: Can we do the same with time complexity?
*if we have efficient uniform access to $\mathcal{A}$.

Interconvertibility between span programs and quantum algorithms

General construction (Reichardt, '09; Jeffery, '20):

Quantum algorithm $\mathcal{A}$ → Span program → Quantum algorithm $\mathcal{B}$

with the following properties:

	Quantum algorithm $\mathcal{A}$	Quantum algorithm $\mathcal{B}$
Query complexity	S	$\mathcal{O}(S)$
Time complexity	T	$\mathcal{O}(T\,\mathrm{polylog}(T))$*
Space complexity	k	$k + \mathcal{O}(\mathrm{polylog}(T))$*

Motivation 2: Can we do the same with time complexity?

*if we have efficient uniform access to $\mathcal{A}$.

Corollary: For every f, there exists a span program that generates a quantum algorithm that computes f with optimal query, time and space complexity.

Span programs – mathematical objects

Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.

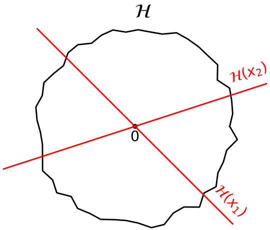
Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.



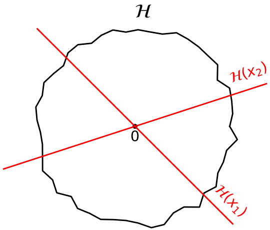
Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.



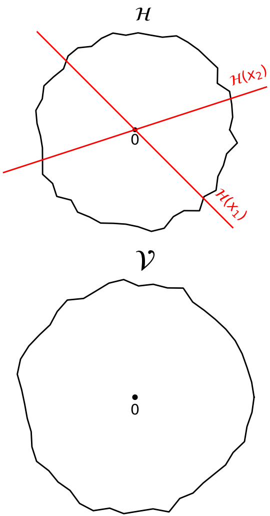
Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.



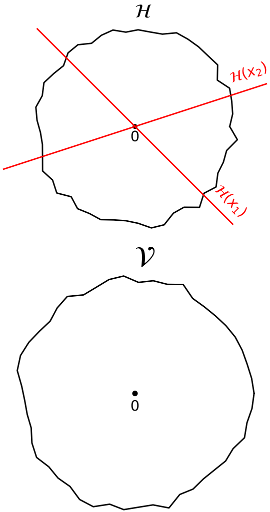
Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.
3. Target vector: $|\tau\rangle \in \mathcal{V}$.



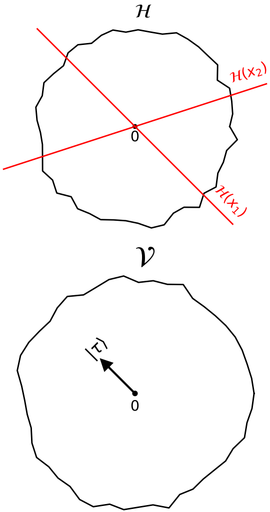
Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.
3. Target vector: $|\tau\rangle \in \mathcal{V}$.



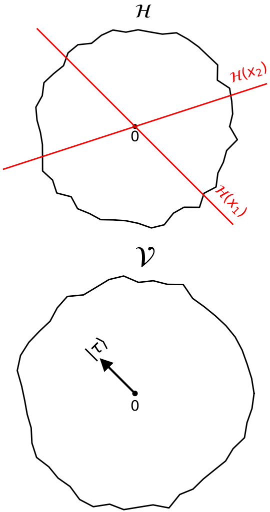
Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.
3. Target vector: $|\tau\rangle \in \mathcal{V}$.
4. Linear operator: $A \in \mathcal{L}(\mathcal{H}, \mathcal{V})$.



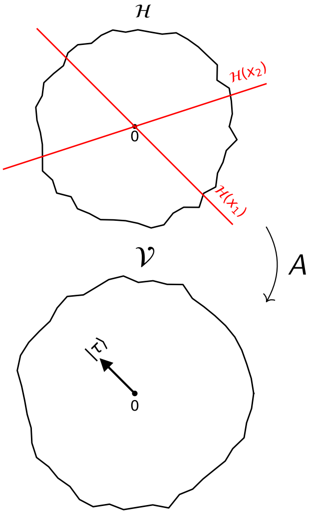
Span programs – mathematical objects

We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.
3. Target vector: $|\tau\rangle \in \mathcal{V}$.
4. Linear operator: $A \in \mathcal{L}(\mathcal{H}, \mathcal{V})$.



Span programs – mathematical objects

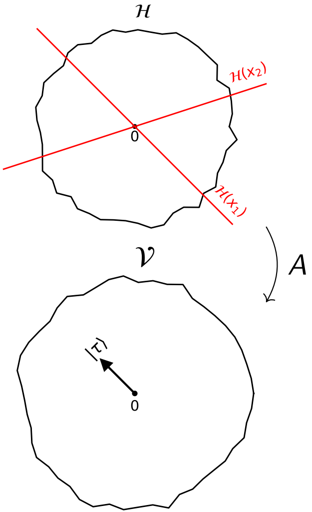
We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.
3. Target vector: $|\tau\rangle \in \mathcal{V}$.
4. Linear operator: $A \in \mathcal{L}(\mathcal{H}, \mathcal{V})$.

Let $|w_0\rangle = A^+ |\tau\rangle$.



Span programs – mathematical objects

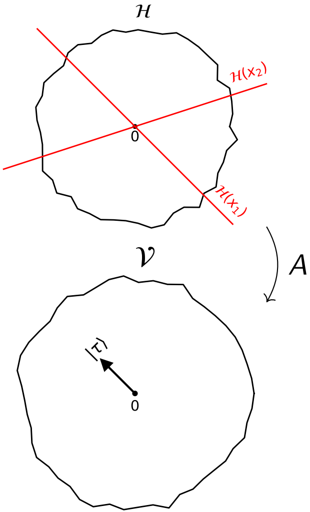
We wish to design a quantum algorithm that computes

$$f : \{0,1\}^n \to \{0,1\}.$$

Four mathematical objects:

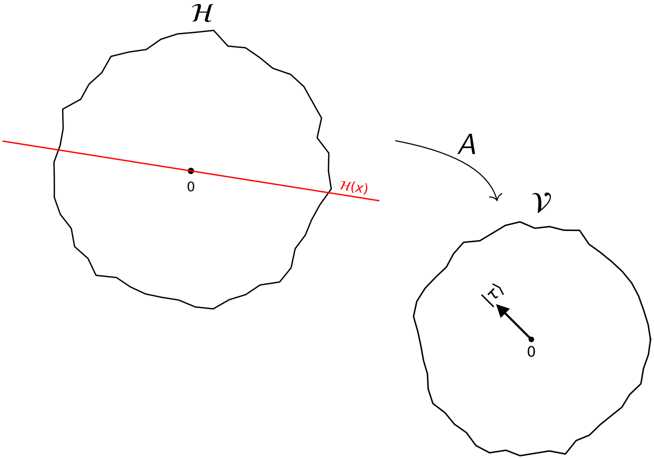
1. Hilbert space: $\mathcal{H}$,
   for every $x \in \{0,1\}^n$, a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$.
2. Target space: $\mathcal{V}$.
3. Target vector: $|\tau\rangle \in \mathcal{V}$.
4. Linear operator: $A \in \mathcal{L}(\mathcal{H}, \mathcal{V})$.

Let $|w_0\rangle = A^+ |\tau\rangle$.
Span program evaluates f if
$f(x) = 1 \Leftrightarrow |w_0\rangle \in \mathcal{H}(x) + \ker(A)$.



