Introduction to Quantum Algorithms
Part I: Quantum Gates and Simon's Algorithm

Martin Rötteler

NEC Laboratories America, Inc.
4 Independence Way, Suite 200
Princeton, NJ 08540, U.S.A.

International Summer School on Quantum Information,
Max-Planck-Institut für Physik komplexer Systeme
Dresden, August 31, 2005

Overview

Today:

- Introduction to qubits, quantum gates, and circuits.
- Appetizer: Two-bit problem where quantum beats classical!
- The power of quantum computing: Simon's algorithm
- Basic principles used:
  - Computing in superposition
  - Constructive/destructive interference

Tomorrow: Shor's algorithm and Grover's algorithm

Basics: Quantum Information

Quantum–bit (qubit)

A qubit is a normalized state in $\mathbb{C}^2$:

$$\alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \alpha, \beta \in \mathbb{C}, \quad \text{where } |\alpha|^2 + |\beta|^2 = 1.$$

Quantum register

A quantum register of length n is a collection of qubits $q_1, \ldots, q_n$.

Possible operations / dynamics

A **quantum computer** can

- perform unitary operations on quantum registers
- measure single qubits

Basics: Quantum Information

Quantum–bit (qubit)

A qubit is a normalized state in $\mathbb{C}^2$:

$$\alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \alpha, \beta \in \mathbb{C}, \quad \text{where } |\alpha|^2 + |\beta|^2 = 1.$$

Quantum register

A quantum register of length n is a collection of qubits $q_1, \ldots, q_n$.

Possible operations / dynamics

A **quantum computer** can

- perform unitary operations on quantum registers
- measure single qubits

Basics: Quantum Information

Quantum–bit (qubit)

A qubit is a normalized state in $\mathbb{C}^2$:

$$\alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \alpha, \beta \in \mathbb{C}, \quad \text{where } |\alpha|^2 + |\beta|^2 = 1.$$

Quantum register

A quantum register of length n is a collection of qubits $q_1, \ldots, q_n$.

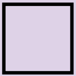
Possible operations / dynamics

A **quantum computer** can

- perform unitary operations on quantum registers
- measure single qubits

Bits vs Qubits

One classical bit



is in one of the basis states

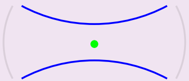
$$\uparrow \;\widehat{=}\; \textit{High} \;\widehat{=}\; 1$$
$$\downarrow \;\widehat{=}\; \textit{Low} \;\widehat{=}\; 0$$

Boolean state space:

$$\mathbb{F}_2 = \{\uparrow, \downarrow\} = \{0, 1\}$$

consisting of 2 elements

One quantum bit (qubit)



is in a coherent superposition

$$|\Psi\rangle = \alpha\,|\uparrow\rangle + \beta\,|\downarrow\rangle$$

of the basis states $|\uparrow\rangle, |\downarrow\rangle$

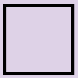
Quantum state space $\mathcal{H}_2 = \mathbb{C}^2$:

$$\left\{ \alpha\,|\uparrow\rangle + \beta\,|\downarrow\rangle : |\alpha|^2 + |\beta|^2 = 1 \right\}$$

of 2 dimensions

Bits vs Qubits

One classical bit



is in one of the basis states

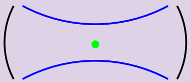
$$\uparrow \ \widehat{=}\ \textit{High}\ \widehat{=}\ 1$$
$$\downarrow \ \widehat{=}\ \textit{Low}\ \widehat{=}\ 0$$

Boolean state space:

$$\mathrm{F}_2 = \{\uparrow, \downarrow\} = \{0, 1\}$$

consisting of 2 elements

One quantum bit (qubit)



is in a coherent superposition

$$|\Psi\rangle = \alpha\,|\!\uparrow\rangle + \beta\,|\!\downarrow\rangle$$

of the basis states $|\!\uparrow\rangle$, $|\!\downarrow\rangle$

Quantum state space $\mathcal{H}_2 = \mathbb{C}^2$:

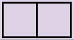
$$\left\{\alpha\,|\!\uparrow\rangle + \beta\,|\!\downarrow\rangle : |\alpha|^2 + |\beta|^2 = 1\right\}$$

of 2 dimensions

Two Bits vs Two Qubits

Two classical bits

Two bits can hold either of the 4 states in the state space
$\{0,1\} \times \{0,1\} = \{0,1\}^2 = \{\uparrow\uparrow, \downarrow\uparrow, \uparrow\downarrow, \downarrow\downarrow\}$.

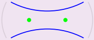


Two coupled qubits

A register of two coupled qubits can hold any of the states

$$|\Psi\rangle = \alpha\,|\uparrow\uparrow\rangle + \beta\,|\downarrow\uparrow\rangle + \gamma\,|\uparrow\downarrow\rangle + \delta\,|\downarrow\downarrow\rangle$$

in the state space $\mathcal{H}_2 \otimes \mathcal{H}_2 = \mathbb{C}^2 \otimes \mathbb{C}^2$.



Two separate qubits

Two separate qubits can hold any of the product states

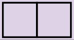
$$|\Psi_1\rangle \otimes |\Psi_2\rangle = (\alpha_1\,|\uparrow\rangle + \beta_1\,|\downarrow\rangle) \otimes (\alpha_2\,|\uparrow\rangle + \beta_2\,|\downarrow\rangle)$$

in the state space $\mathcal{H}_2 \oplus \mathcal{H}_2 \subset \mathbb{C}^2 \oplus \mathbb{C}^2$.

Two Bits vs Two Qubits

Two classical bits

Two bits can hold either of the 4 states in the state space $\{0,1\} \times \{0,1\} = \{0,1\}^2 = \{\uparrow\uparrow, \downarrow\uparrow, \uparrow\downarrow, \downarrow\downarrow\}$.

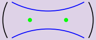


Two coupled qubits

A register of two coupled qubits can hold any of the states

$$|\Psi\rangle = \alpha\,|\uparrow\uparrow\rangle + \beta\,|\downarrow\uparrow\rangle + \gamma\,|\uparrow\downarrow\rangle + \delta\,|\downarrow\downarrow\rangle$$

in the state space $\mathcal{H}_2 \otimes \mathcal{H}_2 = \mathbb{C}^2 \otimes \mathbb{C}^2$.



Two separate qubits

Two separate qubits can hold any of the product states

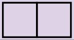
$$|\Psi_1\rangle \otimes |\Psi_2\rangle = (\alpha_1\,|\uparrow\rangle + \beta_1\,|\downarrow\rangle) \otimes (\alpha_2\,|\uparrow\rangle + \beta_2\,|\downarrow\rangle)$$

in the state space $\mathcal{H}_2 \oplus \mathcal{H}_2 \subset \mathbb{C}^2 \oplus \mathbb{C}^2$.

Two Bits vs Two Qubits

Two classical bits

Two bits can hold either of the 4 states in the state space
$\{0,1\} \times \{0,1\} = \{0,1\}^2 = \{\uparrow\uparrow, \downarrow\uparrow, \uparrow\downarrow, \downarrow\downarrow\}$.

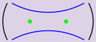


Two coupled qubits

A register of two coupled qubits can hold any of the states

$$|\Psi\rangle = \alpha\,|\uparrow\uparrow\rangle + \beta\,|\downarrow\uparrow\rangle + \gamma\,|\uparrow\downarrow\rangle + \delta\,|\downarrow\downarrow\rangle$$

in the state space $\mathcal{H}_2 \otimes \mathcal{H}_2 = \mathbb{C}^2 \otimes \mathbb{C}^2$.

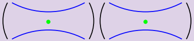


Two separate qubits

Two separate qubits can hold any of the product states

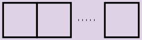
$$|\Psi_1\rangle \otimes |\Psi_2\rangle = (\alpha_1\,|\uparrow\rangle + \beta_1\,|\downarrow\rangle) \otimes (\alpha_2\,|\uparrow\rangle + \beta_2\,|\downarrow\rangle)$$

in the state space $\mathcal{H}_2 \oplus \mathcal{H}_2 \subset \mathbb{C}^2 \oplus \mathbb{C}^2$.



Many Bits vs Many Qubits

Classical register of n bits

 holds one of the 2^n states $\underline{\epsilon} = \epsilon_1 \cdots \epsilon_n$ of the state space $\{0,1\}^n = \{0,1\} \times \cdots \times \{0,1\}$.

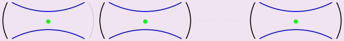
Quantum register of n qubits

 can hold any coherent superposition

$$|\Psi\rangle = \sum_{\underline{\epsilon} \in \{0,1\}^n} \alpha_{\epsilon_1 \cdots \epsilon_n} |\epsilon_1\rangle \otimes |\epsilon_2\rangle \otimes \cdots \otimes |\epsilon_n\rangle$$

in the 2^n dimensional space $\mathcal{H}_{2^n} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = \mathbb{C}^{2^n}$.

Product states of n qubits

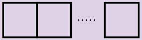
 can only hold a product state

$$|\Psi_1\rangle \otimes |\Psi_2\rangle \otimes \cdots \otimes |\Psi_n\rangle = (\alpha_1 |\uparrow\rangle + \beta_1 |\downarrow\rangle) \otimes \cdots \otimes (\alpha_n |\uparrow\rangle + \beta_n |\downarrow\rangle)$$

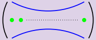
(thus, only linear scaling of system and dimension)

Many Bits vs Many Qubits

Classical register of n bits

 holds one of the 2^n states $\underline{\epsilon} = \epsilon_1 \cdots \epsilon_n$ of the state space $\{0,1\}^n = \{0,1\} \times \cdots \times \{0,1\}$.

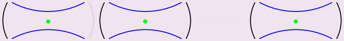
Quantum register of n qubits

 can hold any coherent superposition

$$|\Psi\rangle = \sum_{\underline{\epsilon} \in \{0,1\}^n} \alpha_{\epsilon_1 \cdots \epsilon_n} |\epsilon_1\rangle \otimes |\epsilon_2\rangle \otimes \cdots \otimes |\epsilon_n\rangle$$

in the 2^n dimensional space $\mathcal{H}_{2^n} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = \mathbb{C}^{2^n}$.

Product states of n qubits

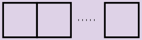
 can only hold a product state

$$|\Psi_1\rangle \otimes |\Psi_2\rangle \otimes \cdots \otimes |\Psi_n\rangle = (\alpha_1 |\uparrow\rangle + \beta_1 |\downarrow\rangle) \otimes \cdots \otimes (\alpha_n |\uparrow\rangle + \beta_n |\downarrow\rangle)$$

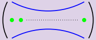
(thus, only linear scaling of system and dimension)

Many Bits vs Many Qubits

Classical register of n bits

 holds one of the 2^n states $\underline{\epsilon} = \epsilon_1 \cdots \epsilon_n$ of the state space $\{0,1\}^n = \{0,1\} \times \cdots \times \{0,1\}$.

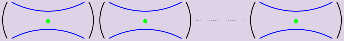
Quantum register of n qubits

 can hold any coherent superposition

$$|\Psi\rangle = \sum_{\underline{\epsilon} \in \{0,1\}^n} \alpha_{\epsilon_1 \cdots \epsilon_n} |\epsilon_1\rangle \otimes |\epsilon_2\rangle \otimes \cdots \otimes |\epsilon_n\rangle$$

in the 2^n dimensional space $\mathcal{H}_{2^n} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = \mathbb{C}^{2^n}$.

Product states of n qubits

 can only hold a product state

$$|\Psi_1\rangle \otimes |\Psi_2\rangle \otimes \cdots \otimes |\Psi_n\rangle = (\alpha_1 |\uparrow\rangle + \beta_1 |\downarrow\rangle) \otimes \cdots \otimes (\alpha_n |\uparrow\rangle + \beta_n |\downarrow\rangle)$$

(thus, only linear scaling of system and dimension)

Our Model of Measurements

von Neumann measurements of one qubit

First, specify a basis B for $\mathbb{C}^2$, e. g. $\{|0\rangle, |1\rangle\}$. The outcome of measuring the state $\alpha|0\rangle + \beta|1\rangle$ is described by a random variable X. The probabilities to observe "0" or "1" are given by

$$\Pr(X = 0) = |\alpha|^2, \quad \Pr(X = 1) = |\beta|^2.$$

Measuring a state in $\mathbb{C}^n$ in an orthonormal basis B

- Recall: Orthonormal Basis of $\mathbb{C}^N$

$$B = \{|\psi_i\rangle : i = 1, \ldots, N\}, \quad \text{where } \langle\psi_i|\psi_j\rangle = \delta_{i,j}$$

- Let $|\varphi\rangle = \sum_{i=1}^N \alpha_i |i\rangle$, where $\sum_{i=1}^N |\alpha_i|^2 = 1$. Then measuring $|\varphi\rangle$ in the basis B gives random variable X_B taking values $1, \ldots, N$:

$$\Pr(X_B = 1) = |\langle\psi_1|\varphi\rangle|^2, \ \ldots \ , \Pr(X_B = N) = |\langle\psi_N|\varphi\rangle|^2.$$

Our Model of Measurements

von Neumann measurements of one qubit

First, specify a basis B for $\mathbb{C}^2$, e. g. $\{|0\rangle, |1\rangle\}$. The outcome of measuring the state $\alpha|0\rangle + \beta|1\rangle$ is described by a random variable X. The probabilities to observe "0" or "1" are given by

$$\Pr(X = 0) = |\alpha|^2, \quad \Pr(X = 1) = |\beta|^2.$$

Measuring a state in $\mathbb{C}^n$ in an orthonormal basis B

- Recall: Orthonormal Basis of $\mathbb{C}^N$

$$B = \{|\psi_i\rangle : i = 1, \dots, N\}, \quad \text{where } \langle\psi_i|\psi_j\rangle = \delta_{i,j}$$

- Let $|\varphi\rangle = \sum_{i=1}^N \alpha_i |i\rangle$, where $\sum_{i=1}^N |\alpha_i|^2 = 1$. Then measuring $|\varphi\rangle$ in the basis B gives random variable X_B taking values $1, \dots, N$:

$$\Pr(X_B = 1) = |\langle\psi_1|\varphi\rangle|^2, \ \dots\ , \Pr(X_B = N) = |\langle\psi_N|\varphi\rangle|^2.$$

Two Important Types of Operations

Local operations

$$\mathbf{1}_N \otimes U = \begin{pmatrix} \boxed{U} & & & \\ & \boxed{U} & & \\ & & \ddots & \\ & & & \boxed{U} \end{pmatrix}, \quad \text{where} \quad U \in \mathcal{U}(2).$$

Conditioned operation: the controlled NOT (CNOT)

$$\boxed{\begin{array}{l} |00\rangle \mapsto |00\rangle \\ |01\rangle \mapsto |01\rangle \\ |10\rangle \mapsto |11\rangle \\ |11\rangle \mapsto |10\rangle \end{array}} \;\widehat{=}\; \begin{pmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 1 & \cdot \end{pmatrix} \;\widehat{=}$$

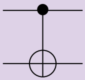
Two Important Types of Operations

Local operations

$$\mathbf{1}_N \otimes U \quad = \begin{pmatrix} \boxed{U} & & & \\ & \boxed{U} & & \\ & & \ddots & \\ & & & \boxed{U} \end{pmatrix}, \quad \text{where} \quad U \in \mathcal{U}(2).$$

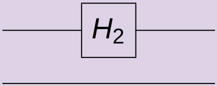
Conditioned operation: the controlled NOT (CNOT)

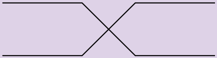
$$\boxed{\begin{array}{c} |00\rangle \mapsto |00\rangle \\ |01\rangle \mapsto |01\rangle \\ |10\rangle \mapsto |11\rangle \\ |11\rangle \mapsto |10\rangle \end{array}} \quad \widehat{=} \quad \begin{pmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 1 & \cdot \end{pmatrix} \quad \widehat{=}$$

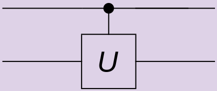


Notation for Quantum Gates

Gate in Feynman notation







Corresponding transformation

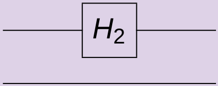
$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ 1 & \cdot & -1 & \cdot \\ \cdot & 1 & \cdot & -1 \end{bmatrix}$$

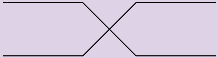
$$\begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

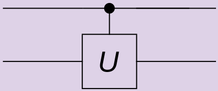
$$\begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & u_{11} & u_{12} \\ \cdot & \cdot & u_{21} & u_{22} \end{bmatrix}$$

Notation for Quantum Gates

Gate in Feynman notation







Corresponding transformation

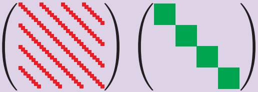
$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ 1 & \cdot & -1 & \cdot \\ \cdot & 1 & \cdot & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & u_{11} & u_{12} \\ \cdot & \cdot & u_{21} & u_{22} \end{bmatrix}$$

Two Constructions for Matrices

Tensor product

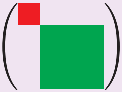
$T_1 \otimes T_2 = (T_1 \otimes I)(I \otimes T_2) =$ 

$I \otimes T_2$ classically parallel

$T_1 \otimes T_2$ quantum parallel $\Longrightarrow$ independent dynamics

classical: expensive, quantum: easy, since local operations

Direct sum

$T_1 \oplus T_2 = (T_1 \oplus I)(I \oplus T_2) =$ 

$I \oplus T_2$ conditional operation

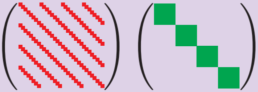
$T_1 \oplus T_2$ CASE operator $\Longrightarrow$ coupled dynamics

classical: easy, quantum: difficult, since entangled

Two Constructions for Matrices

Tensor product

$$\color{red}{T_1} \otimes \color{green}{T_2} = (\color{red}{T_1} \otimes I)(I \otimes \color{green}{T_2}) =$$



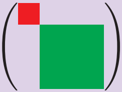
$I \otimes \color{green}{T_2}$ classically parallel

$\color{red}{T_1} \otimes \color{green}{T_2}$ quantum parallel $\Longrightarrow$ independent dynamics

classical: expensive, quantum: easy, since local operations

Direct sum

$$\color{red}{T_1} \oplus \color{green}{T_2} = (\color{red}{T_1} \oplus I)(I \oplus \color{green}{T_2}) =$$



$I \oplus \color{red}{T_2}$ conditional operation

$\color{red}{T_1} \oplus \color{green}{T_2}$ CASE operator $\Longrightarrow$ coupled dynamics

classical: easy, quantum: difficult, since entangled

The Hadamard transform on one qubit

$$H_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}}\sum_{x,y\in\mathbf{F}_2}(-1)^{xy}\,|x\rangle\langle y|\,.$$

The Hadamard transform on n qubits

The n-fold tensor product of H_2 is given by

$$H_2^{\otimes n} = \frac{1}{2^{n/2}}\sum_{x,y\in\mathbf{F}_2^n}(-1)^{x\cdot y}\,|x\rangle\langle y|\,,$$

where $x \cdot y := \sum_{i=1}^{n} x_i y_i \in \mathbf{F}_2$.