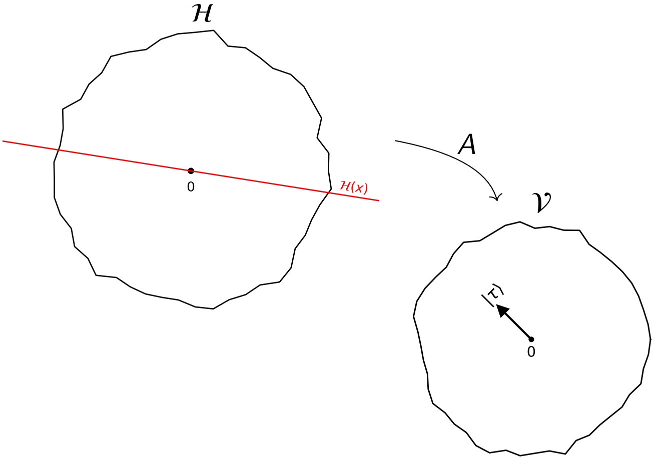
Span programs – visualization

Span programs – visualization



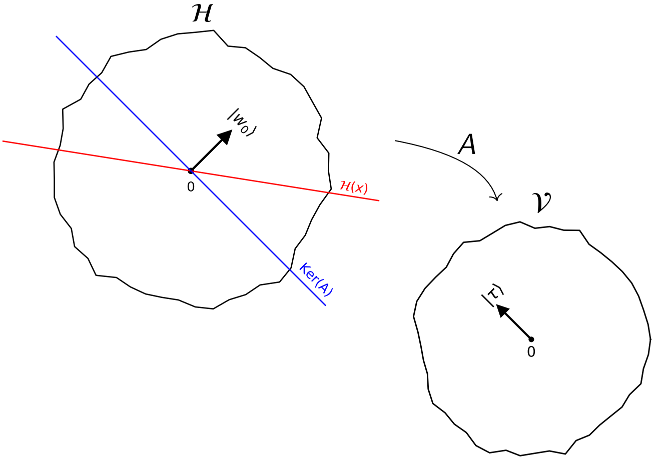
Span programs – visualization

Positive instance: $f(x) = 1$

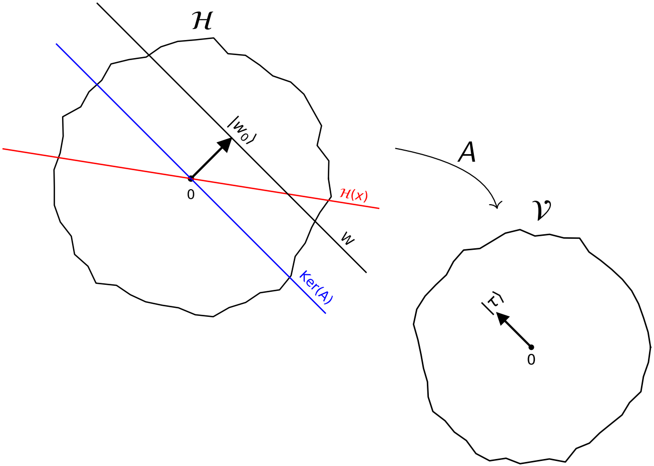


Span programs – visualization

Positive instance: $f(x) = 1$

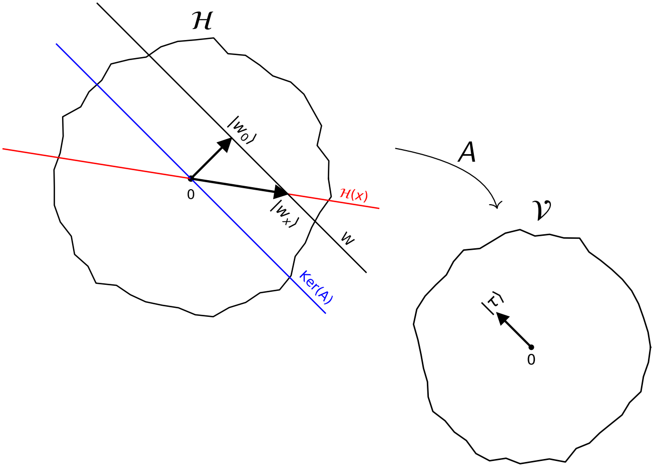


Positive instance: $f(x) = 1$



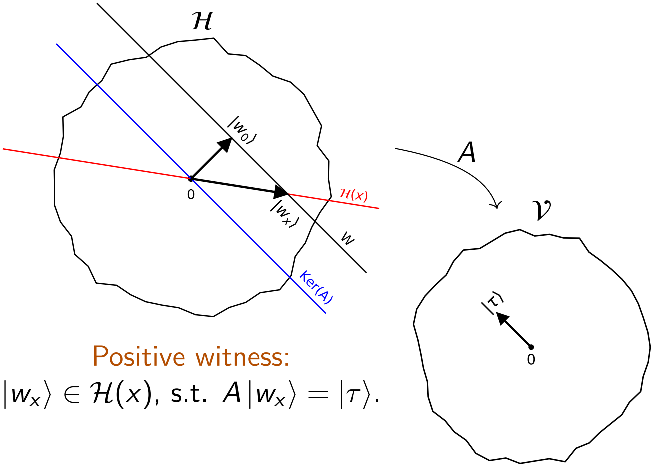
Span programs – visualization

Positive instance: $f(x) = 1$



Span programs – visualization

Positive instance: $f(x) = 1$

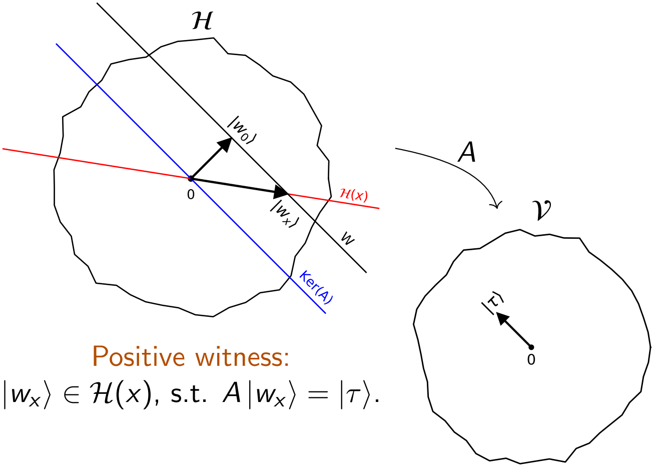


Positive witness:
$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.

Span programs – visualization

Positive instance: $f(x) = 1$

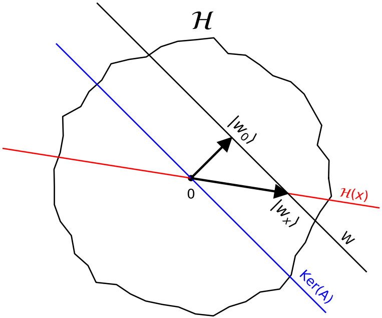
Negative instance: $f(x) = 0$



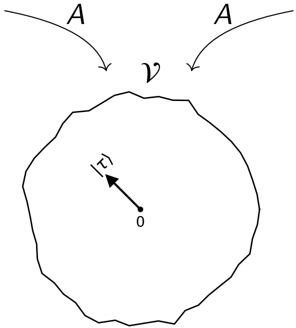
Positive witness:
$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.

Span programs – visualization

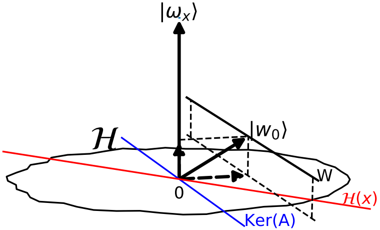
Positive instance: $f(x) = 1$



Positive witness:
$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.

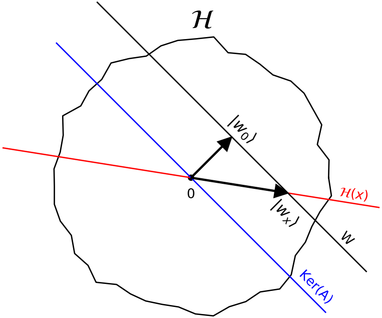


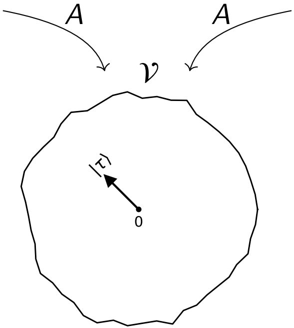
Negative instance: $f(x) = 0$



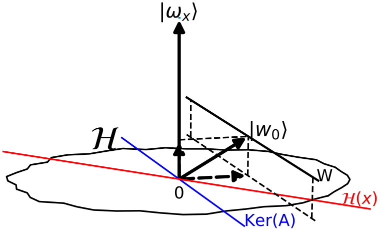
Span programs – visualization

Positive instance: $f(x) = 1$





Negative instance: $f(x) = 0$



Positive witness:

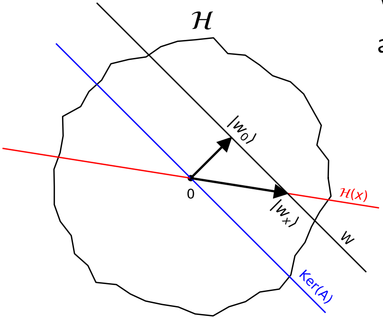
$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.

Negative witness:

$|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle \omega_x | w_0 \rangle = 1$.

Span programs – visualization

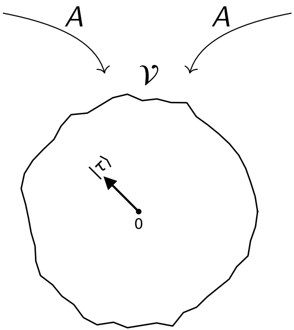
Positive instance: $f(x) = 1$



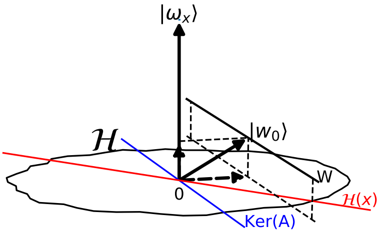
Positive witness:

$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.