The Hadamard Transform

The Hadamard transform on one qubit

$$H_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \sum_{x,y \in \mathbf{F}_2} (-1)^{xy} |x\rangle \langle y| .$$

The Hadamard transform on n qubits

The n-fold tensor product of H_2 is given by

$$H_2^{\otimes n} = \frac{1}{2^{n/2}} \sum_{x,y \in \mathbf{F}_2^n} (-1)^{x \cdot y} |x\rangle \langle y| ,$$

where $x \cdot y := \sum_{i=1}^n x_i y_i \in \mathrm{F}_2$.

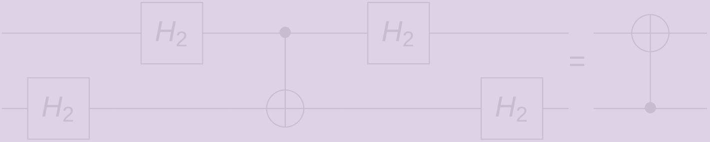
Properties of the Hadamard transform

- By definition H_2 is the following map:

$$|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- Identity involving the Pauli matrices: $H_2 \sigma_x H_2 = \sigma_z$
- Identity involving the CNOT gate:



The Hadamard Transform

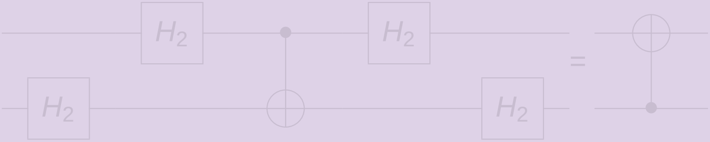
Properties of the Hadamard transform

- By definition H_2 is the following map:

$$|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- Identity involving the Pauli matrices: $H_2 \sigma_x H_2 = \sigma_z$
- Identity involving the CNOT gate:



The Hadamard Transform

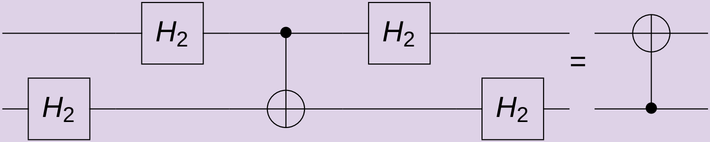
Properties of the Hadamard transform

- By definition H_2 is the following map:

$$|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

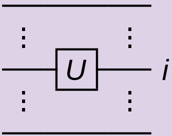
$$|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

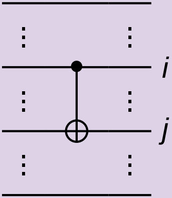
- Identity involving the Pauli matrices: $H_2 \sigma_x H_2 = \sigma_z$
- Identity involving the CNOT gate:



Quantum Gates and Circuits

Elementary quantum gates

$U^{(i)} =$ 

$\mathrm{CNOT}^{(i,j)} =$ 

Universal set of gates

Theorem (Barenco et al., 1995):

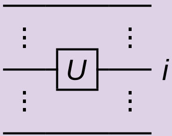
$$\mathcal{U}(2^n) = \langle U^{(i)}, \mathrm{CNOT}^{(i,j)} \quad : \quad i,j = 1,\ldots,n, \quad i \neq j \rangle$$

Quantum gates: main problem

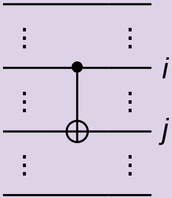
Find efficient factorizations for given $U \in \mathcal{U}(2^n)$!

Elementary quantum gates

$U^{(i)} =$



$\mathrm{CNOT}^{(i,j)} =$



Universal set of gates

Theorem (Barenco et al., 1995):

$$\mathcal{U}(2^n) = \langle U^{(i)}, \mathrm{CNOT}^{(i,j)} \quad : \quad i,j = 1,\ldots,n, \quad i \neq j \rangle$$

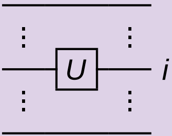
Quantum gates: main problem

Find efficient factorizations for given $U \in \mathcal{U}(2^n)$!

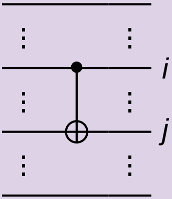
Quantum Gates and Circuits

Elementary quantum gates

$U^{(i)} =$



$\mathrm{CNOT}^{(i,j)} =$



Universal set of gates

Theorem (Barenco et al., 1995):

$$\mathcal{U}(2^n) = \langle U^{(i)}, \mathrm{CNOT}^{(i,j)} \quad : \quad i,j = 1,\ldots,n, \quad i \neq j \rangle$$

Quantum gates: main problem

Find efficient factorizations for given $U \in \mathcal{U}(2^n)$!

Different Levels of Abstraction

Unitary matrix

$$U = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

Factorized unitary matrix

$$U = \quad (I \otimes H_2) \quad (I \oplus \sigma_x) \quad (H_2 \otimes I)$$

$$= \frac{1}{2}\begin{pmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{pmatrix} \left(\begin{array}{c|c} I & \\ \hline & \sigma_x \end{array}\right) \begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{pmatrix}$$

Quantum circuit

Different Levels of Abstraction

Unitary matrix

$$U = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

Factorized unitary matrix

$$U = \quad (I \otimes H_2) \quad (I \oplus \sigma_x) \quad (H_2 \otimes I)$$

$$= \frac{1}{2}\begin{pmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{pmatrix}\left(\begin{array}{c|c} I & \\ \hline & \sigma_x \end{array}\right)\begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{pmatrix}$$

Quantum circuit

Different Levels of Abstraction

Unitary matrix

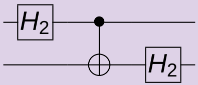
$$U = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix}$$

Factorized unitary matrix

$$U = \quad (I \otimes H_2) \quad (I \oplus \sigma_x) \quad (H_2 \otimes I)$$

$$= \frac{1}{2}\begin{pmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{pmatrix}\left(\begin{array}{c|c} I & \\ \hline & \sigma_x \end{array}\right)\begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{pmatrix}$$

Quantum circuit



Basic Facts About Boolean Functions

Boolean functions

- Any function $f : \{0,1\}^n \to \{0,1\}^m$, where n is the number of input bits and m the number of output bits is said to be a Boolean function.
- Any Boolean function can be represented by a truth table. If $f = (f_1, \ldots, f_m)$, this is a matrix of size $2^n \times m$ where in column i we have the list of values $f_i(x_1, \ldots, x_n)$, where $x_j \in \{0,1\}$ for $j = 1, \ldots, n$.

The number of Boolean functions

- There are 2^{2^n} Boolean functions $f : \{0,1\}^n \to \{0,1\}$, i. e., functions with n inputs and one output (since we can assign an arbitrary value for each of the 2^n inputs).
- There are 2^{m2^n} Boolean functions $f : \{0,1\}^n \to \{0,1\}^m$with n inputs and m outputs.

Basic Facts About Boolean Functions

Boolean functions

- Any function $f : \{0,1\}^n \rightarrow \{0,1\}^m$, where n is the number of input bits and m the number of output bits is said to be a Boolean function.
- Any Boolean function can be represented by a truth table. If $f = (f_1, \ldots, f_m)$, this is a matrix of size $2^n \times m$ where in column i we have the list of values $f_i(x_1, \ldots, x_n)$, where $x_j \in \{0,1\}$ for $j = 1, \ldots, n$.

The number of Boolean functions

- There are 2^{2^n} Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$, i. e., functions with n inputs and one output (since we can assign an arbitrary value for each of the 2^n inputs).
- There are 2^{m2^n} Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}^m$with n inputs and m outputs.

Universal Gates: Classical Computing

Connectives

NOT gate:

x_1	$\overline{x_1}$
0	1
1	0

AND gate:

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

OR gate:

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

NAND gate:

x_1	x_2	$\overline{x_1 \wedge x_2}$
0	0	1
0	1	1
1	0	1
1	1	0

Theorem

Any deterministic classical circuit and thereby any deterministic classical computation can be realized by using NAND gates only. Any probabilistic computation can be realized using NAND gates and in addition one gate which realizes a fair coin flip.

Universal Gates: Classical Computing

Connectives

NOT gate:

x_1	$\overline{x_1}$
0	1
1	0

AND gate:

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

OR gate:

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

NAND gate:

x_1	x_2	$\overline{x_1 \wedge x_2}$
0	0	1
0	1	1
1	0	1
1	1	0

Theorem

Any deterministic classical circuit and thereby any deterministic classical computation can be realized by using NAND gates only. Any probabilistic computation can be realized using NAND gates and in addition one gate which realizes a fair coin flip.

Finite field of two elements

- The set $\{0, 1\}$ can be equipped with a multiplication "$\cdot$" and an addition "$\oplus$" such that the field axioms hold.
- Truth tables for these operations:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

- Then $(\{0, 1\}, \cdot, \oplus)$ becomes a field consisting of two elements only, also denoted by F_2. A finite field with n elements exists if and only if n is a prime power.
- Important identity: $(-1)^{x \oplus y} = (-1)^x \cdot (-1)^y$
- We can also rewrite CNOT: $|x\rangle |y\rangle \mapsto |x\rangle |x \oplus y\rangle$

Loading More Structure Onto Bits

Finite field of two elements

- The set $\{0,1\}$ can be equipped with a multiplication "$\cdot$" and an addition "$\oplus$" such that the field axioms hold.
- Truth tables for these operations:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

- Then $(\{0,1\}, \cdot, \oplus)$ becomes a field consisting of two elements only, also denoted by F_2. A finite field with n elements exists if and only if n is a prime power.
- Important identity: $(-1)^{x \oplus y} = (-1)^x \cdot (-1)^y$
- We can also rewrite CNOT: $|x\rangle\,|y\rangle \mapsto |x\rangle\,|x \oplus y\rangle$

Loading More Structure Onto Bits

Finite field of two elements

- The set $\{0, 1\}$ can be equipped with a multiplication "$\cdot$" and an addition "$\oplus$" such that the field axioms hold.
- Truth tables for these operations:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

- Then $(\{0, 1\}, \cdot, \oplus)$ becomes a field consisting of two elements only, also denoted by F_2. A finite field with n elements exists if and only if n is a prime power.
- Important identity: $(-1)^{x \oplus y} = (-1)^x \cdot (-1)^y$
- We can also rewrite CNOT: $|x\rangle |y\rangle \mapsto |x\rangle |x \oplus y\rangle$

Loading More Structure Onto Bits

Finite field of two elements

- The set $\{0,1\}$ can be equipped with a multiplication "$\cdot$" and an addition "$\oplus$" such that the field axioms hold.
- Truth tables for these operations:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

- Then $(\{0,1\}, \cdot, \oplus)$ becomes a field consisting of two elements only, also denoted by F_2. A finite field with n elements exists if and only if n is a prime power.
- Important identity: $(-1)^{x \oplus y} = (-1)^x \cdot (-1)^y$
- We can also rewrite CNOT: $|x\rangle\,|y\rangle \mapsto |x\rangle\,|x \oplus y\rangle$

Loading More Structure Onto Bits

Finite field of two elements

- The set $\{0, 1\}$ can be equipped with a multiplication "$\cdot$" and an addition "$\oplus$" such that the field axioms hold.
- Truth tables for these operations:

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

- Then $(\{0, 1\}, \cdot, \oplus)$ becomes a field consisting of two elements only, also denoted by F_2. A finite field with n elements exists if and only if n is a prime power.
- Important identity: $(-1)^{x \oplus y} = (-1)^x \cdot (-1)^y$
- We can also rewrite CNOT: $|x\rangle\,|y\rangle \mapsto |x\rangle\,|x \oplus y\rangle$

Reversible Computation of Boolean Functions

Basic issue of reversible computing

Suppose, we want to compute a function $f : \{0,1\}^n \to \{0,1\}^m$ that is not reversible. How can we do this?

One possible solution

Define a new Boolean function which takes $n+m$ inputs and $n+m$ outputs as follows:

$$F(x,y) := (x, y \oplus f(x))$$

Properties of $F(x,y)$

- On the special inputs $(x,0)$, where $x \in \{0,1\}^n$ we obtain that $F(x,0) = (x, f(x))$. Furthermore, F is reversible.
- Theorem (Bennett): If f can be computed using K gates, then F can be computed using $2K+m$ gates.

Reversible Computation of Boolean Functions

Basic issue of reversible computing

Suppose, we want to compute a function $f : \{0,1\}^n \to \{0,1\}^m$ that is not reversible. How can we do this?

One possible solution

Define a new Boolean function which takes $n+m$ inputs and $n+m$ outputs as follows:

$$F(x,y) := (x, y \oplus f(x))$$

Properties of $F(x,y)$

- On the special inputs $(x,0)$, where $x \in \{0,1\}^n$ we obtain that $F(x,0) = (x, f(x))$. Furthermore, F is reversible.
- Theorem (Bennett): If f can be computed using K gates, then F can be computed using $2K+m$ gates.

Reversible Computation of Boolean Functions

Basic issue of reversible computing

Suppose, we want to compute a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ that is not reversible. How can we do this?

One possible solution

Define a new Boolean function which takes $n+m$ inputs and $n+m$ outputs as follows:

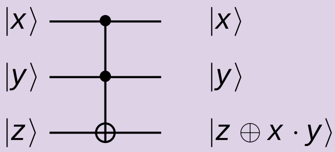
$$F(x,y) := (x, y \oplus f(x))$$

Properties of $F(x,y)$

- On the special inputs $(x,0)$, where $x \in \{0,1\}^n$ we obtain that $F(x,0) = (x, f(x))$. Furthermore, F is reversible.
- Theorem (Bennett): If f can be computed using K gates, then F can be computed using $2K+m$ gates.

The Toffoli gate "TOF"

x	y	z	x'	y'	z'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



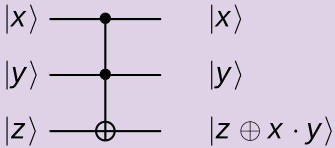
Theorem (Toffoli, 1981)

Any reversible computation can be realized by using TOF gates and ancilla (auxiliary) bits which are initialized to 0.

Universal Gates: Reversible Computing

The Toffoli gate “TOF”

x	y	z	x'	y'	z'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



Theorem (Toffoli, 1981)

Any reversible computation can be realized by using TOF gates and ancilla (auxiliary) bits which are initialized to 0.

Reversible Computation of Boolean Functions

More features of the reversible embedding $F(x,y)$

- Possibly, there are reversible functions that compute $G(x,0) = (f(x), \mathrm{junk}(x))$ and use use fewer than $n+m$ bits.
- Actually, only $\lceil \log_2(\max c_y) \rceil$ many extra bits are needed to make f reversible, where $c_y = |\{x : f(x) = y\}|$ are the sizes of the collisions of f.
- The advantage of F is its uniform definition.

Computing a function by a quantum circuit

Any function $f : \{0,1\}^n \to \{0,1\}^m$ can be computed by means of the reversible function $F : \{0,1\}^{n+m} \to \{0,1\}^{n+m}$. Hence we can compute f also by a quantum circuit

$$U_f \left|x\right\rangle \left|y\right\rangle = \left|x\right\rangle \left|y \oplus f(x)\right\rangle.$$

Reversible Computation of Boolean Functions

More features of the reversible embedding $F(x,y)$

- Possibly, there are reversible functions that compute $G(x,0) = (f(x), \mathrm{junk}(x))$ and use use fewer than $n+m$ bits.
- Actually, only $\lceil \log_2(\max c_y) \rceil$ many extra bits are needed to make f reversible, where $c_y = |\{x : f(x) = y\}|$ are the sizes of the collisions of f.
- The advantage of F is its uniform definition.

Computing a function by a quantum circuit

Any function $f : \{0,1\}^n \to \{0,1\}^m$ can be computed by means of the reversible function $F : \{0,1\}^{n+m} \to \{0,1\}^{n+m}$. Hence we can compute f also by a quantum circuit

$$U_f \left|x\right\rangle \left|y\right\rangle = \left|x\right\rangle \left|y \oplus f(x)\right\rangle.$$

A Two-Bit Problem

The Deutsch Jozsa problem (most simple case)

Given a Boolean function $f : \{0,1\} \to \{0,1\}$. Decide whether the property $f(0) = f(1)$ holds or not.

The four possible functions

x	$f(x)$
0	0
1	0

x	$f(x)$
0	1
1	1

x	$f(x)$
0	0
1	1

x	$f(x)$
0	1
1	0

Observation

- From querying the function only on one input, we cannot determine whether $f(0) = f(1)$ with certainty. E. g., if the answer is $f(0) = 0$, it could be the first or third function.
- On the other hand, two queries determine the function completely. What is the quantum query complexity?

A Two-Bit Problem

The Deutsch Jozsa problem (most simple case)

Given a Boolean function $f : \{0,1\} \rightarrow \{0,1\}$. Decide whether the property $f(0) = f(1)$ holds or not.

The four possible functions

x	$f(x)$
0	0
1	0

x	$f(x)$
0	1
1	1

x	$f(x)$
0	0
1	1

x	$f(x)$
0	1
1	0

Observation

- From querying the function only on one input, we cannot determine whether $f(0) = f(1)$ with certainty. E. g., if the answer is $f(0) = 0$, it could be the first or third function.
- On the other hand, two queries determine the function completely. What is the quantum query complexity?

A Two-Bit Problem

The Deutsch Jozsa problem (most simple case)

Given a Boolean function $f : \{0,1\} \to \{0,1\}$. Decide whether the property $f(0) = f(1)$ holds or not.

The four possible functions

x	$f(x)$
0	0
1	0

x	$f(x)$
0	1
1	1

x	$f(x)$
0	0
1	1

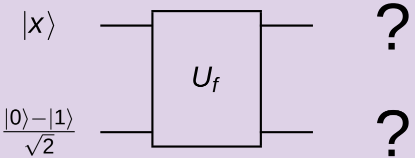
x	$f(x)$
0	1
1	0

Observation

- From querying the function only on one input, we cannot determine whether $f(0) = f(1)$ with certainty. E. g., if the answer is $f(0) = 0$, it could be the first or third function.
- On the other hand, two queries determine the function completely. What is the quantum query complexity?

The Phase “Kick-Back” Trick

Computing the function into the phase

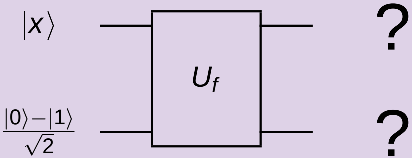


Computing the effect for different inputs

$$\begin{aligned}
|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad &\mapsto \quad |x\rangle \left(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) \\
&= \quad |x\rangle (-1)^{f(x)} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \\
&= \quad (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)
\end{aligned}$$

The Phase “Kick-Back” Trick

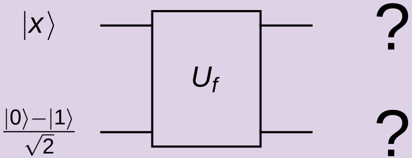
Computing the function into the phase



Computing the effect for different inputs

$$
\begin{aligned}
|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad \mapsto \quad & |x\rangle \left(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) \\
= \quad & |x\rangle (-1)^{f(x)} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \\
= \quad & (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)
\end{aligned}
$$

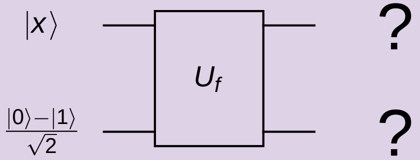
Computing the function into the phase



Computing the effect for different inputs

$$
\begin{aligned}
|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad \mapsto \quad & |x\rangle \left(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) \\
= \quad & |x\rangle (-1)^{f(x)} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \\
= \quad & (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)
\end{aligned}
$$

Computing the function into the phase

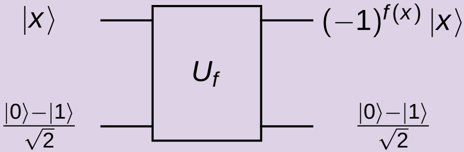


Computing the effect for different inputs

$$
\begin{aligned}
|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} &\mapsto |x\rangle \left(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) \\
&= |x\rangle (-1)^{f(x)} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \\
&= (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)
\end{aligned}
$$

The Phase "Kick-Back" Trick

Computing the function into the phase

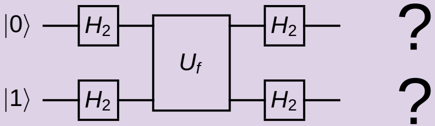


Computing the effect for different inputs

$$\begin{aligned}
|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad &\mapsto \quad |x\rangle \left(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) \\
&= \quad |x\rangle (-1)^{f(x)} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \\
&= \quad (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)
\end{aligned}$$

The Phase "Kick-Back" Trick

The quantum algorithm

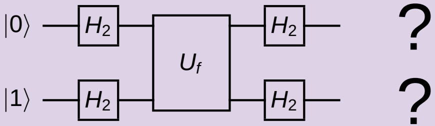


Phase kick-back in superposition

$$
\begin{aligned}
|0\rangle\,|1\rangle &\xmapsto{H_2^{\otimes 2}} \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\xmapsto{U_f} \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&= \frac{(-1)^{f(0)}}{\sqrt{2}}\left(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\xmapsto{H_2^{\otimes 2}} (-1)^{f(0)}\,|f(0)\oplus f(1)\rangle\,|1\rangle
\end{aligned}
$$

The Phase "Kick-Back" Trick

The quantum algorithm



Phase kick-back in superposition

$$|0\rangle\,|1\rangle \overset{H_2^{\otimes 2}}{\longmapsto} \left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

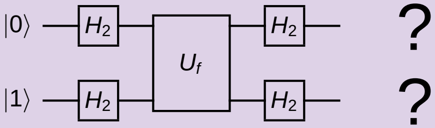
$$\overset{U_f}{\longmapsto} \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

$$= \frac{(-1)^{f(0)}}{\sqrt{2}}\left(|0\rangle+(-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

$$\overset{H_2^{\otimes 2}}{\longmapsto} (-1)^{f(0)}\,|f(0)\oplus f(1)\rangle\,|1\rangle$$

The Phase “Kick-Back” Trick

The quantum algorithm

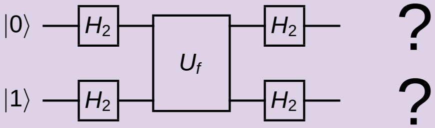


Phase kick-back in superposition

$$
\begin{aligned}
|0\rangle\,|1\rangle &\overset{H_2^{\otimes 2}}{\longmapsto} \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{U_f}{\longmapsto} \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&= \frac{(-1)^{f(0)}}{\sqrt{2}}\left(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{H_2^{\otimes 2}}{\longmapsto} (-1)^{f(0)}\,|f(0)\oplus f(1)\rangle\,|1\rangle
\end{aligned}
$$

The Phase “Kick-Back” Trick

The quantum algorithm

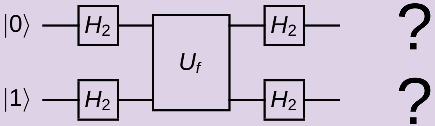


Phase kick-back in superposition

$$
\begin{aligned}
|0\rangle\,|1\rangle \;&\overset{H_2^{\otimes 2}}{\longmapsto}\; \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\\
&\overset{U_f}{\longmapsto}\; \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\\
&=\; \frac{(-1)^{f(0)}}{\sqrt{2}}\left(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\\
&\overset{H_2^{\otimes 2}}{\longmapsto}\; (-1)^{f(0)}\,|f(0)\oplus f(1)\rangle\,|1\rangle
\end{aligned}
$$

The Phase “Kick-Back” Trick

The quantum algorithm

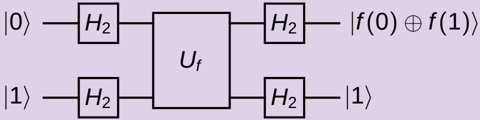


Phase kick-back in superposition

$$
\begin{aligned}
|0\rangle\,|1\rangle \;&\overset{H_2^{\otimes 2}}{\longmapsto}\; \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{U_f}{\longmapsto}\; \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&= \frac{(-1)^{f(0)}}{\sqrt{2}}\left(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{H_2^{\otimes 2}}{\longmapsto}\; (-1)^{f(0)}\,|f(0) \oplus f(1)\rangle\,|1\rangle
\end{aligned}
$$

The Phase "Kick-Back" Trick

The quantum algorithm



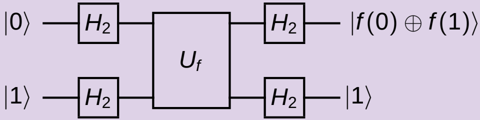
Phase kick-back in superposition

$$
\begin{aligned}
|0\rangle\,|1\rangle \;&\overset{H_2^{\otimes 2}}{\longmapsto}\; \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{U_f}{\longmapsto}\; \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&= \frac{(-1)^{f(0)}}{\sqrt{2}}\left(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{H_2^{\otimes 2}}{\longmapsto}\; (-1)^{f(0)}\,|f(0) \oplus f(1)\rangle\,|1\rangle
\end{aligned}
$$

Hence, we from measuring the first qubit in the computational basis, we obtain the answer $f(0) \oplus f(1)$ which reveals whether $f(0) = f(1)$ or not.

The Phase “Kick-Back” Trick

The quantum algorithm



Phase kick-back in superposition

$$
\begin{aligned}
|0\rangle\,|1\rangle &\overset{H_2^{\otimes 2}}{\longmapsto} \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{U_f}{\longmapsto} \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + \frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&= \frac{(-1)^{f(0)}}{\sqrt{2}}\left(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\overset{H_2^{\otimes 2}}{\longmapsto} (-1)^{f(0)}\,|f(0)\oplus f(1)\rangle\,|1\rangle
\end{aligned}
$$

Hence, we from measuring the first qubit in the computational basis, we obtain the answer $f(0) \oplus f(1)$ which reveals whether $f(0) = f(1)$ or not.

What can be computed?

The strong Church-Turing thesis

Any reasonable algorithmic process carried out by a physical machine can be efficiently simulated by a probabilistic Turing machine. The slow-down for this is at most polynomial.

Efficient computations

Given a problem of size n, an algorithm is said to have a polynomial running time if the number of steps it needs to find a solution is bounded by $p(n)$, where p is a polynomial.

How do quantum algorithms fit in?

They can solve problems which are believed to be intractable for classical computers. There are physically reasonable algorithmic processes carried which seem to be hard to simulate for any classical probabilistic Turing machine.

What can be computed?

The strong Church-Turing thesis

Any reasonable algorithmic process carried out by a physical machine can be efficiently simulated by a probabilistic Turing machine. The slow-down for this is at most polynomial.

Efficient computations

Given a problem of size n, an algorithm is said to have a polynomial running time if the number of steps it needs to find a solution is bounded by $p(n)$, where p is a polynomial.

How do quantum algorithms fit in?

They can solve problems which are believed to be intractable for classical computers. There are physically reasonable algorithmic processes carried which seem to be hard to simulate for any classical probabilistic Turing machine.

What can be computed?

The strong Church-Turing thesis

Any reasonable algorithmic process carried out by a physical machine can be efficiently simulated by a probabilistic Turing machine. The slow-down for this is at most polynomial.

Efficient computations

Given a problem of size n, an algorithm is said to have a polynomial running time if the number of steps it needs to find a solution is bounded by $p(n)$, where p is a polynomial.

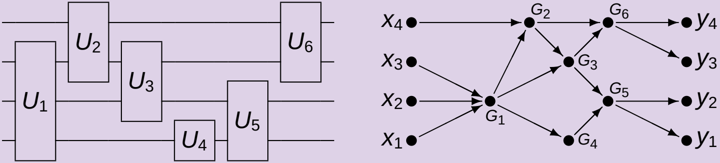
How do quantum algorithms fit in?

They can solve problems which are believed to be intractable for classical computers. There are physically reasonable algorithmic processes carried which seem to be hard to simulate for any classical probabilistic Turing machine.

Computational Model: Quantum Circuits

Example: quantum circuit

Quantum circuit and corresponding directed acyclic graph.



Uniform families of quantum circuits

A family $\mathcal{F} := \{\mathcal{C}_n \in \mathcal{U}(2^n) \mid n \in \mathbb{N}\}$ of quantum circuits is called **uniform** if there exists a polynomial-time deterministic Turing machine which computes $n \mapsto \mathcal{C}_n$, where n is the problem size.

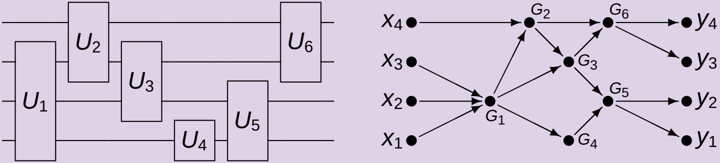
Theorem (Yao '93)

Uniform families of quantum circuits and quantum Turing machines (see Bernstein/Vazirani '93) are equivalent.

Computational Model: Quantum Circuits

Example: quantum circuit

Quantum circuit and corresponding directed acyclic graph.



Uniform families of quantum circuits

A family $\mathcal{F} := \{\mathcal{C}_n \in \mathcal{U}(2^n) \mid n \in \mathbb{N}\}$ of quantum circuits is called **uniform** if there exists a polynomial-time deterministic Turing machine which computes $n \mapsto \mathcal{C}_n$, where n is the problem size.

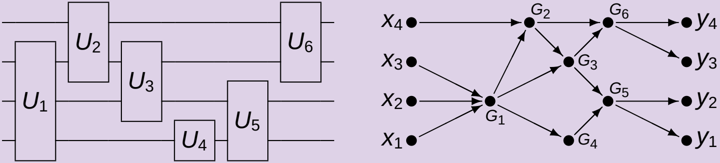
Theorem (Yao '93)

Uniform families of quantum circuits and quantum Turing machines (see Bernstein/Vazirani '93) are equivalent.

Computational Model: Quantum Circuits

Example: quantum circuit

Quantum circuit and corresponding directed acyclic graph.



Uniform families of quantum circuits

A family $\mathcal{F} := \{\mathcal{C}_n \in \mathcal{U}(2^n) \mid n \in \mathbb{N}\}$ of quantum circuits is called **uniform** if there exists a polynomial-time deterministic Turing machine which computes $n \mapsto \mathcal{C}_n$, where n is the problem size.

Theorem (Yao '93)

Uniform families of quantum circuits and quantum Turing machines (see Bernstein/Vazirani '93) are equivalent.

Asymptotics of Functions

Landau notation

- We use Landau notation to compare the asymptotics of two functions $f, g : \mathbb{N} \to \mathbb{C}$. Typically, f is the running time for an algorithm for input of size n and g is another function.
- $f(n) = O(g(n))$ means that for some m there exists a constant $c > 0$, such that $|f(n)| \leq cg(n)$ for all $n \geq m$.
- $f(n) = \Omega(g(n))$ means that for some m there exists a constant $c > 0$, such that $|f(n)| \geq cg(n)$ for all $n \geq m$.
- $f(n) = \Theta(g(n))$ means that both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ hold.

Example

Let $f(n)$ be number of operations needed to compute a classical FFT of a vector of length n. Then $f(n) = O(n \log(n))$.

Asymptotics of Functions

Landau notation

- We use Landau notation to compare the asymptotics of two functions $f, g : \mathbb{N} \to \mathbb{C}$. Typically, f is the running time for an algorithm for input of size n and g is another function.
- $f(n) = O(g(n))$ means that for some m there exists a constant $c > 0$, such that $|f(n)| \leq cg(n)$ for all $n \geq m$.
- $f(n) = \Omega(g(n))$ means that for some m there exists a constant $c > 0$, such that $|f(n)| \geq cg(n)$ for all $n \geq m$.
- $f(n) = \Theta(g(n))$ means that both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ hold.

Example

Let $f(n)$ be number of operations needed to compute a classical FFT of a vector of length n. Then $f(n) = O(n \log(n))$.

Computational Complexity

Measuring the problem size

- Usually algorithms can use different kinds of resources. Typical examples for resources which can be measured are **time**, **space**, **depth**, and **queries**.
- The resources needed are always measured as a function of the input size.

Examples

- Consider the problem of multiplying two n-bit numbers. Clearly, this can be done in $O(n^2)$ additions and multiplications using the school method. The best known method uses $O(n \log n \log \log n)$ operations.
- The converse problem of factoring an n-bit number N into its prime factors is much harder. The currently best known algorithm needs $\exp\left(c(\log N)^{1/3}(\log \log N)^{2/3}\right)$ operations.

Computational Complexity

Measuring the problem size

- Usually algorithms can use different kinds of resources. Typical examples for resources which can be measured are **time**, **space**, **depth**, and **queries**.
- The resources needed are always measured as a function of the input size.