We reflect through $\mathcal{H}(x)$ and then through $\ker(A)$.



Negative instance: $f(x) = 0$

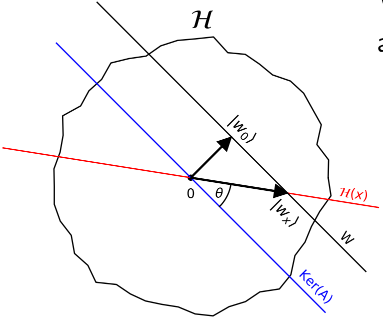


Negative witness:

$|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle \omega_x | w_0 \rangle = 1$.

Span programs – visualization

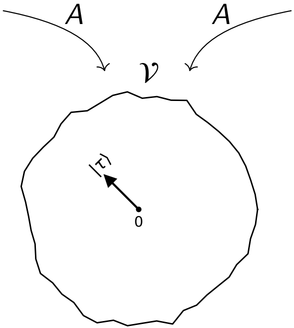
Positive instance: $f(x) = 1$



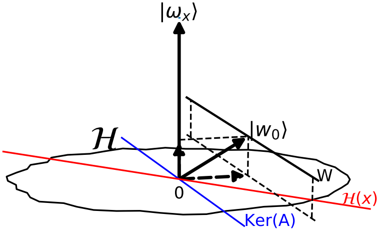
Positive witness:

$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A\,|w_x\rangle = |\tau\rangle$.

We reflect through $\mathcal{H}(x)$ and then through $\ker(A)$.



Negative instance: $f(x) = 0$



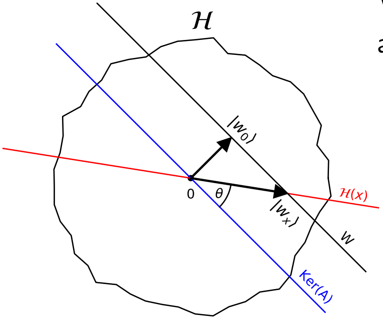
Negative witness:

$|\omega_x\rangle \in \mathcal{H}(x)^{\perp} \cap \ker(A)^{\perp}$, s.t.

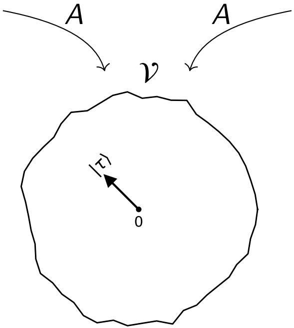
$\langle \omega_x | w_0 \rangle = 1$.

Span programs – visualization

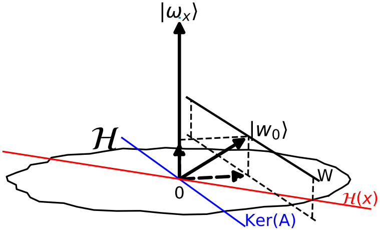
Positive instance: $f(x) = 1$



We reflect through $\mathcal{H}(x)$ and then through $\ker(A)$.



Negative instance: $f(x) = 0$



Positive witness:

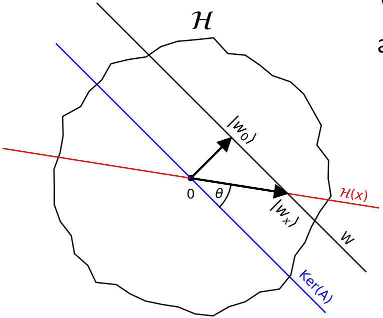
$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.
$|w_0\rangle$ rotates at angle 2θ,
$\theta \geq \sin\theta = \||w_0\rangle\| / \||w_x\rangle\|$.

Negative witness:

$|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t.
$\langle \omega_x | w_0 \rangle = 1$.

Span programs – visualization

Positive instance: $f(x) = 1$



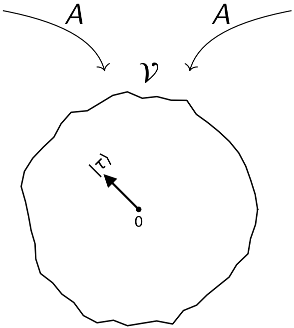
Positive witness:

$|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.

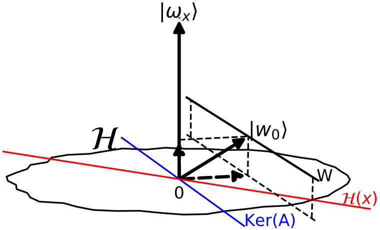
$|w_0\rangle$ rotates at angle 2θ,

$\theta \geq \sin\theta = \||w_0\rangle\| \,/\, \||w_x\rangle\|$.

We reflect through $\mathcal{H}(x)$ and then through $\ker(A)$.



Negative instance: $f(x) = 0$



Negative witness:

$|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t.

$\langle\omega_x|w_0\rangle = 1$.

Part of $|w_0\rangle$ does not rotate.

Span programs – algorithm construction

Witnesses of positive and negative instances:

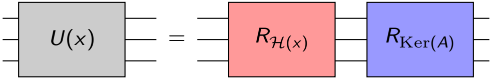
1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0|\omega_x\rangle = 1$.

Span programs – algorithm construction

Witnesses of positive and negative instances:

1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0|\omega_x\rangle = 1$.

Span program unitary:

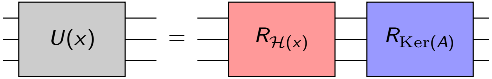


Span programs – algorithm construction

Witnesses of positive and negative instances:

1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0|\omega_x\rangle = 1$.

Span program unitary:



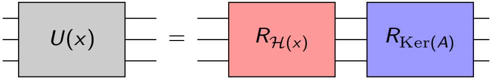
Run phase estimation with initial state $|w_0\rangle / \||w_0\rangle\|$.

Span programs – algorithm construction

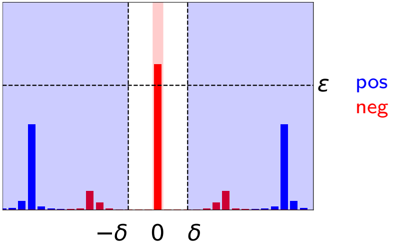
Witnesses of positive and negative instances:

1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0|\omega_x\rangle = 1$.

Span program unitary:



Run phase estimation with initial state $|w_0\rangle / \||w_0\rangle\|$.
Outcome distribution:



with $\delta = \frac{\||w_0\rangle\|}{\||w_x\rangle\|}$ and $\varepsilon = \frac{1}{\||\omega_x\rangle\| \cdot \||w_0\rangle\|}$.

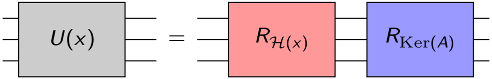
Span programs – algorithm construction

Witnesses of positive and negative instances:

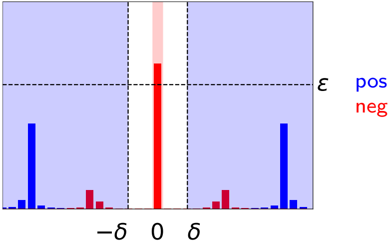
1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0|\omega_x\rangle = 1$.

Algorithm compiled from span program:

Span program unitary:



Run phase estimation with initial state $|w_0\rangle / \||w_0\rangle\|$.
Outcome distribution:



with $\delta = \frac{\||w_0\rangle\|}{\||w_x\rangle\|}$ and $\varepsilon = \frac{1}{\||\omega_x\rangle\| \cdot \||w_0\rangle\|}$.

Span programs – algorithm construction

Witnesses of positive and negative instances:

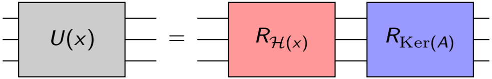
1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A\,|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0 | \omega_x \rangle = 1$.

Algorithm compiled from span program:

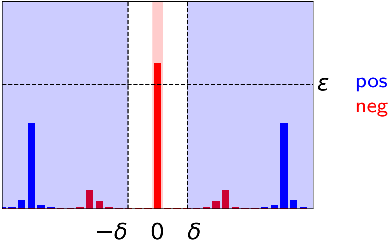
1. Run phase estimation up to precision

$$\delta = \frac{\| |w_0\rangle \|}{\max_{x \in f^{(-1)}(1)} \| |w_x\rangle \|}.$$

Span program unitary:



Run phase estimation with initial state $|w_0\rangle \,/\, \| |w_0\rangle \|$.
Outcome distribution:



with $\delta = \frac{\| |w_0\rangle \|}{\| |w_x\rangle \|}$ and $\varepsilon = \frac{1}{\| |\omega_x\rangle \| \cdot \| |w_0\rangle \|}$.