Examples

- Consider the problem of multiplying two n-bit numbers. Clearly, this can be done in $O(n^2)$ additions and multiplications using the school method. The best known method uses $O(n \log n \log\log n)$ operations.
- The converse problem of factoring an n-bit number N into its prime factors is much harder. The currently best known algorithm needs $\exp\left(c(\log N)^{1/3}(\log\log N)^{2/3}\right)$ operations.

Query Problems

Query complexity and black boxes

- Instead of counting the number of operations we can count the number of queries to f in order to solve the problem.
- Often upper and lower bounds can be shown for the query complexity model only.
- Black-box model: we assume that we are given a function f but cannot analyze how f is actually implemented.
- Most real-world problems are actually white-box, for example FACTORING, GRAPH-ISO, etc. Lower bounds in the white-box problems are typically very weak.

Examples

- Black box problems: Simon's algorithm, Grover's algorithm.
- White-box problems: Factoring and dlog, phase estimation.

Query Problems

Query complexity and black boxes

- Instead of counting the number of operations we can count the number of queries to f in order to solve the problem.
- Often upper and lower bounds can be shown for the query complexity model only.
- Black-box model: we assume that we are given a function f but cannot analyze how f is actually implemented.
- Most real-world problems are actually white-box, for example FACTORING, GRAPH-ISO, etc. Lower bounds in the white-box problems are typically very weak.

Examples

- Black box problems: Simon's algorithm, Grover's algorithm.
- White-box problems: Factoring and dlog, phase estimation.

Simon's Problem

The XOR bit mask problem

We consider only functions $f : \{0,1\}^n \rightarrow \{0,1\}^n$ for which there exists $s \in \{0,1\}^n$ such that

- $\forall x \in \{0,1\}^n$ we have $f(x) = f(x \oplus s)$,
- $\forall x, y \in \{0,1\}^n$ we have that if $x \neq y \oplus s$, then $f(x) \neq f(y)$.

The task is to find the hidden string s.

Example where $f : \{0,1\}^3 \rightarrow \{0,1\}^3$

0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	1

Here $s = (0,1,1)$.

Simon's Problem

The XOR bit mask problem

We consider only functions $f : \{0,1\}^n \to \{0,1\}^n$ for which there exists $s \in \{0,1\}^n$ such that

- $\forall x \in \{0,1\}^n$ we have $f(x) = f(x \oplus s)$,
- $\forall x, y \in \{0,1\}^n$ we have that if $x \neq y \oplus s$, then $f(x) \neq f(y)$.

The task is to find the hidden string s.

Example where $f : \{0,1\}^3 \to \{0,1\}^3$

$$
\begin{array}{ccc|ccc}
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1
\end{array}
$$

Here $s = (0,1,1)$.

Simon's Problem

The XOR bit mask problem

We consider only functions $f : \{0,1\}^n \to \{0,1\}^n$ for which there exists $s \in \{0,1\}^n$ such that

- $\forall x \in \{0,1\}^n$ we have $f(x) = f(x \oplus s)$,
- $\forall x, y \in \{0,1\}^n$ we have that if $x \neq y \oplus s$, then $f(x) \neq f(y)$.

The task is to find the hidden string s.

Example where $f : \{0,1\}^3 \to \{0,1\}^3$

$$\begin{array}{ccc|ccc}
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1
\end{array}$$

Here $s = (0,1,1)$.

Some comments about Simon's problem

- There is some similarity to the Deutsch-Jozsa problem, however, here the task is not to distinguish between two cases (constant vs balanced) but between exponentially many cases (one for each unknown string s).
- Problem might seem artificial, but Shor's algorithm has the same underlying idea. Note: This is a promise problem!
- Problem gave the first strong (polynomial vs exponential) separation between quantum and classical computing. Before that only super-polynomial separations were known.

Literature

D. R. Simon.
On the Power of Quantum Computation.
Proc. FOCS 94, 116–123. IEEE, 1994.

Simon's Problem

Some comments about Simon's problem

- There is some similarity to the Deutsch-Jozsa problem, however, here the task is not to distinguish between two cases (constant vs balanced) but between exponentially many cases (one for each unknown string s).
- Problem might seem artificial, but Shor's algorithm has the same underlying idea. Note: This is a promise problem!
- Problem gave the first strong (polynomial vs exponential) separation between quantum and classical computing. Before that only super-polynomial separations were known.

Literature

D. R. Simon.
On the Power of Quantum Computation.
Proc. FOCS 94, 116–123. IEEE, 1994.

Simon's Problem

Some comments about Simon's problem

- There is some similarity to the Deutsch-Jozsa problem, however, here the task is not to distinguish between two cases (constant vs balanced) but between exponentially many cases (one for each unknown string s).
- Problem might seem artificial, but Shor's algorithm has the same underlying idea. Note: This is a promise problem!
- Problem gave the first strong (polynomial vs exponential) separation between quantum and classical computing. Before that only super-polynomial separations were known.

Literature

D. R. Simon.
On the Power of Quantum Computation.
Proc. FOCS 94, 116–123. IEEE, 1994.

Some comments about Simon's problem

- There is some similarity to the Deutsch-Jozsa problem, however, here the task is not to distinguish between two cases (constant vs balanced) but between exponentially many cases (one for each unknown string s).
- Problem might seem artificial, but Shor's algorithm has the same underlying idea. Note: This is a promise problem!
- Problem gave the first strong (polynomial vs exponential) separation between quantum and classical computing. Before that only super-polynomial separations were known.

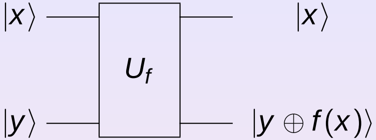
Literature

D. R. Simon.
On the Power of Quantum Computation.
Proc. FOCS 94, 116–123. IEEE, 1994.

Simon's Problem: Specification

How the function f is given

We are given f as a black box in form of a quantum circuit computing U_f:



How can we query f?

- Classical algorithm: makes queries $x_1, \ldots, x_k$ resulting in the answers $f(x_1), \ldots, f(x_k)$ (counts as k queries).
- Quantum algorithm: can query in superposition, i. e., starting with the state $|\varphi\rangle = \sum_{i=1}^{k} |x_i\rangle$ results in

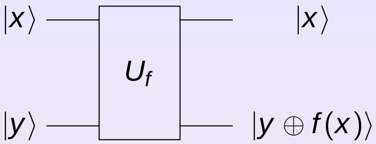
$$U_f\,|\varphi\rangle\,|0\rangle = \sum_{i=1}^{k} U_f\,|x_i\rangle\,|0\rangle = \sum_{i=1}^{k} |x_i\rangle\,|f(x_i)\rangle\,.$$

This counts as one query!

Simon's Problem: Specification

How the function f is given

We are given f as a black box in form of a quantum circuit computing U_f:



How can we query f?

- Classical algorithm: makes queries $x_1, \ldots, x_k$ resulting in the answers $f(x_1), \ldots, f(x_k)$ (counts as k queries).
- Quantum algorithm: can query in superposition, i. e., starting with the state $|\varphi\rangle = \sum_{i=1}^{k} |x_i\rangle$ results in

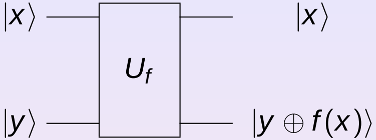
$$U_f |\varphi\rangle |0\rangle = \sum_{i=1}^{k} U_f |x_i\rangle |0\rangle = \sum_{i=1}^{k} |x_i\rangle |f(x_i)\rangle .$$

This counts as one query!

Simon's Problem: Specification

How the function f is given

We are given f as a black box in form of a quantum circuit computing U_f:



How can we query f?

- Classical algorithm: makes queries $x_1, \ldots, x_k$ resulting in the answers $f(x_1), \ldots, f(x_k)$ (counts as k queries).
- Quantum algorithm: can query in superposition, i. e., starting with the state $|\varphi\rangle = \sum_{i=1}^{k} |x_i\rangle$ results in

$$U_f |\varphi\rangle |0\rangle = \sum_{i=1}^{k} U_f |x_i\rangle |0\rangle = \sum_{i=1}^{k} |x_i\rangle |f(x_i)\rangle .$$

This counts as one query!

Partial measurements / “non-demolition” measurements

- Suppose we are given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and have prepared the state

$$|\psi\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle |f(x)\rangle = \frac{1}{2^{n/2}} \left(\sum_{x:f(x)=0} \alpha_x |x\rangle |0\rangle + \sum_{x:f(x)=1} \alpha_x |x\rangle |1\rangle \right).$$

- Measuring the last qubit gives a random variable X.

$$\Pr(X=0) = \sum_{x:f(x)=0} |\alpha_x|^2, \; \Pr(X=1) = \sum_{x:f(x)=1} |\alpha_x|^2.$$

- This talk: We only consider von Neumann measurements of some (or all) of the qubits.

Measuring the State of the System

Partial measurements / “non-demolition” measurements

- Suppose we are given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and have prepared the state

$$\begin{aligned} |\psi\rangle &= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle |f(x)\rangle \\ &= \frac{1}{2^{n/2}} \left(\sum_{x : f(x)=0} \alpha_x |x\rangle |0\rangle + \sum_{x : f(x)=1} \alpha_x |x\rangle |1\rangle \right). \end{aligned}$$

- Measuring the last qubit gives a random variable X.

$$\Pr(X = 0) = \sum_{x : f(x)=0} |\alpha_x|^2, \ \Pr(X = 1) = \sum_{x : f(x)=1} |\alpha_x|^2.$$

- This talk: We only consider von Neumann measurements of some (or all) of the qubits.

Measuring the State of the System

Partial measurements / "non-demolition" measurements

- Suppose we are given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and have prepared the state

$$\begin{aligned} |\psi\rangle &= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle |f(x)\rangle \\ &= \frac{1}{2^{n/2}} \left(\sum_{x:f(x)=0} \alpha_x |x\rangle |0\rangle + \sum_{x:f(x)=1} \alpha_x |x\rangle |1\rangle \right). \end{aligned}$$

- Measuring the last qubit gives a random variable X.

$$\Pr(X=0) = \sum_{x:f(x)=0} |\alpha_x|^2, \ \Pr(X=1) = \sum_{x:f(x)=1} |\alpha_x|^2.$$

- This talk: We only consider von Neumann measurements of some (or all) of the qubits.

Measuring the State of the System

Why partial/"non-demolition" measurements

- We can still work with the collapsed state! For instance if measuring the last qubit of

$$|\psi\rangle = \frac{1}{2^{n/2}} \left(\sum_{x:f(x)=0} \alpha_x |x\rangle |0\rangle + \sum_{x:f(x)=1} \alpha_x |x\rangle |1\rangle \right)$$

  yields the result $X = $ "$1''$, then the collapsed state is

$$|\psi_1\rangle = \frac{1}{\sqrt{s_1}} \sum_{x:f(x)=1} \alpha_x |x\rangle , \quad \text{where } s_1 = |\{x : f(x) = 1\}|.$$

- Note: cannot be used to find solutions to $f(x) = 1$. Why?
- Further reading: the most general operation which can be applied to a quantum system in order to obtain classical information is a POVM (positive operator valued measure).

Why partial/“non-demolition” measurements

- We can still work with the collapsed state! For instance if measuring the last qubit of

$$|\psi\rangle = \frac{1}{2^{n/2}}\left(\sum_{x:f(x)=0} \alpha_x |x\rangle |0\rangle + \sum_{x:f(x)=1} \alpha_x |x\rangle |1\rangle\right)$$

  yields the result $X = $ “1”, then the collapsed state is

$$|\psi_1\rangle = \frac{1}{\sqrt{s_1}} \sum_{x:f(x)=1} \alpha_x |x\rangle, \quad \text{where } s_1 = |\{x : f(x) = 1\}|.$$

- Note: cannot be used to find solutions to $f(x) = 1$. Why?
- Further reading: the most general operation which can be applied to a quantum system in order to obtain classical information is a POVM (positive operator valued measure).

Measuring the State of the System

Why partial/“non-demolition” measurements

- We can still work with the collapsed state! For instance if measuring the last qubit of

$$|\psi\rangle = \frac{1}{2^{n/2}} \left(\sum_{x:f(x)=0} \alpha_x |x\rangle |0\rangle + \sum_{x:f(x)=1} \alpha_x |x\rangle |1\rangle \right)$$

  yields the result $X = “1''$, then the collapsed state is

$$|\psi_1\rangle = \frac{1}{\sqrt{s_1}} \sum_{x:f(x)=1} \alpha_x |x\rangle , \quad \text{where } s_1 = |\{x : f(x) = 1\}|.$$

- Note: cannot be used to find solutions to $f(x) = 1$. Why?
- Further reading: the most general operation which can be applied to a quantum system in order to obtain classical information is a POVM (positive operator valued measure).

Simon's Problem: Preparing Useful States

Creating the uniform superposition

The basic idea is to prepare

$$|0\rangle\,|0\rangle \overset{H_2^{\otimes n}}{\longmapsto} \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbf{F}_2^n} |x\rangle\,|0\rangle$$

$$\overset{U_f}{\longmapsto} \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbf{F}_2^n} |x\rangle\,|f(x)\rangle\,.$$

Collapsing the uniform superposition

Now, measuring the second register will yield a random $s \in \mathbf{F}_2^n$ in the image of f. The state collapses to

$$|\varphi_{x_0,s}\rangle = \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle).$$

Simon's Problem: Preparing Useful States

Creating the uniform superposition

The basic idea is to prepare

$$
\begin{aligned}
|0\rangle\,|0\rangle \quad &\overset{H_2^{\otimes n}}{\longmapsto} \quad \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbf{F}_2^n} |x\rangle\,|0\rangle \\
&\overset{U_f}{\longmapsto} \quad \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbf{F}_2^n} |x\rangle\,|f(x)\rangle\,.
\end{aligned}
$$

Collapsing the uniform superposition

Now, measuring the second register will yield a random $s \in \mathbf{F}_2^n$ in the image of f. The state collapses to

$$
|\varphi_{x_0,s}\rangle = \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle).
$$

Computing the Hadamard Transform

We apply $H_2^{\otimes n}$ to the collapsed states $|\varphi_{x_0,s}\rangle$:

$$
\begin{aligned}
H_2^{\otimes n}\,|\varphi_{x_0,s}\rangle &= \left(\frac{1}{2^{n/2}}\sum_{x,y\in\mathbf{F}_2^n}(-1)^{x\cdot y}\,|x\rangle\,\langle y|\right)\left(\frac{1}{\sqrt{2}}(|x_0\rangle+|x_0\oplus s\rangle)\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}\,|x\rangle+\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot(x_0\oplus s)}\,|x\rangle\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}\left((-1)^{x\cdot x_0}\,|x\rangle+(-1)^{x\cdot x_0\oplus x\cdot s}\right)|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}((-1)^{x\cdot x_0}\,|x\rangle+(-1)^{x\cdot x_0}(-1)^{x\cdot s})\,|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}\left(1+(-1)^{x\cdot s}\right)|x\rangle
\end{aligned}
$$

Computing the Hadamard Transform

We apply $H_2^{\otimes n}$ to the collapsed states $|\varphi_{x_0,s}\rangle$:

$$
\begin{aligned}
H_2^{\otimes n}|\varphi_{x_0,s}\rangle &= \left(\frac{1}{2^{n/2}}\sum_{x,y\in\mathbf{F}_2^n}(-1)^{x\cdot y}|x\rangle\langle y|\right)\left(\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0\oplus s\rangle)\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}|x\rangle + \sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot(x_0\oplus s)}|x\rangle\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}\left((-1)^{x\cdot x_0}|x\rangle + (-1)^{x\cdot x_0\oplus x\cdot s)}\right)|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}((-1)^{x\cdot x_0}|x\rangle + (-1)^{x\cdot x_0}(-1)^{x\cdot s})|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}(1+(-1)^{x\cdot s})|x\rangle
\end{aligned}
$$

Computing the Hadamard Transform

We apply $H_2^{\otimes n}$ to the collapsed states $|\varphi_{x_0,s}\rangle$:

$$
\begin{aligned}
H_2^{\otimes n}|\varphi_{x_0,s}\rangle &= \left(\frac{1}{2^{n/2}}\sum_{x,y\in\mathbf{F}_2^n}(-1)^{x\cdot y}|x\rangle\langle y|\right)\left(\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0\oplus s\rangle)\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}|x\rangle + \sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot(x_0\oplus s)}|x\rangle\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}\left((-1)^{x\cdot x_0}|x\rangle + (-1)^{x\cdot x_0\oplus x\cdot s)}\right)|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}\left((-1)^{x\cdot x_0}|x\rangle + (-1)^{x\cdot x_0}(-1)^{x\cdot s}\right)|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}\left(1+(-1)^{x\cdot s}\right)|x\rangle
\end{aligned}
$$

Computing the Hadamard Transform

We apply $H_2^{\otimes n}$ to the collapsed states $|\varphi_{x_0,s}\rangle$:

$$
\begin{aligned}
H_2^{\otimes n}\,|\varphi_{x_0,s}\rangle &= \left(\frac{1}{2^{n/2}}\sum_{x,y\in\mathbf{F}_2^n}(-1)^{x\cdot y}\,|x\rangle\,\langle y|\right)\left(\frac{1}{\sqrt{2}}(|x_0\rangle+|x_0\oplus s\rangle)\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}\,|x\rangle+\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot(x_0\oplus s)}\,|x\rangle\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}\left((-1)^{x\cdot x_0}\,|x\rangle+(-1)^{x\cdot x_0\oplus x\cdot s}\right)|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}\left((-1)^{x\cdot x_0}\,|x\rangle+(-1)^{x\cdot x_0}(-1)^{x\cdot s}\right)|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}\left(1+(-1)^{x\cdot s}\right)|x\rangle
\end{aligned}
$$

Computing the Hadamard Transform

We apply $H_2^{\otimes n}$ to the collapsed states $|\varphi_{x_0,s}\rangle$:

$$
\begin{aligned}
H_2^{\otimes n}|\varphi_{x_0,s}\rangle &= \left(\frac{1}{2^{n/2}}\sum_{x,y\in\mathbf{F}_2^n}(-1)^{x\cdot y}|x\rangle\langle y|\right)\left(\frac{1}{\sqrt{2}}(|x_0\rangle+|x_0\oplus s\rangle)\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\left(\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}|x\rangle+\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot(x_0\oplus s)}|x\rangle\right)\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}\left((-1)^{x\cdot x_0}|x\rangle+(-1)^{x\cdot x_0\oplus x\cdot s)}\right)|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}((-1)^{x\cdot x_0}|x\rangle+(-1)^{x\cdot x_0}(-1)^{x\cdot s})|x\rangle\\
&= \frac{1}{\sqrt{2^{n+1}}}\sum_{x\in\mathbf{F}_2^n}(-1)^{x\cdot x_0}\left(1+(-1)^{x\cdot s}\right)|x\rangle
\end{aligned}
$$

What have we gained by doing this?

$$H_2^{\otimes n} |\varphi_{x_0,s}\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{F}_2^n} (-1)^{x \cdot x_0} \left(1 + (-1)^{x \cdot s}\right) |x\rangle$$

$$= \frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \cdot s = 0}} (-1)^{x \cdot x_0} |x\rangle$$

Hence measuring this state yields a random element in

$$\langle s \rangle^{\perp} = \{x \in \mathbb{F}_2^n | x \cdot s = 0\}.$$

What we really want...

... are elements from $\langle s \rangle$ itself (there is only 0 and s itself since $\langle s \rangle$ is one-dimensional). How can we compute $\langle s \rangle$ from $\langle s \rangle^{\perp}$?

Simon's Problem: Destructive Interference

What have we gained by doing this?

$$H_2^{\otimes n} |\varphi_{x_0,s}\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbf{F}_2^n} (-1)^{x\cdot x_0} \left(1 + (-1)^{x\cdot s}\right) |x\rangle$$

$$= \frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbf{F}_2^n \\ x\cdot s = 0}} (-1)^{x\cdot x_0} |x\rangle$$

Hence measuring this state yields a random element in

$$\langle s\rangle^{\perp} = \{x \in \mathbf{F}_2^n | x \cdot s = 0\}.$$

What we really want...

... are elements from $\langle s\rangle$ itself (there is only 0 and s itself since $\langle s\rangle$ is one-dimensional). How can we compute $\langle s\rangle$ from $\langle s\rangle^{\perp}$?

What have we gained by doing this?

$$
\begin{aligned}
H_2^{\otimes n} |\varphi_{x_0,s}\rangle &= \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{F}_2^n} (-1)^{x \cdot x_0} \left(1 + (-1)^{x \cdot s}\right) |x\rangle \\
&= \frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \cdot s = 0}} (-1)^{x \cdot x_0} |x\rangle
\end{aligned}
$$

Hence measuring this state yields a random element in

$$\langle s \rangle^{\perp} = \{x \in \mathbb{F}_2^n | x \cdot s = 0\}.$$

What we really want...

... are elements from $\langle s \rangle$ itself (there is only 0 and s itself since $\langle s \rangle$ is one-dimensional). How can we compute $\langle s \rangle$ from $\langle s \rangle^{\perp}$?

Simon's Problem: Destructive Interference

What have we gained by doing this?

$$H_2^{\otimes n} |\varphi_{x_0,s}\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{F}_2^n} (-1)^{x \cdot x_0} \left(1 + (-1)^{x \cdot s}\right) |x\rangle$$

$$= \frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \cdot s = 0}} (-1)^{x \cdot x_0} |x\rangle$$

Hence measuring this state yields a random element in

$$\langle s \rangle^{\perp} = \{x \in \mathbb{F}_2^n | x \cdot s = 0\}.$$

What we really want...

... are elements from $\langle s \rangle$ itself (there is only 0 and s itself since $\langle s \rangle$ is one-dimensional). How can we compute $\langle s \rangle$ from $\langle s \rangle^{\perp}$?

Solution to Simon's Problem

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbf{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x\in\mathbf{F}_2^n \\ x\cdot s=0}} (-1)^{x\cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbf{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Solution to Simon's Problem

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbf{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}}\sum_{\substack{x\in\mathbf{F}_2^n\\ x\cdot s=0}}(-1)^{x\cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbf{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbf{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $\displaystyle \frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbf{F}_2^n \\ x \cdot s = 0}} (-1)^{x \cdot x_0}\,|x\rangle$
6. Measure first register: Sample $y \in \mathbf{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Solution to Simon's Problem

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbf{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x\in\mathbf{F}_2^n \\ x\cdot s=0}} (-1)^{x\cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbf{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbb{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x\in\mathbb{F}_2^n \\ x\cdot s=0}} (-1)^{x\cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbb{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Solution to Simon's Problem

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbb{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}}\sum_{\substack{x\in\mathbb{F}_2^n\\ x\cdot s=0}}(-1)^{x\cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbb{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Solution to Simon's Problem

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbf{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x\in\mathbf{F}_2^n \\ x\cdot s=0}} (-1)^{x\cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbf{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Solution to Simon's Problem

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.
Task: Find the unknown bit-string $s \in \mathbb{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \cdot s = 0}} (-1)^{x \cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbb{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Solution to Simon's Problem

Quantum algorithm

Given: Function $f : \{0,1\}^n \to \{0,1\}^n$ with Simon promise, i. e., preimages of a fixed image have the form x_0 and $x_0 \oplus s$.

Task: Find the unknown bit-string $s \in \mathbb{F}_2^n$.

Repeat the following steps $n-1$ times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = |\varphi_{x_0,s}\rangle$
5. Compute $H_2^{\otimes n}$ on first register: $$\frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x\in\mathbb{F}_2^n \\ x\cdot s=0}} (-1)^{x\cdot x_0}\,|x\rangle$$
6. Measure first register: Sample $y \in \mathbb{F}_2^n$ with $y \cdot s = 0$.

Further classical post-processing is necessary!

Classical Postprocessing in Simon's Algorithm

Classical post-processing

- After $n-1$ iterations: $y_1, \ldots, y_{n-1} \in \mathbb{F}_2^n$ with $y_i \cdot s = 0$.
- We have to infer s by a purely classical computation.
- Show high probability of success over the choice of y_i.

Linear algebra over $\mathbb{F}_2$

We are given the linear system of equations

$$A \cdot s = \begin{pmatrix} y_1 \\ \hline \vdots \\ \hline y_{n-1} \end{pmatrix} \cdot s = 0$$

Hence, we have to compute the kernel of $A \in \mathbb{F}_2^{(n-1)\times n}$. If the kernel is one-dimensional, then s is uniquely determined.

Classical Postprocessing in Simon's Algorithm

Classical post-processing

- After $n-1$ iterations: $y_1, \ldots, y_{n-1} \in \mathbb{F}_2^n$ with $y_i \cdot s = 0$.
- We have to infer s by a purely classical computation.
- Show high probability of success over the choice of y_i.

Linear algebra over $\mathbb{F}_2$

We are given the linear system of equations

$$A \cdot s = \left(\begin{array}{c} y_1 \\ \hline \vdots \\ \hline y_{n-1} \end{array} \right) \cdot s = 0$$

Hence, we have to compute the kernel of $A \in \mathbb{F}_2^{(n-1)\times n}$. If the kernel is one-dimensional, then s is uniquely determined.

Classical Post-Processing in Simon's Algorithm

The probability of success

Since $\dim(\langle s\rangle) = 1$ we have that $\dim(\langle s\rangle^\perp) = n-1$. Hence we have to bound the probability that $n-1$ random vectors in F_2^n are linear independent:

$$\begin{aligned}
\Pr(rk(A) = n-1) &= \left(\frac{2^{n-1}-1}{2^{n-1}}\right)\left(\frac{2^{n-1}-2}{2^{n-1}}\right)\cdot\ldots\ldots\left(\frac{2^{n-1}-2^{n-2}}{2^{n-1}}\right) \\
&= \left(1-\frac{1}{2^{n-1}}\right)\left(1-\frac{1}{2^{n-2}}\right)\cdot\ldots\ldots\left(1-\frac{1}{4}\right)\cdot\frac{1}{2} \\
&\geq \left(1-\left(\frac{1}{2^{n-1}}+\ldots+\frac{1}{4}\right)\right)\frac{1}{2} \geq \left(1-\frac{1}{2}\right)\cdot\frac{1}{2} \geq \frac{1}{4}
\end{aligned}$$

Complexity of the quantum algorithm

- We have used n iterations and each individual run uses one query, $2n$ Hadamard transforms, and $n+1$ single qubit measurements
- Postprocessing: Computing the kernel of a matrix of size $n \times n$ is linear algebra and can be solved in time $O(n^3)$.
- Overall we have found a polynomial-time quantum algorithm.

Classical Post-Processing in Simon's Algorithm

The probability of success

Since $\dim(\langle s\rangle) = 1$ we have that $\dim(\langle s\rangle^{\perp}) = n-1$. Hence we have to bound the probability that $n-1$ random vectors in F_2^n are linear independent:

$$\begin{aligned}
\Pr(rk(A) = n-1) &= \left(\frac{2^{n-1}-1}{2^{n-1}}\right)\left(\frac{2^{n-1}-2}{2^{n-1}}\right)\cdot\ldots\ldots\left(\frac{2^{n-1}-2^{n-2}}{2^{n-1}}\right) \\
&= \left(1-\frac{1}{2^{n-1}}\right)\left(1-\frac{1}{2^{n-2}}\right)\cdot\ldots\ldots\left(1-\frac{1}{4}\right)\cdot\frac{1}{2} \\
&\geq \left(1-\left(\frac{1}{2^{n-1}}+\ldots+\frac{1}{4}\right)\right)\frac{1}{2} \geq \left(1-\frac{1}{2}\right)\cdot\frac{1}{2} \geq \frac{1}{4}
\end{aligned}$$

Complexity of the quantum algorithm

- We have used n iterations and each individual run uses one query, $2n$ Hadamard transforms, and $n+1$ single qubit measurements
- Postprocessing: Computing the kernel of a matrix of size $n \times n$ is linear algebra and can be solved in time $O(n^3)$.
- Overall we have found a polynomial-time quantum algorithm.

The probability of success

Since $\dim(\langle s\rangle) = 1$ we have that $\dim(\langle s\rangle^\perp) = n-1$. Hence we have to bound the probability that $n-1$ random vectors in $\mathbb{F}_2^n$ are linear independent:

$$\begin{aligned}
\Pr(rk(A) = n-1) &= \left(\frac{2^{n-1}-1}{2^{n-1}}\right)\left(\frac{2^{n-1}-2}{2^{n-1}}\right)\cdots\cdots\left(\frac{2^{n-1}-2^{n-2}}{2^{n-1}}\right) \\
&= \left(1-\frac{1}{2^{n-1}}\right)\left(1-\frac{1}{2^{n-2}}\right)\cdots\cdots\left(1-\frac{1}{4}\right)\cdot\frac{1}{2} \\
&\geq \left(1-\left(\frac{1}{2^{n-1}}+\ldots+\frac{1}{4}\right)\right)\frac{1}{2} \geq \left(1-\frac{1}{2}\right)\cdot\frac{1}{2} \geq \frac{1}{4}
\end{aligned}$$

Complexity of the quantum algorithm

- We have used n iterations and each individual run uses one query, $2n$ Hadamard transforms, and $n+1$ single qubit measurements
- Postprocessing: Computing the kernel of a matrix of size $n \times n$ is linear algebra and can be solved in time $O(n^3)$.
- Overall we have found a polynomial-time quantum algorithm.

Classical Post-Processing in Simon's Algorithm

The probability of success

Since $\dim(\langle s \rangle) = 1$ we have that $\dim(\langle s \rangle^\perp) = n-1$. Hence we have to bound the probability that $n-1$ random vectors in $\mathbb{F}_2^n$ are linear independent:

$$\begin{aligned}
\Pr(rk(A) = n-1) &= \left(\frac{2^{n-1}-1}{2^{n-1}}\right)\left(\frac{2^{n-1}-2}{2^{n-1}}\right)\cdots\cdots\left(\frac{2^{n-1}-2^{n-2}}{2^{n-1}}\right) \\
&= \left(1-\frac{1}{2^{n-1}}\right)\left(1-\frac{1}{2^{n-2}}\right)\cdots\cdots\left(1-\frac{1}{4}\right)\cdot\frac{1}{2} \\
&\geq \left(1-\left(\frac{1}{2^{n-1}}+\ldots+\frac{1}{4}\right)\right)\frac{1}{2} \geq \left(1-\frac{1}{2}\right)\cdot\frac{1}{2} \geq \frac{1}{4}
\end{aligned}$$

Complexity of the quantum algorithm

- We have used n iterations and each individual run uses one query, $2n$ Hadamard transforms, and $n+1$ single qubit measurements
- Postprocessing: Computing the kernel of a matrix of size $n \times n$ is linear algebra and can be solved in time $O(n^3)$.
- Overall we have found a polynomial-time quantum algorithm.

Simon's Problem: Classical Lower Bound

Theorem

Let $\mathcal{A}$ be a classical probabilistic algorithm which determines s using k queries to the function f. Then $k = \Omega(2^{n/2})$.

Computational complexity theory lingo

Hence there exists an oracle $\mathcal{O}$ with respect which we have a separation between classical and quantum computation:

$$\mathrm{BPP}^{\mathcal{O}} \neq \mathrm{BQP}^{\mathcal{O}}$$

- This is a so-called "relativized result", i. e., it holds in a world in which calls to the oracle cost only one query.
- Whether this also holds for our world, i. e., without oracles, is a major open problem in theoretical computer science.
- Looking ahead: Why does the fact that *FACTORING* is in BQP, whereas no classical algorithm is known for it, does not imply that BPP $\neq$ BQP?

Simon's Problem: Classical Lower Bound

Theorem

Let $\mathcal{A}$ be a classical probabilistic algorithm which determines s using k queries to the function f. Then $k = \Omega(2^{n/2})$.

Computational complexity theory lingo

Hence there exists an oracle $\mathcal{O}$ with respect which we have a separation between classical and quantum computation:

$$\mathrm{BPP}^{\mathcal{O}} \neq \mathrm{BQP}^{\mathcal{O}}$$

- This is a so-called "relativized result", i. e., it holds in a world in which calls to the oracle cost only one query.
- Whether this also holds for our world, i. e., without oracles, is a major open problem in theoretical computer science.
- Looking ahead: Why does the fact that *FACTORING* is in BQP, whereas no classical algorithm is known for it, does not imply that BPP $\neq$ BQP?

Simon's Problem: Classical Lower Bound

Theorem

Let $\mathcal{A}$ be a classical probabilistic algorithm which determines s using k queries to the function f. Then $k = \Omega(2^{n/2})$.

Computational complexity theory lingo

Hence there exists an oracle $\mathcal{O}$ with respect which we have a separation between classical and quantum computation:

$$\mathrm{BPP}^{\mathcal{O}} \neq \mathrm{BQP}^{\mathcal{O}}$$

- This is a so-called "relativized result", i. e., it holds in a world in which calls to the oracle cost only one query.
- Whether this also holds for our world, i. e., without oracles, is a major open problem in theoretical computer science.
- Looking ahead: Why does the fact that *FACTORING* is in BQP, whereas no classical algorithm is known for it, does not imply that BPP $\neq$ BQP?

Simon's Problem: Classical Lower Bound

Theorem

Let $\mathcal{A}$ be a classical probabilistic algorithm which determines s using k queries to the function f. Then $k = \Omega(2^{n/2})$.

Computational complexity theory lingo

Hence there exists an oracle $\mathcal{O}$ with respect which we have a separation between classical and quantum computation:

$$\mathrm{BPP}^{\mathcal{O}} \neq \mathrm{BQP}^{\mathcal{O}}$$

- This is a so-called "relativized result", i. e., it holds in a world in which calls to the oracle cost only one query.
- Whether this also holds for our world, i. e., without oracles, is a major open problem in theoretical computer science.
- Looking ahead: Why does the fact that *FACTORING* is in BQP, whereas no classical algorithm is known for it, does not imply that BPP $\neq$ BQP?

Simon's Problem: Classical Lower Bound

Theorem

Let $\mathcal{A}$ be a classical probabilistic algorithm which determines s using k queries to the function f. Then $k = \Omega(2^{n/2})$.

Computational complexity theory lingo

Hence there exists an oracle $\mathcal{O}$ with respect which we have a separation between classical and quantum computation:

$$\mathrm{BPP}^{\mathcal{O}} \neq \mathrm{BQP}^{\mathcal{O}}$$

- This is a so-called "relativized result", i. e., it holds in a world in which calls to the oracle cost only one query.
- Whether this also holds for our world, i. e., without oracles, is a major open problem in theoretical computer science.
- Looking ahead: Why does the fact that *FACTORING* is in BQP, whereas no classical algorithm is known for it, does not imply that BPP $\neq$ BQP?