Span programs – algorithm construction

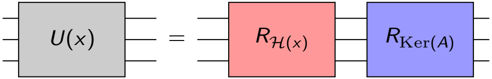
Witnesses of positive and negative instances:

1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A\,|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0|\omega_x\rangle = 1$.

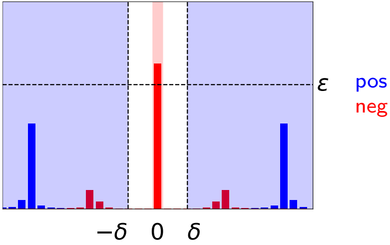
Algorithm compiled from span program:

1. Run phase estimation up to precision
$$\delta = \frac{\||w_0\rangle\|}{\max_{x \in f^{(-1)}(1)} \||w_x\rangle\|}.$$
2. Run amplitude estimation on top of that up to precision
$$\varepsilon = \frac{1}{\||w_0\rangle\| \cdot \max_{x \in f^{(-1)}(0)} \||\omega_x\rangle\|}.$$

Span program unitary:



Run phase estimation with initial state $|w_0\rangle / \||w_0\rangle\|$.
Outcome distribution:



with $\delta = \frac{\||w_0\rangle\|}{\||w_x\rangle\|}$ and $\varepsilon = \frac{1}{\||\omega_x\rangle\| \cdot \||w_0\rangle\|}$.

Span programs – algorithm construction

Witnesses of positive and negative instances:

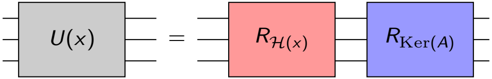
1. Positive witness:
   $|w_x\rangle \in \mathcal{H}(x)$, s.t. $A|w_x\rangle = |\tau\rangle$.
2. Negative witness:
   $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \ker(A)^\perp$, s.t. $\langle w_0|\omega_x\rangle = 1$.

Algorithm compiled from span program:

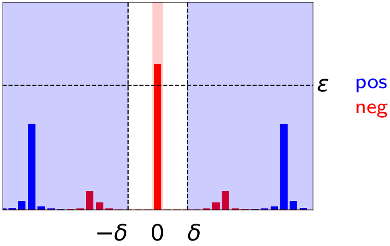
1. Run phase estimation up to precision
$$\delta = \frac{\||w_0\rangle\|}{\max_{x\in f^{(-1)}(1)} \||w_x\rangle\|}.$$
2. Run amplitude estimation on top of that up to precision
$$\varepsilon = \frac{1}{\||w_0\rangle\| \cdot \max_{x\in f^{(-1)}(0)} \||\omega_x\rangle\|}.$$

No. calls to $U(x)$ is
$$\mathcal{O}\left(\frac{1}{\delta\varepsilon}\right) = \mathcal{O}\left(\max_{x\in f^{(-1)}(1)} \||w_x\rangle\| \cdot \max_{x\in f^{(-1)}(0)} \||\omega_x\rangle\|\right).$$

Span program unitary:



Run phase estimation with initial state $|w_0\rangle / \||w_0\rangle\|$.
Outcome distribution:



with $\delta = \frac{\||w_0\rangle\|}{\||w_x\rangle\|}$ and $\varepsilon = \frac{1}{\||\omega_x\rangle\|\cdot\||w_0\rangle\|}$.

Span programs – algorithm analysis

Shorthand notation:

$$W_+ = \max_{x \in f^{(-1)}(1)} \left\| |w_x\rangle \right\|^2 \qquad \text{and} \qquad W_- = \max_{x \in f^{(-1)}(0)} \left\| |\omega_x\rangle \right\|^2 .$$

Span programs – algorithm analysis

Shorthand notation:

$$W_+ = \max_{x \in f^{(-1)}(1)} \| |w_x\rangle \|^2 \qquad \text{and} \qquad W_- = \max_{x \in f^{(-1)}(0)} \| |\omega_x\rangle \|^2 .$$

Implementation cost of the algorithm compiled from a span program:

Type	Cost
No. calls to $R_{\ker(A)}$	$\mathcal{O}(\sqrt{W_+ W_-})$
No. calls to $R_{\mathcal{H}(x)}$	$\mathcal{O}(\sqrt{W_+ W_-})$
No. calls to $C_{\|w_0\rangle}$	$\mathcal{O}(\sqrt{W_+ W_-})$
No. calls to $R_{\|0\rangle}$	$\mathcal{O}(\sqrt{W_+ W_-})$
No. extra gates	$\mathcal{O}(\text{polylog}\, W_+ W_-)$
No. extra qubits	$\mathcal{O}(\text{polylog}\, W_+ W_-)$

Span programs compiled from algorithms (I)

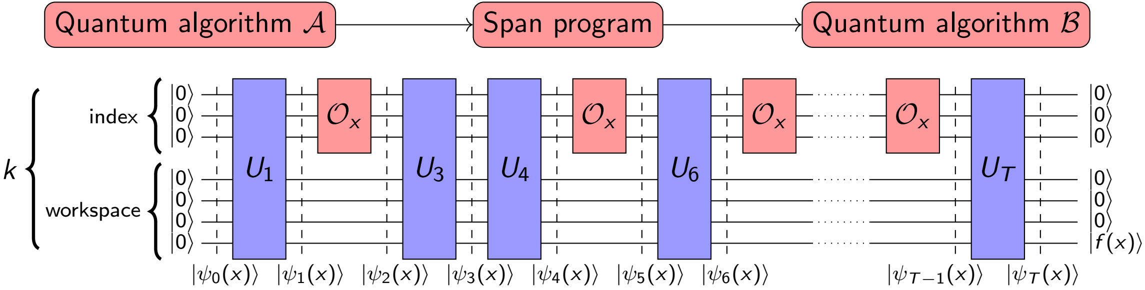
Span programs compiled from algorithms (I)

Span program $\longrightarrow$ Quantum algorithm $\mathcal{B}$

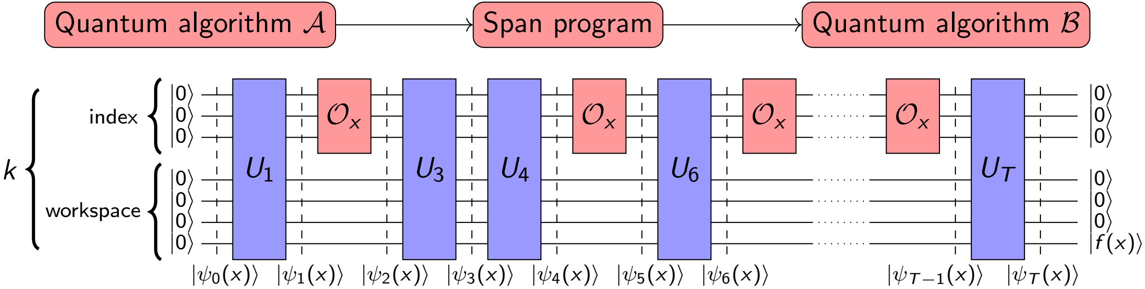
Span programs compiled from algorithms (I)

$$\text{Quantum algorithm } \mathcal{A} \longrightarrow \text{Span program} \longrightarrow \text{Quantum algorithm } \mathcal{B}$$

Span programs compiled from algorithms (I)

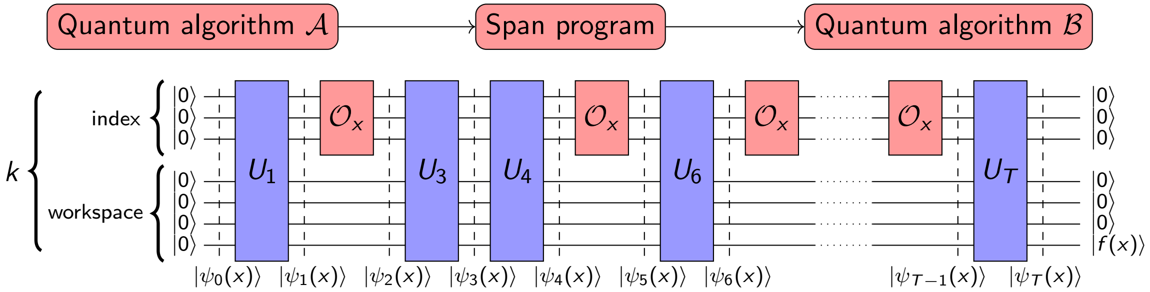


Span programs compiled from algorithms (I)



1. $|\psi_0(x)\rangle = |00\cdots 00\rangle$.
2. $|\psi_T(x)\rangle = |00\cdots 0f(x)\rangle$.