Simon's Problem: Classical Lower Bound

Sketch of proof of the $\Omega(2^{n/2})$ lower bound

- Suppose $\mathcal{A}$ makes k queries $x_1, \ldots, x_k$, where $x_i \neq x_j$. Let $\mathcal{F}_k := \{f(x_i) : i = 1, \ldots, k\}$ and $\mathcal{E}_k := \{x_i \oplus x_j : i \neq j\}$.
- If $|\mathcal{F}_k| < k$ then we have found a collision, i. e. a pair (i_0, j_0) with $f(x_{i_0}) = f(x_{j_0})$. Then $s = x_{i_0} \oplus x_{j_0}$.
- Suppose there was no collision. Then $s \neq \mathcal{E}_k$ and $|\mathcal{E}_k| = \binom{k}{2}$ candidates have been eliminated.
- However, there are $2^n - 1 - \binom{k}{2}$ candidates for s. We show that they are equally likely for a given $\mathcal{F}_k$. Then $k = \Omega(2^n)$.
- Bayes rule:

$$\Pr(s = s_0 \,|\, |\mathcal{F}|_k = k) = \frac{\Pr(|\mathcal{F}_k| = k \,|\, s = s_0) \cdot \Pr(s = s_0)}{\Pr(|\mathcal{F}_k| = k)}.$$

The a priori probabilities $\Pr(s = s_0)$ are equal and by symmetry $\Pr(|\mathcal{F}_k| = k \,|\, s = s_0)$ is independent of s_0. $\square$

Simon's Problem: Classical Lower Bound

Sketch of proof of the $\Omega(2^{n/2})$ lower bound

- Suppose $\mathcal{A}$ makes k queries $x_1, \ldots, x_k$, where $x_i \neq x_j$. Let $\mathcal{F}_k := \{f(x_i) : i = 1, \ldots, k\}$ and $\mathcal{E}_k := \{x_i \oplus x_j : i \neq j\}$.
- If $|\mathcal{F}_k| < k$ then we have found a collision, i. e. a pair (i_0, j_0) with $f(x_{i_0}) = f(x_{j_0})$. Then $s = x_{i_0} \oplus x_{j_0}$.
- Suppose there was no collision. Then $s \neq \mathcal{E}_k$ and $|\mathcal{E}_k| = \binom{k}{2}$ candidates have been eliminated.
- However, there are $2^n - 1 - \binom{k}{2}$ candidates for s. We show that they are equally likely for a given $\mathcal{F}_k$. Then $k = \Omega(2^n)$.
- Bayes rule:

$$\Pr(s = s_0 \big| |\mathcal{F}|_k = k) = \frac{\Pr(|\mathcal{F}_k| = k \big| s = s_0) \cdot \Pr(s = s_0)}{\Pr(|\mathcal{F}_k| = k)}.$$

The a priori probabilities $\Pr(s = s_0)$ are equal and by symmetry $\Pr(|\mathcal{F}_k| = k \big| s = s_0)$ is independent of s_0.

Simon's Problem: Classical Lower Bound

Sketch of proof of the $\Omega(2^{n/2})$ lower bound

- Suppose $\mathcal{A}$ makes k queries $x_1, \ldots, x_k$, where $x_i \neq x_j$. Let $\mathcal{F}_k := \{f(x_i) : i = 1, \ldots, k\}$ and $\mathcal{E}_k := \{x_i \oplus x_j : i \neq j\}$.
- If $|\mathcal{F}_k| < k$ then we have found a collision, i. e. a pair (i_0, j_0) with $f(x_{i_0}) = f(x_{j_0})$. Then $s = x_{i_0} \oplus x_{j_0}$.
- Suppose there was no collision. Then $s \neq \mathcal{E}_k$ and $|\mathcal{E}_k| = \binom{k}{2}$ candidates have been eliminated.
- However, there are $2^n - 1 - \binom{k}{2}$ candidates for s. We show that they are equally likely for a given $\mathcal{F}_k$. Then $k = \Omega(2^n)$.
- Bayes rule:

$$\Pr(s = s_0 \,|\, |\mathcal{F}|_k = k) = \frac{\Pr(|\mathcal{F}_k| = k \,|\, s = s_0) \cdot \Pr(s = s_0)}{\Pr(|\mathcal{F}_k| = k)}.$$

The a priori probabilities $\Pr(s = s_0)$ are equal and by symmetry $\Pr(|\mathcal{F}_k| = k \,|\, s = s_0)$ is independent of s_0. $\square$

Simon's Problem: Classical Lower Bound

Sketch of proof of the $\Omega(2^{n/2})$ lower bound

- Suppose $\mathcal{A}$ makes k queries $x_1, \ldots, x_k$, where $x_i \neq x_j$. Let $\mathcal{F}_k := \{f(x_i) : i = 1, \ldots, k\}$ and $\mathcal{E}_k := \{x_i \oplus x_j : i \neq j\}$.
- If $|\mathcal{F}_k| < k$ then we have found a collision, i. e. a pair (i_0, j_0) with $f(x_{i_0}) = f(x_{j_0})$. Then $s = x_{i_0} \oplus x_{j_0}$.
- Suppose there was no collision. Then $s \neq \mathcal{E}_k$ and $|\mathcal{E}_k| = \binom{k}{2}$ candidates have been eliminated.
- However, there are $2^n - 1 - \binom{k}{2}$ candidates for s. We show that they are equally likely for a given $\mathcal{F}_k$. Then $k = \Omega(2^n)$.
- Bayes rule:

$$\Pr(s = s_0 \big| |\mathcal{F}|_k = k) = \frac{\Pr(|\mathcal{F}_k| = k \big| s = s_0) \cdot \Pr(s = s_0)}{\Pr(|\mathcal{F}_k| = k)}.$$

The a priori probabilities $\Pr(s = s_0)$ are equal and by symmetry $\Pr(|\mathcal{F}_k| = k \big| s = s_0)$ is independent of s_0. □

Simon's Problem: Classical Lower Bound

Sketch of proof of the $\Omega(2^{n/2})$ lower bound

- Suppose $\mathcal{A}$ makes k queries $x_1, \ldots, x_k$, where $x_i \neq x_j$. Let $\mathcal{F}_k := \{f(x_i) : i = 1, \ldots, k\}$ and $\mathcal{E}_k := \{x_i \oplus x_j : i \neq j\}$.
- If $|\mathcal{F}_k| < k$ then we have found a collision, i. e. a pair (i_0, j_0) with $f(x_{i_0}) = f(x_{j_0})$. Then $s = x_{i_0} \oplus x_{j_0}$.
- Suppose there was no collision. Then $s \neq \mathcal{E}_k$ and $|\mathcal{E}_k| = \binom{k}{2}$ candidates have been eliminated.
- However, there are $2^n - 1 - \binom{k}{2}$ candidates for s. We show that they are equally likely for a given $\mathcal{F}_k$. Then $k = \Omega(2^n)$.
- Bayes rule:

$$\Pr(s = s_0 \big| |\mathcal{F}|_k = k) = \frac{\Pr(|\mathcal{F}_k| = k \big| s = s_0) \cdot \Pr(s = s_0)}{\Pr(|\mathcal{F}_k| = k)}.$$

The a priori probabilities $\Pr(s = s_0)$ are equal and by symmetry $\Pr(|\mathcal{F}_k| = k \big| s = s_0)$ is independent of s_0. □

For Further Reading

- G. Alber, Th. Beth, M. Horodecki, P. Horodecki, R. Horodecki, M. Roetteler, H. Weinfurter, and A. Zeilinger.
  Quantum Information: An Introduction to Basic Theoretical Concepts and Experiments.
  Springer, 2001.
- J. Gruska
  Quantum Computing.
  McGraw-Hill, 1999.
- A. Yu. Kitaev, A. H. Shen, and M. N. Vyalyi.
  Classical and Quantum Computation.
  Graduate Studies in Mathematics, vol. 47, AMS, 2002.
- M. Nielsen und I. Chuang.
  Quantum Computation and Information.
  Cambridge University Press, 2000.

Conclusions

- Elementary quantum gates:
  - Controlled NOT gate
  - Local unitary transformations
- A simple quantum algorithm on two qubits which distinguishes constant from balanced functions.
- Separation: 1 query (quantum) vs 2 queries (classical)
- Quantum algorithm for Simon's problem based on:
  - Computing with superpositions
  - Interference of computational paths

Introduction to Quantum Algorithms
Part II: The Algorithms of Shor and Grover

Martin Rötteler

NEC Laboratories America, Inc.
4 Independence Way, Suite 200
Princeton, NJ 08540, U.S.A.

International Summer School on Quantum Information,
Max-Planck-Institut für Physik komplexer Systeme
Dresden, September 1, 2005

Overview

Today:

- Shor's algorithm
  - Modular exponentiation
  - Period extraction via Quantum Fourier Transform
  - Classical post-processing
- Generalizations of Shor's algorithm
- Grover's algorithm for searching a list
- Universal quantum gates

Outlook:

- On September 19, Markus Grassl will continue with an introduction to quantum error-correcting codes.

The Integer Factorization Problem

Basic problem

Given a natural number N. Find a (prime) factor of N.

Best known classical algorithm

The number field sieve has a complexity of

$$\exp\left((1.923 + o(1))(\log N)^{1/3}(\log\log N)^{2/3}\right)$$

which is (sub)exponential in the number $n = \log N$ of bits of N.

Making money with factoring

The company RSA offers \$ 200.000 for anybody who can factor a certain 617 digit number N. This number is known to be of the form $N = pq$ but finding p and q is infeasible using the best known classical algorithms.

The Integer Factorization Problem

Basic problem

Given a natural number N. Find a (prime) factor of N.

Best known classical algorithm

The number field sieve has a complexity of

$$\exp\left((1.923 + o(1))(\log N)^{1/3}(\log\log N)^{2/3}\right)$$

which is (sub)exponential in the number $n = \log N$ of bits of N.

Making money with factoring

The company RSA offers \$ 200.000 for anybody who can factor a certain 617 digit number N. This number is known to be of the form $N = pq$ but finding p and q is infeasible using the best known classical algorithms.

The Integer Factorization Problem

Basic problem

Given a natural number N. Find a (prime) factor of N.

Best known classical algorithm

The number field sieve has a complexity of

$$\exp\left((1.923 + o(1))(\log N)^{1/3}(\log\log N)^{2/3}\right)$$

which is (sub)exponential in the number $n = \log N$ of bits of N.

Making money with factoring

The company RSA offers \$ 200.000 for anybody who can factor a certain 617 digit number N. This number is known to be of the form $N = pq$ but finding p and q is infeasible using the best known classical algorithms.

The RSA Factoring Challenge

RSA-200 is factored!

In May 2005 a small team of people has factored RSA-200. At 663 bits, this is the largest RSA Challenge Number factored.

The classical effort undertaken

Sieving equivalent of 55 years on a single 2.2 GHz Opteron CPU. The matrix step took about 3 months on a cluster of 80 2.2 GHz Opterons. Computed from late 2003 to May 2005.

RSA-200 and its factors

$$N = \begin{array}{l} 27997833911221327870829467638722601621070446786955 \\ 42853756000992932612840010760934567105295536085606 \\ 18223519109513657886371059544820065767750985805576 \\ 13579098734950144178863178946295187237869221823983 \end{array}$$

$$p = \begin{array}{l} 35324619344027701212726049781984643686711974001976 \\ 25023649303468776121253679423200058547956528088349 \end{array}$$

$$q = \begin{array}{l} 79258699544783330333470858414800596877379758573642 \\ 19960734330341455767872818152135381409304740185467 \end{array}$$

The RSA Factoring Challenge

RSA-200 is factored!

In May 2005 a small team of people has factored RSA-200. At 663 bits, this is the largest RSA Challenge Number factored.

The classical effort undertaken

Sieving equivalent of 55 years on a single 2.2 GHz Opteron CPU. The matrix step took about 3 months on a cluster of 80 2.2 GHz Opterons. Computed from late 2003 to May 2005.

RSA-200 and its factors

$$N = \begin{array}{l} 27997833911221327870829467638722601621070446786955 \\ 42853756000992932612840010760934567105295536085606 \\ 18223519109513657886371059544820065767750985805576 \\ 13579098734950144178863178946295187237869221823983 \end{array}$$

$$p = \begin{array}{l} 35324619344027701212726049781984643686711974001976 \\ 25023649303468776121253679423200058547956528088349 \end{array}$$

$$q = \begin{array}{l} 79258699544783330333470858414800596877379758573642 \\ 19960734330341455767872818152135381409304740185467 \end{array}$$

The RSA Factoring Challenge

RSA-200 is factored!

In May 2005 a small team of people has factored RSA-200. At 663 bits, this is the largest RSA Challenge Number factored.

The classical effort undertaken

Sieving equivalent of 55 years on a single 2.2 GHz Opteron CPU. The matrix step took about 3 months on a cluster of 80 2.2 GHz Opterons. Computed from late 2003 to May 2005.

RSA-200 and its factors

$$N = \begin{array}{l}27997833911221327870829467638722601621070446786955\\42853756000992932612840010760934567105295536085606\\18223519109513657886371059544820065767750985805576\\13579098734950144178863178946295187237869221823983\end{array}$$

$$p = \begin{array}{l}35324619344027701212726049781984643686711974001976\\25023649303468776121253679423200058547956528088349\end{array}$$

$$q = \begin{array}{l}79258699544783330333470858414800596877379758573642\\19960734330341455767872818152135381409304740185467\end{array}$$

Shor's Algorithm: Reduction to Order Finding

Reformulating the factoring problem

We can factor N if the following problem can be solved:

- **Input:** A number a with $1 < a < N$.
- **Output:** The order r of a modulo N, i. e., the smallest integer $r > 0$ such that $a^r \equiv 1 \mod N$.

Why is this a reduction?

Suppose we want to find a divisor of N different from $+1$ or -1.

- Pick a random a with $1 < a < N$ and find its order r
- Suppose that r is even (happens with high probability):

$$0 = (a^r - 1) = (a^{r/2} - 1)(a^{r/2} + 1) \mod N.$$

- If $a^{r/2} \neq \pm 1$ then $\gcd(a^{r/2} - 1, N)$ and $\gcd(a^{r/2} + 1, N)$ yield at least one nontrivial divisor of N.

Shor's Algorithm: Reduction to Order Finding

Reformulating the factoring problem

We can factor N if the following problem can be solved:

- **Input:** A number a with $1 < a < N$.
- **Output:** The order r of a modulo N, i. e., the smallest integer $r > 0$ such that $a^r \equiv 1 \mod N$.

Why is this a reduction?

Suppose we want to find a divisor of N different from $+1$ or -1.

- Pick a random a with $1 < a < N$ and find its order r
- Suppose that r is even (happens with high probability):

$$0 = (a^r - 1) = (a^{r/2} - 1)(a^{r/2} + 1) \mod N.$$

- If $a^{r/2} \neq \pm 1$ then $\gcd(a^{r/2} - 1, N)$ and $\gcd(a^{r/2} + 1, N)$ yield at least one nontrivial divisor of N.

Shor's Algorithm: Reduction to Order Finding

Remarks about this reduction

- Note that $\gcd(a, b)$ of two n-bit integers can be computed in $poly(\log(n))$ time.
- There were two events in which the reduction fails: (i) we pick a with an odd order r and (ii) $a^{r/2} = \pm 1$. We have to bound the probability for one of these events to occur.

Theorem

Let $N = p_1^{\mu_1} \dots p_m^{\mu_m}$ with $m \geq 2$ and $p_i > 2$. Then

$$\Pr(\text{neither (i) nor (ii) occurs}) \geq 1 - \frac{1}{2^m}$$

The big question

How can we efficiently determine the multiplicative order of a random element a modulo N?

Shor's Algorithm: Reduction to Order Finding

Remarks about this reduction

- Note that $\gcd(a, b)$ of two n-bit integers can be computed in $poly(\log(n))$ time.
- There were two events in which the reduction fails: (i) we pick a with an odd order r and (ii) $a^{r/2} = \pm 1$. We have to bound the probability for one of these events to occur.

Theorem

Let $N = p_1^{\mu_1} \dots p_m^{\mu_m}$ with $m \geq 2$ and $p_i > 2$. Then

$$\Pr(\text{neither (i) nor (ii) occurs}) \geq 1 - \frac{1}{2^m}$$

The big question

How can we efficiently determine the multiplicative order of a random element a modulo N?

Shor's Algorithm: Reduction to Order Finding

Remarks about this reduction

- Note that $\gcd(a, b)$ of two n-bit integers can be computed in $poly(\log(n))$ time.
- There were two events in which the reduction fails: (i) we pick a with an odd order r and (ii) $a^{r/2} = \pm 1$. We have to bound the probability for one of these events to occur.

Theorem

Let $N = p_1^{\mu_1} \dots p_m^{\mu_m}$ with $m \geq 2$ and $p_i > 2$. Then

$$\Pr(\text{neither (i) nor (ii) occurs}) \geq 1 - \frac{1}{2^m}$$

The big question

How can we efficiently determine the multiplicative order of a random element a modulo N?

Defining a Period Function

The modular exponentiation map

Let N be an integer and let $a \in Z_N$.

- Let M be an integer. The modular exponentiation is the map $f : x \mapsto (a^x \bmod N)$ from Z_M to Z_N.
- Result: The map f can be implemented efficiently using standard arithmetic in $O(poly(\log N))$ operations.
- Hence also the map $U_f : |x\rangle |y\rangle \mapsto |x\rangle |a^x \bmod N\rangle$ can be implemented efficiently.
- Recall that the order of a is defined as the smallest integer r such that $a^r = 1 \bmod N$.

Observation

The function $f : x \mapsto (a^x \bmod N)$ is periodic and has period length r, i. e., $f(x) = f(x + r)$ for all inputs x.

Defining a Period Function

The modular exponentiation map

Let N be an integer and let $a \in Z_N$.

- Let M be an integer. The modular exponentiation is the map $f : x \mapsto (a^x \mod N)$ from Z_M to Z_N.
- Result: The map f can be implemented efficiently using standard arithmetic in $O(poly(\log N))$ operations.
- Hence also the map $U_f : |x\rangle |y\rangle \mapsto |x\rangle |a^x \mod N\rangle$ can be implemented efficiently.
- Recall that the order of a is defined as the smallest integer r such that $a^r = 1 \mod N$.

Observation

The function $f : x \mapsto (a^x \mod N)$ is periodic and has period length r, i. e., $f(x) = f(x + r)$ for all inputs x.

Defining a Period Function

The modular exponentiation map

Let N be an integer and let $a \in Z_N$.

- Let M be an integer. The modular exponentiation is the map $f : x \mapsto (a^x \bmod N)$ from Z_M to Z_N.
- Result: The map f can be implemented efficiently using standard arithmetic in $O(poly(\log N))$ operations.
- Hence also the map $U_f : |x\rangle |y\rangle \mapsto |x\rangle |a^x \bmod N\rangle$ can be implemented efficiently.
- Recall that the order of a is defined as the smallest integer r such that $a^r = 1 \bmod N$.

Observation

The function $f : x \mapsto (a^x \bmod N)$ is periodic and has period length r, i. e., $f(x) = f(x + r)$ for all inputs x.

Defining a Period Function

The modular exponentiation map

Let N be an integer and let $a \in Z_N$.

- Let M be an integer. The modular exponentiation is the map $f : x \mapsto (a^x \bmod N)$ from Z_M to Z_N.
- Result: The map f can be implemented efficiently using standard arithmetic in $O(poly(\log N))$ operations.
- Hence also the map $U_f : |x\rangle |y\rangle \mapsto |x\rangle |a^x \bmod N\rangle$ can be implemented efficiently.
- Recall that the order of a is defined as the smallest integer r such that $a^r = 1 \bmod N$.

Observation

The function $f : x \mapsto (a^x \bmod N)$ is periodic and has period length r, i. e., $f(x) = f(x + r)$ for all inputs x.

Defining a Period Function

The modular exponentiation map

Let N be an integer and let $a \in Z_N$.

- Let M be an integer. The modular exponentiation is the map $f : x \mapsto (a^x \bmod N)$ from Z_M to Z_N.
- Result: The map f can be implemented efficiently using standard arithmetic in $O(poly(\log N))$ operations.
- Hence also the map $U_f : |x\rangle\,|y\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$ can be implemented efficiently.
- Recall that the order of a is defined as the smallest integer r such that $a^r = 1 \bmod N$.

Observation

The function $f : x \mapsto (a^x \bmod N)$ is periodic and has period length r, i. e., $f(x) = f(x + r)$ for all inputs x.

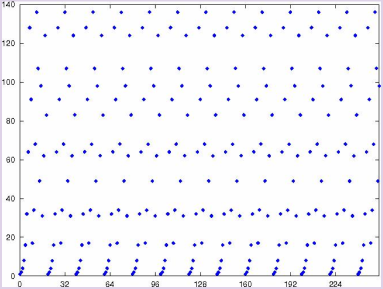
Setting up a Periodic State

Observation

The function $f : x \mapsto a^x \bmod N$ is periodic and has period length r, i. e., $f(x) = f(x+r)$ for all inputs x.

The graph of the function $f(x) = 2^x \bmod 165$

$|y = f(x)\rangle$



$|x\rangle$

Shor's Problem: Preparing Useful States

Creating the graph of f

Let $f(x) = a^x \bmod N$ be the modular exponentiation and let $U_f : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$ be as usual. We compute (letting $M = 2^m$)

$$|0\rangle |0\rangle \overset{H_2^{\otimes m}}{\mapsto} \frac{1}{\sqrt{M}} \sum_{x \in Z_M} |x\rangle |0\rangle \overset{U_f}{\mapsto} \frac{1}{\sqrt{M}} \sum_{x \in Z_M} |x\rangle |f(x)\rangle .$$

Collapsing the graph of f

Now, measuring the second register will yield a random $s \in Z_N$ in the image of f. The state collapses to

$$\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle .$$

Creating the graph of f

Let $f(x) = a^x \bmod N$ be the modular exponentiation and let $U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$ be as usual. We compute (letting $M = 2^m$)

$$|0\rangle|0\rangle \overset{H_2^{\otimes m}}{\mapsto} \frac{1}{\sqrt{M}} \sum_{x \in Z_M} |x\rangle|0\rangle \overset{U_f}{\mapsto} \frac{1}{\sqrt{M}} \sum_{x \in Z_M} |x\rangle|f(x)\rangle .$$

Collapsing the graph of f

Now, measuring the second register will yield a random $s \in Z_N$ in the image of f. The state collapses to

$$\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle .$$

An Application of the DFT: Period Extraction

Motivation

- We would like to apply the trick from Simon's algorithm:

$$|\varphi_{x_0,s}\rangle = \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) \mapsto \frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \cdot s = 0}} (-1)^{x \cdot x_0} |x\rangle .$$

- The unknown offset x_0 is transfered into the phases.

- The analogue of $|\varphi_{x_0,s}\rangle$ in case of the cyclic group Z_N is

$$|\psi_{x_0,r}\rangle = \frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle$$

- Again, we would like to transfer x_0 into the phases.

An Application of the DFT: Period Extraction

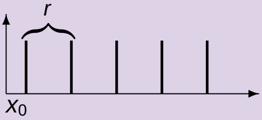
Motivation

- We would like to apply the trick from Simon's algorithm:

$$|\varphi_{x_0,s}\rangle = \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) \mapsto \frac{1}{\sqrt{2^{n+1}}} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \cdot s = 0}} (-1)^{x \cdot x_0} |x\rangle .$$

- The unknown offset x_0 is transfered into the phases.
- The analogue of $|\varphi_{x_0,s}\rangle$ in case of the cyclic group Z_N is

$$|\psi_{x_0,r}\rangle = \frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle$$



- Again, we would like to transfer x_0 into the phases.

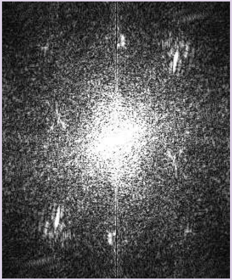
The Discrete Fourier Transformation (DFT)

Definition of the DFT

$$\mathrm{DFT}_N := \frac{1}{\sqrt{N}} \left[\omega_N^{k\cdot\ell} \right]_{k,\ell=0\ldots N-1}, \quad \omega_N = e^{2\pi i/N}$$

Example

$\xrightarrow{\text{DFT}}$



Estimating Character Sums

Useful trick in quantum computing (Character Lemma)

Lemma: For all $i = 0, \ldots, n-1$ the following holds:

$$\sum_{j=0}^{n-1} \omega_n^{ij} = n \cdot \delta_{i,0}$$

Proof: Let $S := \sum_{j=0}^{n-1} \omega_n^{ij}$. Then

$$\omega_n^i S = \sum_{j=0}^{n-1} \omega_n^i \omega_n^{ij} = \sum_{j=0}^{n-1} \omega_n^{i(j+1)} = \sum_{j=0}^{n-1} \omega_n^{ij} = S$$

- If $i \neq 0$ then $\omega_n^i \neq 1$, i. e., $(1 - \omega_n^i) \neq 0$. Hence $S = 0$.
- If $i = 0$ then $\omega_n^i = 1$ which implies that $S = n$.

Useful trick in quantum computing (Character Lemma)

Lemma: For all $i = 0, \ldots, n-1$ the following holds:

$$\sum_{j=0}^{n-1} \omega_n^{ij} = n \cdot \delta_{i,0}$$

Proof: Let $S := \sum_{j=0}^{n-1} \omega_n^{ij}$. Then

$$\omega_n^i S = \sum_{j=0}^{n-1} \omega_n^i \omega_n^{ij} = \sum_{j=0}^{n-1} \omega_n^{i(j+1)} = \sum_{j=0}^{n-1} \omega_n^{ij} = S$$

- If $i \neq 0$ then $\omega_n^i \neq 1$, i. e., $(1 - \omega_n^i) \neq 0$. Hence $S = 0$.
- If $i = 0$ then $\omega_n^i = 1$ which implies that $S = n$.

Estimating Character Sums

Useful trick in quantum computing (Character Lemma)

Lemma: For all $i = 0, \ldots, n-1$ the following holds:

$$\sum_{j=0}^{n-1} \omega_n^{ij} = n \cdot \delta_{i,0}$$

Proof: Let $S := \sum_{j=0}^{n-1} \omega_n^{ij}$. Then

$$\omega_n^i S = \sum_{j=0}^{n-1} \omega_n^i \omega_n^{ij} = \sum_{j=0}^{n-1} \omega_n^{i(j+1)} = \sum_{j=0}^{n-1} \omega_n^{ij} = S$$

- If $i \neq 0$ then $\omega_n^i \neq 1$, i. e., $(1 - \omega_n^i) \neq 0$. Hence $S = 0$.
- If $i = 0$ then $\omega_n^i = 1$ which implies that $S = n$.

Useful trick in quantum computing (Character Lemma)

Lemma: For all $i = 0, \ldots, n-1$ the following holds:

$$\sum_{j=0}^{n-1} \omega_n^{ij} = n \cdot \delta_{i,0}$$

Proof: Let $S := \sum_{j=0}^{n-1} \omega_n^{ij}$. Then

$$\omega_n^i S = \sum_{j=0}^{n-1} \omega_n^i \omega_n^{ij} = \sum_{j=0}^{n-1} \omega_n^{i(j+1)} = \sum_{j=0}^{n-1} \omega_n^{ij} = S$$

- If $i \neq 0$ then $\omega_n^i \neq 1$, i. e., $(1 - \omega_n^i) \neq 0$. Hence $S = 0$.
- If $i = 0$ then $\omega_n^i = 1$ which implies that $S = n$.

Extracting the Period of a Function

Theorem (Fourier duality)

Let $N \in \mathbb{N}$ and let $r \in Z_N$ be a divisor of N, and let $x_0 \in Z_N$. Then

$$\mathrm{DFT}_N \, |\psi_{x_0,r}\rangle = \mathrm{DFT}_N \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle \right) = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \omega_N^{\ell x_0 \frac{N}{r}} \left| \ell \frac{N}{r} \right\rangle$$

Proof:

$$\begin{aligned}
\mathrm{DFT}_N \, |\psi_{x_0,r}\rangle &= \left(\frac{1}{\sqrt{N}} \sum_{i,j=0}^{N-1} \omega_N^{ij} |i\rangle \langle j| \right) \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle \right) \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \left(\sum_{k=0}^{N/r-1} \omega_N^{i(x_0 + k \cdot r)} \right) |i\rangle \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{ix_0} \underbrace{\left(\sum_{k=0}^{N/r-1} \omega_N^{ikr} \right)}_{=:\alpha_i} |i\rangle
\end{aligned}$$

Extracting the Period of a Function

Theorem (Fourier duality)

Let $N \in \mathrm{N}$ and let $r \in Z_N$ be a divisor of N, and let $x_0 \in Z_N$. Then

$$\mathrm{DFT}_N \, |\psi_{x_0,r}\rangle = \mathrm{DFT}_N \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle \right) = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \omega_N^{\ell x_0 \frac{N}{r}} \left| \ell \frac{N}{r} \right\rangle$$

Proof:

$$\begin{aligned}
\mathrm{DFT}_N \, |\psi_{x_0,r}\rangle &= \left(\frac{1}{\sqrt{N}} \sum_{i,j=0}^{N-1} \omega_N^{ij} \, |i\rangle \langle j| \right) \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle \right) \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \left(\sum_{k=0}^{N/r-1} \omega_N^{i(x_0 + k \cdot r)} \right) |i\rangle \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{i x_0} \underbrace{\left(\sum_{k=0}^{N/r-1} \omega_N^{ikr} \right)}_{=:\alpha_i} |i\rangle
\end{aligned}$$

Extracting the Period of a Function

Theorem (Fourier duality)

Let $N \in \mathbb{N}$ and let $r \in \mathbb{Z}_N$ be a divisor of N, and let $x_0 \in \mathbb{Z}_N$. Then

$$\mathrm{DFT}_N \left|\psi_{x_0,r}\right\rangle = \mathrm{DFT}_N \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} \left|x_0 + k \cdot r\right\rangle \right) = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \omega_N^{\ell x_0 \frac{N}{r}} \left|\ell \frac{N}{r}\right\rangle$$

Proof:

$$\begin{aligned}
\mathrm{DFT}_N \left|\psi_{x_0,r}\right\rangle &= \left(\frac{1}{\sqrt{N}} \sum_{i,j=0}^{N-1} \omega_N^{ij} \left|i\right\rangle \left\langle j\right| \right) \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} \left|x_0 + k \cdot r\right\rangle \right) \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \left(\sum_{k=0}^{N/r-1} \omega_N^{i(x_0+k\cdot r)} \right) \left|i\right\rangle \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{ix_0} \underbrace{\left(\sum_{k=0}^{N/r-1} \omega_N^{ikr} \right)}_{=:\alpha_i} \left|i\right\rangle
\end{aligned}$$

Extracting the Period of a Function

Theorem (Fourier duality)

Let $N \in \mathbb{N}$ and let $r \in \mathbb{Z}_N$ be a divisor of N, and let $x_0 \in \mathbb{Z}_N$. Then

$$\mathrm{DFT}_N \left|\psi_{x_0,r}\right\rangle = \mathrm{DFT}_N \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle \right) = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \omega_N^{\ell x_0 \frac{N}{r}} \left| \ell \frac{N}{r} \right\rangle$$

Proof:

$$\begin{aligned}
\mathrm{DFT}_N \left|\psi_{x_0,r}\right\rangle &= \left(\frac{1}{\sqrt{N}} \sum_{i,j=0}^{N-1} \omega_N^{ij} |i\rangle \langle j| \right) \left(\frac{1}{\sqrt{N/r}} \sum_{k=0}^{N/r-1} |x_0 + k \cdot r\rangle \right) \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \left(\sum_{k=0}^{N/r-1} \omega_N^{i(x_0+k\cdot r)} \right) |i\rangle \\
&= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{i x_0} \underbrace{\left(\sum_{k=0}^{N/r-1} \omega_N^{ikr} \right)}_{=:\alpha_i} |i\rangle
\end{aligned}$$

Constructive / Destructive Interference

Computing the coefficients α_i

For each $i = 0, \ldots, N-1$ we have to compute $\alpha_i = \sum_{k=0}^{N/r-1} \omega_N^{ikr}$.

- Case 1: $i = \frac{N}{r}\ell$ for some $\ell = 0, \ldots, r-1$. Then

$$\alpha_i = \sum_{k=0}^{N/r-1} \omega_N^{\frac{N}{r}\ell kr} = \sum_{k=0}^{N/r-1} 1 = \frac{N}{r}.$$

- Case 2: $i \neq \frac{N}{r}\ell$ for all $\ell = 0, \ldots, r-1$. Then

$$\alpha_i = \sum_{k=0}^{N/r-1} \left(\omega_N^{\frac{N}{r}\ell kr}\right)^k = 0$$

by the Character Lemma.

Constructive / Destructive Interference

Computing the coefficients α_i

For each $i = 0, \ldots, N-1$ we have to compute $\alpha_i = \sum_{k=0}^{N/r-1} \omega_N^{ikr}$.

- Case 1: $i = \frac{N}{r}\ell$ for some $\ell = 0, \ldots, r-1$. Then

$$\alpha_i = \sum_{k=0}^{N/r-1} \omega_N^{\frac{N}{r}\ell kr} = \sum_{k=0}^{N/r-1} 1 = \frac{N}{r}.$$

- Case 2: $i \neq \frac{N}{r}\ell$ for all $\ell = 0, \ldots, r-1$. Then

$$\alpha_i = \sum_{k=0}^{N/r-1} \left(\omega_N^{\frac{N}{r}\ell kr}\right)^k = 0$$

by the Character Lemma.

Constructive / Destructive Interference

End of proof (Fourier duality)

$$\mathrm{DFT}_N \left|\psi_{x_0,r}\right\rangle = \frac{\sqrt{r}}{N}\sum_{i=0}^{N-1}\omega_N^{ix_0}\left(\sum_{k=0}^{N/r-1}\omega_N^{ikr}\right)|i\rangle = \frac{\sqrt{r}}{N}\sum_{i=0}^{N-1}\omega_N^{ix_0}\alpha_i\,|i\rangle$$

$$= \frac{\sqrt{r}}{N}\sum_{\ell=0}^{r-1}\omega_N^{\ell\frac{N}{r}x_0}\frac{N}{r}\left|\ell\frac{N}{r}\right\rangle = \frac{1}{\sqrt{r}}\sum_{\ell=0}^{r-1}\omega_N^{\ell\frac{N}{r}x_0}\left|\ell\frac{N}{r}\right\rangle$$

What happens if r does not divide N?