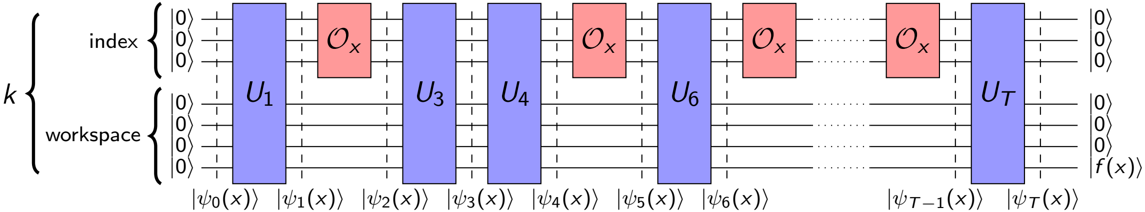
Span programs compiled from algorithms (I)



1. $|\psi_0(x)\rangle = |00\cdots00\rangle$.
2. $|\psi_T(x)\rangle = |00\cdots0f(x)\rangle$.
3. Every U_j can only consist of $\mathcal{O}(\text{polylog}(T))$ elementary gates.

Span programs compiled from algorithms (I)

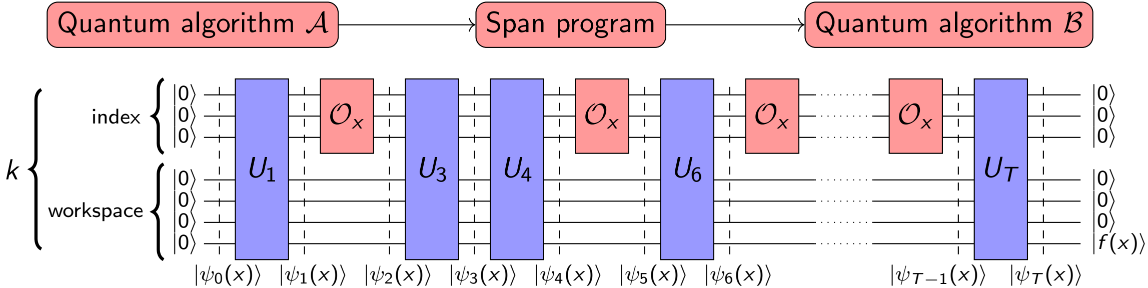
Quantum algorithm $\mathcal{A}$ $\longrightarrow$ Span program $\longrightarrow$ Quantum algorithm $\mathcal{B}$



1. $|\psi_0(x)\rangle = |00\cdots00\rangle$.
2. $|\psi_T(x)\rangle = |00\cdots0f(x)\rangle$.
3. Every U_j can only consist of $\mathcal{O}(\text{polylog}(T))$ elementary gates.

1. S: Query complexity

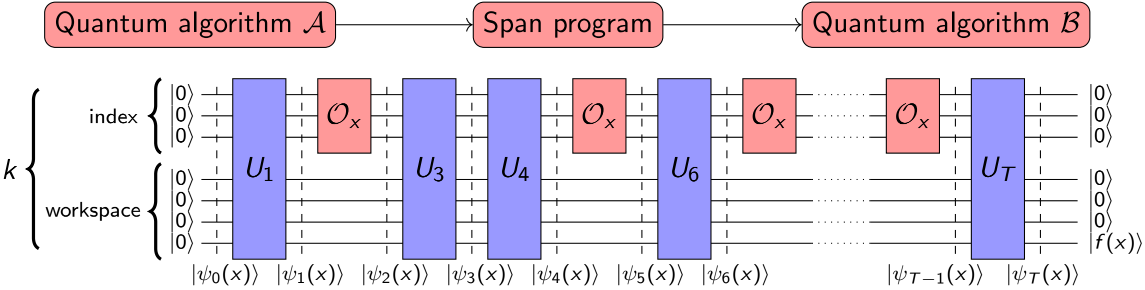
Span programs compiled from algorithms (I)



1. $|\psi_0(x)\rangle = |00\cdots00\rangle$.
2. $|\psi_T(x)\rangle = |00\cdots0f(x)\rangle$.
3. Every U_j can only consist of $\mathcal{O}(\text{polylog}(T))$ elementary gates.

1. S: Query complexity
2. T: No. time steps

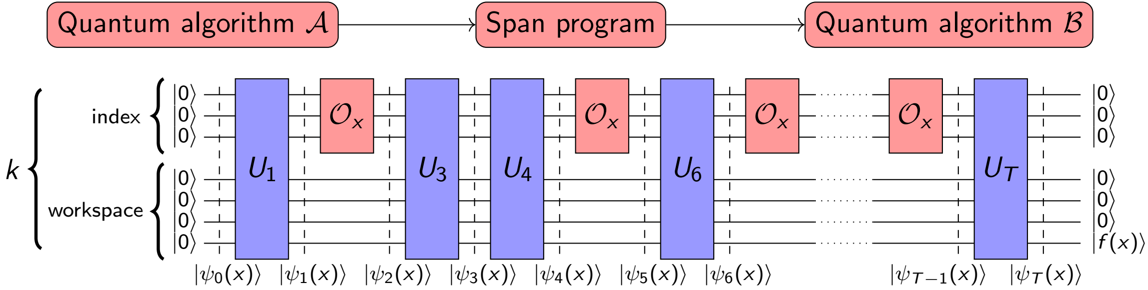
Span programs compiled from algorithms (I)



1. $|\psi_0(x)\rangle = |00\cdots 00\rangle$.
2. $|\psi_T(x)\rangle = |00\cdots 0f(x)\rangle$.
3. Every U_j can only consist of $\mathcal{O}(\text{polylog}(T))$ elementary gates.

1. S: Query complexity
2. T: No. time steps
3. k: No. qubits

Span programs compiled from algorithms (I)



1. $|\psi_0(x)\rangle = |00 \cdots 00\rangle$.
2. $|\psi_T(x)\rangle = |00 \cdots 0f(x)\rangle$.
3. Every U_j can only consist of $\mathcal{O}(\text{polylog}(T))$ elementary gates.

1. S: Query complexity
2. T: No. time steps
3. k: No. qubits
4. ε: Error probability

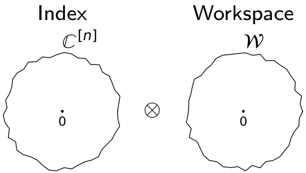
Span programs compiled from algorithms (II)

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:

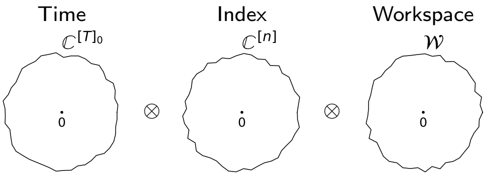
Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



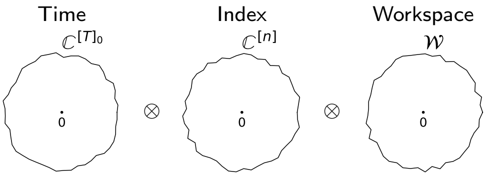
Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:

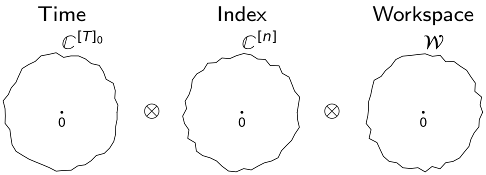


Span program operator A:

$$|t\rangle |\psi\rangle \overset{A}{\mapsto} |t\rangle |\psi\rangle - |t+1\rangle U_{t+1} |\psi\rangle ,$$

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



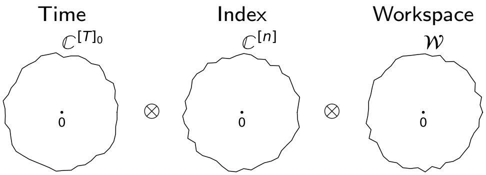
Span program operator A:

$$|t\rangle |\psi\rangle \stackrel{A}{\mapsto} |t\rangle |\psi\rangle - |t+1\rangle U_{t+1} |\psi\rangle ,$$

Target vector $|\tau\rangle$: $|\tau\rangle = |0\rangle |00\cdots00\rangle - |T\rangle |00\cdots01\rangle$.

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



Span program operator A:

$$|t\rangle|\psi\rangle \overset{A}{\mapsto} |t\rangle|\psi\rangle - |t+1\rangle U_{t+1}|\psi\rangle,$$

Target vector $|\tau\rangle$: $|\tau\rangle = |0\rangle|00\cdots00\rangle - |T\rangle|00\cdots01\rangle$.
Core idea:

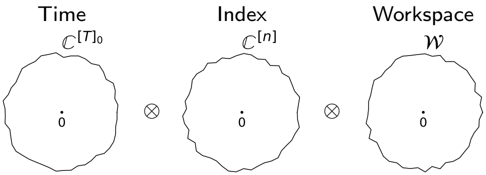
$$|w_x\rangle := \sum_{t=0}^{T-1} |t\rangle|\psi_t(x)\rangle \overset{A}{\mapsto} |0\rangle|\psi_0(x)\rangle - |T\rangle|\psi_T(x)\rangle$$

$$= |0\rangle|00\cdots00\rangle - |T\rangle|00\cdots0f(x)\rangle,$$

which equals $|\tau\rangle$ for positive instances.

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



Problems:

Span program operator A:

$$|t\rangle |\psi\rangle \overset{A}{\mapsto} |t\rangle |\psi\rangle - |t+1\rangle U_{t+1} |\psi\rangle ,$$

Target vector $|\tau\rangle$: $|\tau\rangle = |0\rangle |00\cdots 00\rangle - |T\rangle |00\cdots 01\rangle$.
Core idea:

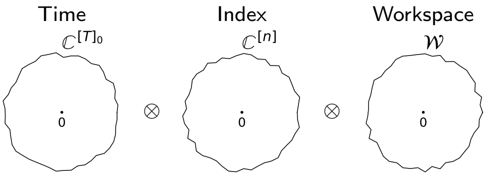
$$|w_x\rangle := \sum_{t=0}^{T-1} |t\rangle |\psi_t(x)\rangle \overset{A}{\mapsto} |0\rangle |\psi_0(x)\rangle - |T\rangle |\psi_T(x)\rangle$$

$$= |0\rangle |00\cdots 00\rangle - |T\rangle |00\cdots 0f(x)\rangle ,$$

which equals $|\tau\rangle$ for positive instances.

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



Span program operator A:

$$|t\rangle |\psi\rangle \overset{A}{\mapsto} |t\rangle |\psi\rangle - |t+1\rangle U_{t+1} |\psi\rangle ,$$

Target vector $|\tau\rangle$: $|\tau\rangle = |0\rangle |00\cdots 00\rangle - |T\rangle |00\cdots 01\rangle$.

Core idea:

$$|w_x\rangle := \sum_{t=0}^{T-1} |t\rangle |\psi_t(x)\rangle \overset{A}{\mapsto} |0\rangle |\psi_0(x)\rangle - |T\rangle |\psi_T(x)\rangle$$

$$= |0\rangle |00\cdots 00\rangle - |T\rangle |00\cdots 0f(x)\rangle ,$$

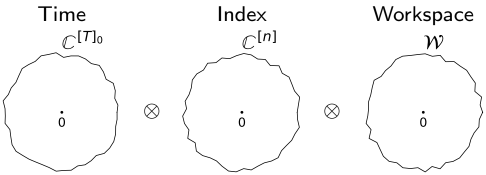
which equals $|\tau\rangle$ for positive instances.

Problems:

1. *The definition of A depends on x.* Solved by making $\mathcal{H}$ a little larger when $t+1$ is a query time step.

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



Span program operator A:

$$|t\rangle |\psi\rangle \overset{A}{\mapsto} |t\rangle |\psi\rangle - |t+1\rangle U_{t+1} |\psi\rangle ,$$

Target vector $|\tau\rangle$: $|\tau\rangle = |0\rangle |00\cdots 00\rangle - |T\rangle |00\cdots 01\rangle$.

Core idea:

$$|w_x\rangle := \sum_{t=0}^{T-1} |t\rangle |\psi_t(x)\rangle \overset{A}{\mapsto} |0\rangle |\psi_0(x)\rangle - |T\rangle |\psi_T(x)\rangle$$

$$= |0\rangle |00\cdots 00\rangle - |T\rangle |00\cdots 0f(x)\rangle ,$$

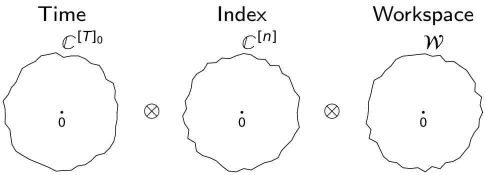
which equals $|\tau\rangle$ for positive instances.

Problems:

1. *The definition of A depends on x.* Solved by making $\mathcal{H}$ a little larger when $t+1$ is a query time step.
2. *The witness size $W_+ = \mathcal{O}(T)$.* Solved by tuning some weights.

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



Span program operator A:

$$|t\rangle\,|\psi\rangle \overset{A}{\mapsto} |t\rangle\,|\psi\rangle - |t+1\rangle\, U_{t+1}\,|\psi\rangle\,,$$

Target vector $|\tau\rangle$: $|\tau\rangle = |0\rangle\,|00\cdots 00\rangle - |T\rangle\,|00\cdots 01\rangle\,.$

Core idea:

$$|w_x\rangle := \sum_{t=0}^{T-1} |t\rangle\,|\psi_t(x)\rangle \overset{A}{\mapsto} |0\rangle\,|\psi_0(x)\rangle - |T\rangle\,|\psi_T(x)\rangle$$

$$= |0\rangle\,|00\cdots 00\rangle - |T\rangle\,|00\cdots 0f(x)\rangle\,,$$

which equals $|\tau\rangle$ for positive instances.

Problems:

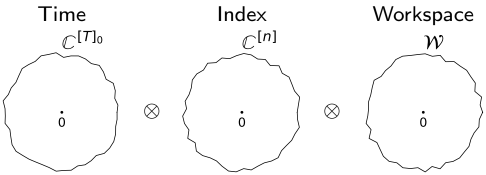
1. *The definition of A depends on x.*
   Solved by making $\mathcal{H}$ a little larger when $t+1$ is a query time step.
2. *The witness size $W_+ = \mathcal{O}(T)$.*
   Solved by tuning some weights.

After some modifications:

$$W_+ = \mathcal{O}(S) \qquad \text{and} \qquad W_- = \mathcal{O}(S).$$

Span programs compiled from algorithms (II)

Hilbert space $\mathcal{H}$ & target space $\mathcal{V}$:



Span program operator A:

$$|t\rangle\,|\psi\rangle \overset{A}{\mapsto} |t\rangle\,|\psi\rangle - |t+1\rangle\, U_{t+1}\,|\psi\rangle\,,$$

Target vector $|\tau\rangle$: $|\tau\rangle = |0\rangle\,|00\cdots 00\rangle - |T\rangle\,|00\cdots 01\rangle\,.$

Core idea:

$$|w_x\rangle := \sum_{t=0}^{T-1} |t\rangle\,|\psi_t(x)\rangle \overset{A}{\mapsto} |0\rangle\,|\psi_0(x)\rangle - |T\rangle\,|\psi_T(x)\rangle$$

$$= |0\rangle\,|00\cdots 00\rangle - |T\rangle\,|00\cdots 0f(x)\rangle\,,$$

which equals $|\tau\rangle$ for positive instances.

Problems:

1. *The definition of A depends on x.*
   Solved by making $\mathcal{H}$ a little larger when $t+1$ is a query time step.
2. *The witness size $W_+ = \mathcal{O}(T)$.*
   Solved by tuning some weights.

After some modifications:

$$W_+ = \mathcal{O}(S) \qquad \text{and} \qquad W_- = \mathcal{O}(S).$$

Hence number of calls to the subroutines is

$$\mathcal{O}(\sqrt{W_+W_-}) = \mathcal{O}(S).$$

Implementation of the subroutines of the algorithm span program

Implementation of the subroutines of the algorithm span program

It remains to calculate the implementation cost of $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$.

Implementation of the subroutines of the algorithm span program

It remains to calculate the implementation cost of $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$.
We require the following oracles:

$$\mathcal{O}_x : |i\rangle \mapsto (-1)^{x_i} |i\rangle , \qquad \mathcal{O}_{\mathcal{A}} : |t\rangle |\psi\rangle \mapsto |t\rangle U_t |\psi\rangle \qquad \text{and} \qquad \mathcal{O}_{\mathcal{S}} : |t\rangle \mapsto (-1)^{t \in \mathcal{S}} |t\rangle$$

Implementation of the subroutines of the algorithm span program

It remains to calculate the implementation cost of $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$.
We require the following oracles:

$$\mathcal{O}_x : |i\rangle \mapsto (-1)^{x_i} |i\rangle, \qquad \mathcal{O}_{\mathcal{A}} : |t\rangle |\psi\rangle \mapsto |t\rangle U_t |\psi\rangle \qquad \text{and} \qquad \mathcal{O}_{\mathcal{S}} : |t\rangle \mapsto (-1)^{t \in \mathcal{S}} |t\rangle$$

Analysis of the implementation of the subroutines:

Subroutine	Queries to $\mathcal{O}_x$	Queries to $\mathcal{O}_S$	Queries to $\mathcal{O}_{\mathcal{A}}$	No. extra gates	No. extra qubits	Implementation error
$R_{\ker(A)}$	0	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S\,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T))$	0
$R_{\mathcal{H}(x)}$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	0	$\mathcal{O}(\mathrm{polylog}(T))$	$\mathcal{O}(1)$	0
$C_{\lvert w_0\rangle}$	0	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S\,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T))$	$\mathcal{O}(\sqrt{\varepsilon})$
$R_{\lvert 0\rangle}$	0	0	0	$\mathcal{O}(\log(T))$	$\mathcal{O}(\log(T))$	$\mathcal{O}(\sqrt{\varepsilon})$
Total	$\mathcal{O}(S)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$	$\mathcal{O}(T\,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T))$	$\mathcal{O}(S\sqrt{\varepsilon})$
With error red.	$\mathcal{O}(S\log(S))$	$\mathcal{O}(T\log(S))$	$\mathcal{O}(T\log(S))$	$\mathcal{O}(T\,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T)+k^{o(1)})$	$\mathcal{O}(\sqrt{\varepsilon})$

Implementation of the subroutines of the algorithm span program

It remains to calculate the implementation cost of $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$.
We require the following oracles:

$$\mathcal{O}_x : |i\rangle \mapsto (-1)^{x_i} |i\rangle, \qquad \mathcal{O}_{\mathcal{A}} : |t\rangle |\psi\rangle \mapsto |t\rangle U_t |\psi\rangle \qquad \text{and} \qquad \mathcal{O}_{\mathcal{S}} : |t\rangle \mapsto (-1)^{t \in \mathcal{S}} |t\rangle$$

Analysis of the implementation of the subroutines:

Subroutine	Queries to $\mathcal{O}_x$	Queries to $\mathcal{O}_{\mathcal{S}}$	Queries to $\mathcal{O}_{\mathcal{A}}$	No. extra gates	No. extra qubits	Implementation error
$R_{\ker(A)}$	0	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S \,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T))$	0
$R_{\mathcal{H}(x)}$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	0	$\mathcal{O}(\mathrm{polylog}(T))$	$\mathcal{O}(1)$	0
$C_{\|w_0\rangle}$	0	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S)$	$\mathcal{O}(T/S \,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T))$	$\mathcal{O}(\sqrt{\varepsilon})$
$R_{\|0\rangle}$	0	0	0	$\mathcal{O}(\log(T))$	$\mathcal{O}(\log(T))$	$\mathcal{O}(\sqrt{\varepsilon})$
Total	$\mathcal{O}(S)$	$\mathcal{O}(T)$	$\mathcal{O}(T)$	$\mathcal{O}(T \,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T))$	$\mathcal{O}(S\sqrt{\varepsilon})$
With error red.	$\mathcal{O}(S\log(S))$	$\mathcal{O}(T\log(S))$	$\mathcal{O}(T\log(S))$	$\mathcal{O}(T \,\mathrm{polylog}(T))$	$\mathcal{O}(\mathrm{polylog}(T)+k^{o(1)})$	$\mathcal{O}(\sqrt{\varepsilon})$

Efficient uniform access: implementation of $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$ only takes $\mathcal{O}(\mathrm{polylog}(T))$ gates.

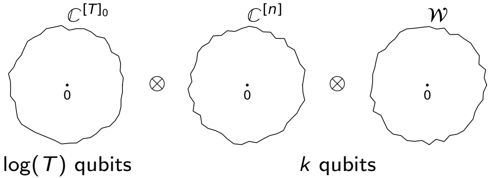
Implementing subspace

Implementing subspace

1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.

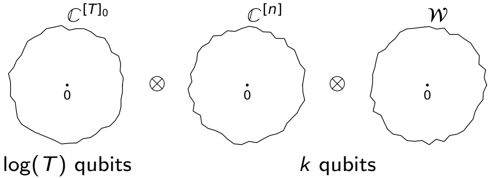
Implementing subspace

1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.



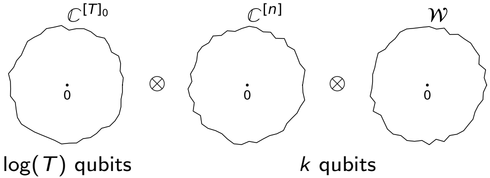
Implementing subspace

1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.
2. Reflecting through the all-zeros state naively takes $\mathcal{O}(k + \log(T))$ gates.



Implementing subspace

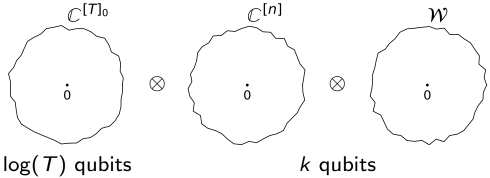
1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.
2. Reflecting through the all-zeros state naively takes $\mathcal{O}(k + \log(T))$ gates.
3. Core idea:



Implementing subspace

1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.
2. Reflecting through the all-zeros state naively takes $\mathcal{O}(k + \log(T))$ gates.
3. Core idea:
   1. We define an implementing subspace:

$$\mathcal{H}_x = \text{span}\{|t\rangle \, |\psi_t(x)\rangle : t \in [T]_0\},$$

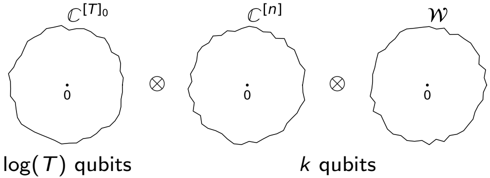


Implementing subspace

1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.
2. Reflecting through the all-zeros state naively takes $\mathcal{O}(k + \log(T))$ gates.
3. Core idea:
   1. We define an implementing subspace:

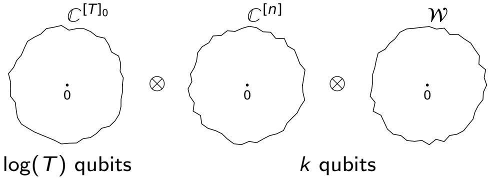
      $$\mathcal{H}_x = \operatorname{span}\{|t\rangle\,|\psi_t(x)\rangle : t \in [T]_0\},$$

   2. We prove that $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$ leave this space invariant.



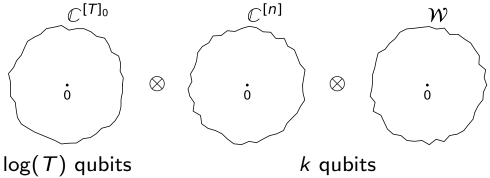
Implementing subspace

1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.
2. Reflecting through the all-zeros state naively takes $\mathcal{O}(k + \log(T))$ gates.
3. Core idea:
   1. We define an implementing subspace:
   $$\mathcal{H}_x = \mathrm{span}\{|t\rangle\,|\psi_t(x)\rangle : t \in [T]_0\},$$
   2. We prove that $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$ leave this space invariant.
   3. We prove that implementing $R_{|0\rangle}$ on this space can be done in $\mathcal{O}(\log(T))$ gates.



Implementing subspace

1. The state space is $\mathcal{O}(k + \log(T))$ qubits in size.
2. Reflecting through the all-zeros state naively takes $\mathcal{O}(k + \log(T))$ gates.
3. Core idea:
   1. We define an implementing subspace:
      $$\mathcal{H}_x = \mathrm{span}\{|t\rangle\,|\psi_t(x)\rangle : t \in [T]_0\},$$
   2. We prove that $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$ leave this space invariant.
   3. We prove that implementing $R_{|0\rangle}$ on this space can be done in $\mathcal{O}(\log(T))$ gates.
4. Technique of independent interest.



Application: variable-time search

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$,
each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$,
each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$,
each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$,
each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$,
each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$, each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

We are given access to these algorithms through

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$,
each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

We are given access to these algorithms through

1. $\mathcal{O}_x : |j\rangle |i\rangle \mapsto (-1)^{x_i^{(j)}} |j\rangle |i\rangle$,

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$,
each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

We are given access to these algorithms through

1. $\mathcal{O}_x : |j\rangle |i\rangle \mapsto (-1)^{x_i^{(j)}} |j\rangle |i\rangle$,
2. $\mathcal{O}_{\mathcal{A}} : |j\rangle |t\rangle |\psi\rangle \mapsto |j\rangle |t\rangle U_t^{(j)} |\psi\rangle$,

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$, each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

We are given access to these algorithms through

1. $\mathcal{O}_x : |j\rangle\,|i\rangle \mapsto (-1)^{x_i^{(j)}}\,|j\rangle\,|i\rangle\,,$
2. $\mathcal{O}_{\mathcal{A}} : |j\rangle\,|t\rangle\,|\psi\rangle \mapsto |j\rangle\,|t\rangle\,U_t^{(j)}\,|\psi\rangle\,,$
3. $\mathcal{O}_{\mathcal{S}} : |j\rangle\,|t\rangle \mapsto (-1)^{t \in \mathcal{S}^{(j)}}\,|j\rangle\,|t\rangle\,.$

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$, each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

We are given access to these algorithms through

1. $\mathcal{O}_x : |j\rangle\,|i\rangle \mapsto (-1)^{x_i^{(j)}}\,|j\rangle\,|i\rangle\,,$
2. $\mathcal{O}_{\mathcal{A}} : |j\rangle\,|t\rangle\,|\psi\rangle \mapsto |j\rangle\,|t\rangle\,U_t^{(j)}\,|\psi\rangle\,,$
3. $\mathcal{O}_{\mathcal{S}} : |j\rangle\,|t\rangle \mapsto (-1)^{t\in\mathcal{S}^{(j)}}\,|j\rangle\,|t\rangle\,.$

Now, we define the OR: $f(x^{(1)},\dots,x^{(n)}) = f_1(x^{(1)}) \vee f_2(x^{(2)}) \vee \cdots \vee f_n(x^{(n)})$.

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$, each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

We are given access to these algorithms through

1. $\mathcal{O}_x : |j\rangle |i\rangle \mapsto (-1)^{x_i^{(j)}} |j\rangle |i\rangle$,
2. $\mathcal{O}_{\mathcal{A}} : |j\rangle |t\rangle |\psi\rangle \mapsto |j\rangle |t\rangle U_t^{(j)} |\psi\rangle$,
3. $\mathcal{O}_{\mathcal{S}} : |j\rangle |t\rangle \mapsto (-1)^{t \in \mathcal{S}^{(j)}} |j\rangle |t\rangle$.

Now, we define the OR: $f(x^{(1)}, \dots, x^{(n)}) = f_1(x^{(1)}) \vee f_2(x^{(2)}) \vee \cdots \vee f_n(x^{(n)})$.
Variable-time search: how quickly can we compute f?

Application: variable-time search

Suppose we have n algorithms $\{\mathcal{A}_j\}_{j=1}^n$, each computing a function $f_j : \{0,1\}^{m_j} \to \{0,1\}$.

1. S_j: Query complexity
2. T_j: No. time steps
3. k_j: No. qubits
4. ε_j: Error probability

We are given access to these algorithms through

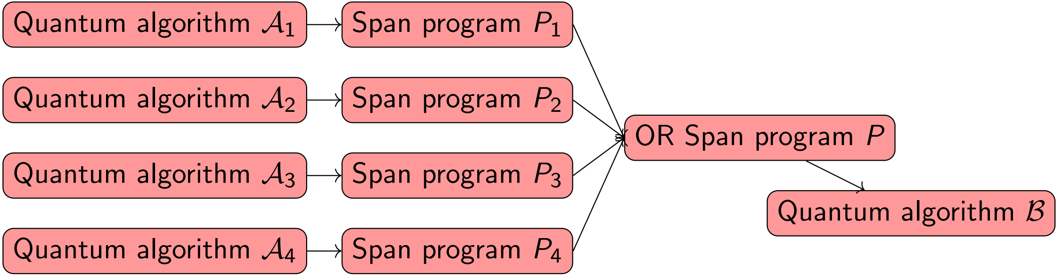
1. $\mathcal{O}_x : |j\rangle\,|i\rangle \mapsto (-1)^{x_i^{(j)}}\,|j\rangle\,|i\rangle$,
2. $\mathcal{O}_{\mathcal{A}} : |j\rangle\,|t\rangle\,|\psi\rangle \mapsto |j\rangle\,|t\rangle\,U_t^{(j)}\,|\psi\rangle$,
3. $\mathcal{O}_{\mathcal{S}} : |j\rangle\,|t\rangle \mapsto (-1)^{t\in\mathcal{S}^{(j)}}\,|j\rangle\,|t\rangle$.

Now, we define the OR: $f(x^{(1)},\dots,x^{(n)}) = f_1(x^{(1)}) \vee f_2(x^{(2)}) \vee \cdots \vee f_n(x^{(n)})$.
Variable-time search: how quickly can we compute f?

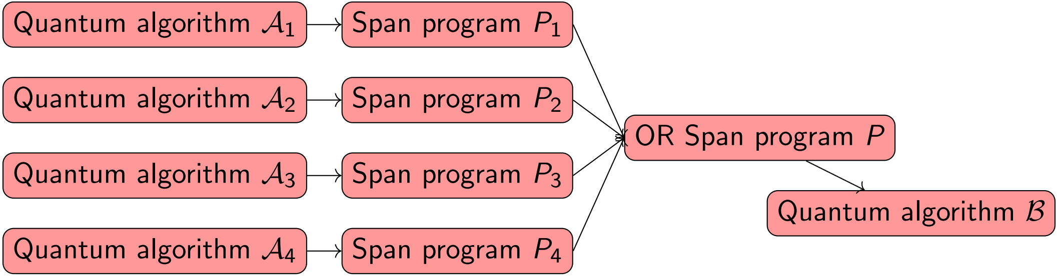
Method	No. queries to $\mathcal{O}_x$	No. queries to $\mathcal{O}_{\mathcal{A}}$ & $\mathcal{O}_{\mathcal{S}}$	No. extra gates
Naive approach	$\sum_{j=1}^n S_j$	$\sum_{j=1}^n T_j$	$\widetilde{\mathcal{O}}(\sum_{j=1}^n T_j)$
Ambainis '06 (I)	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n S_j^2}\right)$	X	?
Ambainis '06 (II)	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2}\right)$	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2}\right)$	?

Composition of span programs

Composition of span programs



Composition of span programs



The composition of span programs for *OR* is known (Reichardt, '09).

Algorithm for variable time search

Algorithm for variable time search

Careful construction of the subroutines $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$ yields:

Algorithm for variable time search

Careful construction of the subroutines $R_{\ker(A)}$, $R_{\mathcal{H}(x)}$, $C_{|w_0\rangle}$ and $R_{|0\rangle}$ yields:

Method	No. queries to $\mathcal{O}_x$	No. queries to $\mathcal{O}_{\mathcal{A}}$ & $\mathcal{O}_{\mathcal{S}}$	No. extra gates	Error probability
Naive approach	$\sum_{j=1}^n S_j$	$\sum_{j=1}^n T_j$	$\widetilde{\mathcal{O}}(\sum_{j=1}^n T_j)$	$\mathcal{O}\left(\sum_{j=1}^n \varepsilon_j\right)$
Ambainis '06 (I)	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n S_j^2}\right)$	X	?	-
Ambainis '06 (II)	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2}\right)$	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2}\right)$	?	-
Our result	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n S_j^2}\right)$	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2}\right)$	$\widetilde{\mathcal{O}}\left(\sqrt{\sum_{j=1}^n T_j^2}\right)$	$\mathcal{O}\left(\sum_{j=1}^n S_j^2 \sum_{j=1}^n \varepsilon_j\right)$
With error red.	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n S_j^2} \cdot \log\left(n \sum_{j=1}^n S_j^2\right)\right)$	$\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2} \cdot \log\left(n \sum_{j=1}^n S_j^2\right)\right)$	$\widetilde{\mathcal{O}}\left(\sqrt{\sum_{j=1}^n T_j^2}\right)$	$\mathcal{O}(\varepsilon_{\max})$

Summary

Summary

Our results:

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.
2. Construction algo $\Rightarrow$ span program $\Rightarrow$ algo that preserves time complexity.

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.
2. Construction algo $\Rightarrow$ span program $\Rightarrow$ algo that preserves time complexity.
3. Application to variable time search.

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.
2. Construction algo $\Rightarrow$ span program $\Rightarrow$ algo that preserves time complexity.
3. Application to variable time search.

Open problems:

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.
2. Construction algo $\Rightarrow$ span program $\Rightarrow$ algo that preserves time complexity.
3. Application to variable time search.

Open problems:

1. Make the construction more efficient w.r.t. the error probability.

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.
2. Construction algo $\Rightarrow$ span program $\Rightarrow$ algo that preserves time complexity.
3. Application to variable time search.

Open problems:

1. Make the construction more efficient w.r.t. the error probability.
2. Extend to more composition results, for instance composition for k-threshold?

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.
2. Construction algo $\Rightarrow$ span program $\Rightarrow$ algo that preserves time complexity.
3. Application to variable time search.

Open problems:

1. Make the construction more efficient w.r.t. the error probability.
2. Extend to more composition results, for instance composition for k-threshold?
3. Figure out the time complexity of Belovs's k-element distinctness algorithm.

Summary

Our results:

1. Implementation cost of span programs in terms of subroutines.
2. Construction algo $\Rightarrow$ span program $\Rightarrow$ algo that preserves time complexity.
3. Application to variable time search.

Open problems:

1. Make the construction more efficient w.r.t. the error probability.
2. Extend to more composition results, for instance composition for k-threshold?
3. Figure out the time complexity of Belovs's k-element distinctness algorithm.

Thanks for your attention!