In this case the state can be approximated very accurately by

$$\mathrm{DFT}_N\left(\sum_k |x_0 + k\cdot r\rangle\right) \approx \sum_k \omega_N^{\ell\mu x_0}\,|\ell\mu\rangle$$

with an element $\mu \in Z_N$ such that $\mu r \approx N$.

Constructive / Destructive Interference

End of proof (Fourier duality)

$$\begin{aligned}
\mathrm{DFT}_N \, |\psi_{x_0,r}\rangle &= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{i x_0} \left(\sum_{k=0}^{N/r-1} \omega_N^{ikr} \right) |i\rangle = \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{i x_0} \alpha_i \, |i\rangle \\
&= \frac{\sqrt{r}}{N} \sum_{\ell=0}^{r-1} \omega_N^{\ell \frac{N}{r} x_0} \frac{N}{r} \left| \ell \frac{N}{r} \right\rangle = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \omega_N^{\ell \frac{N}{r} x_0} \left| \ell \frac{N}{r} \right\rangle
\end{aligned}$$

What happens if r does not divide N?

In this case the state can be approximated very accurately by

$$\mathrm{DFT}_N \left(\sum_k |x_0 + k \cdot r\rangle \right) \approx \sum_k \omega_N^{\ell \mu x_0} \, |\ell \mu\rangle$$

with an element $\mu \in Z_N$ such that $\mu r \approx N$.

Constructive / Destructive Interference

End of proof (Fourier duality)

$$\begin{aligned}
\mathrm{DFT}_N \, |\psi_{x_0,r}\rangle &= \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{ix_0} \left(\sum_{k=0}^{N/r-1} \omega_N^{ikr} \right) |i\rangle = \frac{\sqrt{r}}{N} \sum_{i=0}^{N-1} \omega_N^{ix_0} \alpha_i \, |i\rangle \\
&= \frac{\sqrt{r}}{N} \sum_{\ell=0}^{r-1} \omega_N^{\ell \frac{N}{r} x_0} \frac{N}{r} \left| \ell \frac{N}{r} \right\rangle = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \omega_N^{\ell \frac{N}{r} x_0} \left| \ell \frac{N}{r} \right\rangle
\end{aligned}$$

What happens if r does not divide N?

In this case the state can be approximated very accurately by

$$\mathrm{DFT}_N \left(\sum_k |x_0 + k \cdot r\rangle \right) \approx \sum_k \omega_N^{\ell \mu x_0} \, |\ell \mu\rangle$$

with an element $\mu \in Z_N$ such that $\mu r \approx N$.

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \,|\ell \frac{N}{r}\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Solution to the Period Extraction Problem

Quantum algorithm

Given: Modular exponentiation function $U_a : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$.

Task: Find the order r of a modulo N.

Repeat the following steps one time: (w/o normalizations, $M = 2^m >> N$)

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\sum_{x=0}^{M-1} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\sum_{x=0}^{M-1} |x\rangle\,|a^x \bmod N\rangle$
4. Measure second register: $\sum_{k=0}^{M/r-1} |x_0 + k \cdot r\rangle$
5. Compute DFT_M on first register: $\approx \sum_{\ell=0}^{r-1} \omega_M^{\ell \frac{N}{r} x_0} \left|\ell \frac{N}{r}\right\rangle$
6. Measure first register: Sample a rational number $\frac{p}{q}$ which is very close to $\frac{\ell_0}{r}$.

How can we classically reconstruct r from $\frac{p}{q}$?

Classical Post-Processing

How to obtain r from $\frac{p}{q}$?

Since we know N and ℓ_0, we could easily compute r from $\ell_0 \frac{N}{r}$. However, we are just given $\frac{p}{q}$ for which we only know that the following holds

$$\left| \frac{p}{q} - \frac{\ell_0}{r} \right| < \frac{1}{2r^2}$$

Diophantine approximation

We apply the continued fractions algorithm to $\frac{p}{q}$. This will lead to several principal fractions and actually $\frac{\ell_0}{r}$ will be one of them. Note that we can check whether a candidate r is indeed the order.

Theorem (Shor '94)

$$\text{FACTORING} \in \text{BQP}.$$

How to obtain r from $\frac{p}{q}$?

Since we know N and ℓ_0, we could easily compute r from $\ell_0 \frac{N}{r}$. However, we are just given $\frac{p}{q}$ for which we only know that the following holds

$$\left| \frac{p}{q} - \frac{\ell_0}{r} \right| < \frac{1}{2r^2}$$

Diophantine approximation

We apply the continued fractions algorithm to $\frac{p}{q}$. This will lead to several principal fractions and actually $\frac{\ell_0}{r}$ will be one of them. Note that we can check whether a candidate r is indeed the order.

Theorem (Shor '94)

FACTORING $\in$ BQP.

Classical Post-Processing

How to obtain r from $\frac{p}{q}$?

Since we know N and ℓ_0, we could easily compute r from $\ell_0 \frac{N}{r}$. However, we are just given $\frac{p}{q}$ for which we only know that the following holds

$$\left| \frac{p}{q} - \frac{\ell_0}{r} \right| < \frac{1}{2r^2}$$

Diophantine approximation

We apply the continued fractions algorithm to $\frac{p}{q}$. This will lead to several principal fractions and actually $\frac{\ell_0}{r}$ will be one of them. Note that we can check whether a candidate r is indeed the order.

Theorem (Shor '94)

$$\text{FACTORING} \in \text{BQP}.$$

Diophantine Approximation

The continued fractions algorithm

Input: $x \in \mathbb{R}$. Output: Sequence of $b_i \in \mathbb{Z}$ (possibly infinite) which represents x. Let $b_0 := \lfloor x \rfloor$, $x_1 := \frac{1}{x - b_0}$, $b_1 := \lfloor x_1 \rfloor$, $x_2 := \frac{1}{x_1 - b_1}$, ... Then

$$x = b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \frac{1}{\dots}}}$$

Example with rational input

Suppose that $x = \frac{5021264471}{8589934592}$. The algorithm results in (vector of b_i's):

$$[1, 1, 2, 2, 5, 3, 1, 1, 3, 1, 11, 1, 1, 21, 1, 2, 5, 1, 1, 1, 1, 1, 2, 1, 2, 1, 3]$$

Consider the *convergents* C_n which are obtained by truncating after n steps:

n	1	2	3	4	5	6	7	8	9	10
C_n	1	$\frac{1}{2}$	$\frac{3}{5}$	$\frac{7}{12}$	$\frac{38}{65}$	$\frac{121}{207}$	$\frac{159}{272}$	$\frac{280}{479}$	$\frac{999}{1709}$	$\frac{1279}{2188}$

n	11	12	13	14	...	27
C_n	$\frac{15608}{25777}$	$\frac{16347}{27965}$	$\frac{31415}{53742}$	$\frac{676062}{1156547}$	...	$\frac{5021264471}{8589934592}$

Diophantine Approximation

The continued fractions algorithm

Input: $x \in \mathbb{R}$. Output: Sequence of $b_i \in \mathbb{Z}$ (possibly infinite) which represents x. Let $b_0 := \lfloor x \rfloor$, $x_1 := \frac{1}{x - b_0}$, $b_1 := \lfloor x_1 \rfloor$, $x_2 := \frac{1}{x_1 - b_1}$, ... Then

$$x = b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \frac{1}{\dots}}}$$

Example with rational input

Suppose that $x = \frac{5021264471}{8589934592}$. The algorithm results in (vector of b_i's):

$$[1, 1, 2, 2, 5, 3, 1, 1, 3, 1, 11, 1, 1, 21, 1, 2, 5, 1, 1, 1, 1, 1, 2, 1, 2, 1, 3]$$

Consider the *convergents* C_n which are obtained by truncating after n steps:

n	1	2	3	4	5	6	7	8	9	10
C_n	1	$\frac{1}{2}$	$\frac{3}{5}$	$\frac{7}{12}$	$\frac{38}{65}$	$\frac{121}{207}$	$\frac{159}{272}$	$\frac{280}{479}$	$\frac{999}{1709}$	$\frac{1279}{2188}$

n	11	12	13	14	...	27
C_n	$\frac{15608}{25777}$	$\frac{16347}{27965}$	$\frac{31415}{53742}$	$\frac{676062}{1156547}$	...	$\frac{5021264471}{8589934592}$

Diophantine Approximation

Lagrange's Theorem

Let $x \in \mathbb{Q}$ and assume that we are given $\frac{p}{q} \in \mathbb{Q}$ such that

$$\left| x - \frac{p}{q} \right| \leq \frac{1}{2q^2}$$

Then x is a convergent C_n of $\frac{p}{q}$, namely that for which $|C_n - \frac{p}{q}| < \frac{1}{2q^2}$.

Example (cont'd)

Let $y = \frac{31415}{53742}$ be the inverse period. Since denominator of y is $\leq 2^{16}$ we can work with precision $\leq 2^{33}$. Suppose we measure $\frac{p}{q} = \frac{5021264471}{8589934592}$:

n	1	2	3	4	5	6	7	8	9	10
C_n	1	$\frac{1}{2}$	$\frac{3}{5}$	$\frac{7}{12}$	$\frac{38}{65}$	$\frac{121}{207}$	$\frac{159}{272}$	$\frac{280}{479}$	$\frac{999}{1709}$	$\frac{1279}{2188}$

n	11	12	13	14	...	27
C_n	$\frac{15608}{25777}$	$\frac{16347}{27965}$	$\frac{31415}{53742}$	$\frac{676062}{1156547}$	...	$\frac{5021264471}{8589934592}$

Diophantine Approximation

Lagrange's Theorem

Let $x \in \mathbb{Q}$ and assume that we are given $\frac{p}{q} \in \mathbb{Q}$ such that

$$\left| x - \frac{p}{q} \right| \leq \frac{1}{2q^2}$$

Then x is a convergent C_n of $\frac{p}{q}$, namely that for which $|C_n - \frac{p}{q}| < \frac{1}{2q^2}$.

Example (cont'd)

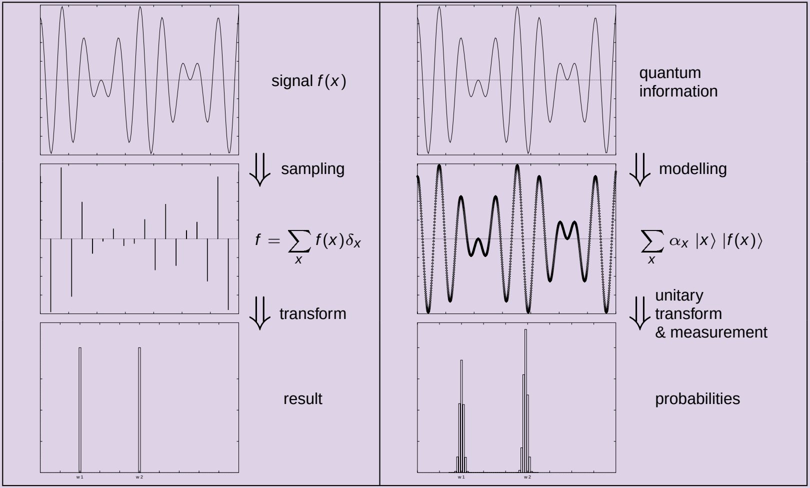
Let $y = \frac{31415}{53742}$ be the inverse period. Since denominator of y is $\leq 2^{16}$ we can work with precision $\leq 2^{33}$. Suppose we measure $\frac{p}{q} = \frac{5021264471}{8589934592}$:

n	1	2	3	4	5	6	7	8	9	10
C_n	1	$\frac{1}{2}$	$\frac{3}{5}$	$\frac{7}{12}$	$\frac{38}{65}$	$\frac{121}{207}$	$\frac{159}{272}$	$\frac{280}{479}$	$\frac{999}{1709}$	$\frac{1279}{2188}$

n	11	12	13	14	...	27
C_n	$\frac{15608}{25777}$	$\frac{16347}{27965}$	$\frac{31415}{53742}$	$\frac{676062}{1156547}$	...	$\frac{5021264471}{8589934592}$

Parallels between DSP and QC

Digital Signal Processing | Quantum Computing



Example: Fast Fourier Transform (FFT)

Cooley-Tukey FFT

The matrix $\mathrm{DFT}_N = \frac{1}{\sqrt{N}} \left[\omega_N^{k \cdot \ell} \right]_{k,\ell=0\ldots N-1}$, where $\omega_N = e^{2\pi i/N}$ can be written as a short product of sparse matrices.

$$\mathrm{DFT}_4 = \Pi_{rev} \cdot (\mathbf{1}_2 \otimes \mathrm{DFT}_2) \cdot \mathrm{diag} \cdot (\mathrm{DFT}_2 \otimes \mathbf{1}_2)$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & i \end{bmatrix} \cdot \begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{bmatrix}$$

FFT Theorem

Multiplication with DFT_N can be performed classically in $O(N \log N)$ elementary operations.

We can do much better on a quantum computer!

Example: Fast Fourier Transform (FFT)

Cooley-Tukey FFT

The matrix $\mathrm{DFT}_N = \frac{1}{\sqrt{N}} \left[\omega_N^{k \cdot \ell} \right]_{k,\ell=0\ldots N-1}$, where $\omega_N = e^{2\pi i/N}$ can be written as a short product of sparse matrices.

$$\mathrm{DFT}_4 = \Pi_{rev} \cdot (\mathbf{1}_2 \otimes \mathrm{DFT}_2) \cdot \mathrm{diag} \cdot (\mathrm{DFT}_2 \otimes \mathbf{1}_2)$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & i \end{bmatrix} \cdot \begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{bmatrix}$$

FFT Theorem

Multiplication with DFT_N can be performed classically in $O(N \log N)$ elementary operations.

We can do much better on a quantum computer!

Example: Fast Fourier Transform (FFT)

Cooley-Tukey FFT

The matrix $\mathrm{DFT}_N = \frac{1}{\sqrt{N}} \left[\omega_N^{k \cdot \ell} \right]_{k,\ell=0 \ldots N-1}$, where $\omega_N = e^{2\pi i/N}$ can be written as a short product of sparse matrices.

$$\mathrm{DFT}_4 = \Pi_{rev} \cdot (\mathbf{1}_2 \otimes \mathrm{DFT}_2) \cdot \mathrm{diag} \cdot (\mathrm{DFT}_2 \otimes \mathbf{1}_2)$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & i \end{bmatrix} \cdot \begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{bmatrix}$$

FFT Theorem

Multiplication with DFT_N can be performed classically in $O(N \log N)$ elementary operations.

We can do much better on a quantum computer!

Example: Fast Fourier Transform (FFT)

Cooley-Tukey FFT

The matrix $\mathrm{DFT}_N = \frac{1}{\sqrt{N}} \left[\omega_N^{k \cdot \ell} \right]_{k,\ell=0\ldots N-1}$, where $\omega_N = e^{2\pi i/N}$ can be written as a short product of sparse matrices.

$$\mathrm{DFT}_4 \quad = \quad \Pi_{rev} \quad \cdot (\mathbf{1}_2 \otimes \mathrm{DFT}_2) \cdot \quad \mathrm{diag} \quad \cdot \quad (\mathrm{DFT}_2 \otimes \mathbf{1}_2)$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & 1 & 1 \\ & & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & i \end{bmatrix} \cdot \begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \end{bmatrix}$$

FFT Theorem

Multiplication with DFT_N can be performed classically in $O(N \log N)$ elementary operations.

We can do much better on a quantum computer!

Fast Fourier Transform

Cooley-Tukey Formula

$$\Pi_n \, \mathrm{DFT}_{2^n} = \left(\begin{array}{c|c} \mathrm{DFT}_{2^{n-1}} & \mathrm{DFT}_{2^{n-1}} \\ \hline \mathrm{DFT}_{2^{n-1}} \, D_n & -\mathrm{DFT}_{2^{n-1}} \, D_n \end{array}\right)$$

$$= (\mathbf{1}_2 \otimes \mathrm{DFT}_{2^{n-1}}) \cdot (\mathbf{1}_{2^{n-1}} \oplus D_n) \cdot (\mathrm{DFT}_2 \otimes \mathbf{1}_{2^{n-1}})$$

Factorization of the **twiddle factors**

$$D_n := \begin{pmatrix} 1 & & & \\ & \omega_{2^n} & & \\ & & \omega_{2^n}^2 & & \\ & & & \ddots & \\ & & & & \omega_{2^n}^{2^{n-1}-1} \end{pmatrix} = \begin{pmatrix} 1 & \\ & \omega_{2^n}^{2^{n-2}} \end{pmatrix} \otimes \ldots \otimes \begin{pmatrix} 1 & \\ & \omega_{2^n} \end{pmatrix}$$

Fast Fourier Transform

Cooley-Tukey Formula

$$\Pi_n \, \mathrm{DFT}_{2^n} = \left(\begin{array}{c|c} \mathrm{DFT}_{2^{n-1}} & \mathrm{DFT}_{2^{n-1}} \\ \hline \mathrm{DFT}_{2^{n-1}} \, D_n & -\mathrm{DFT}_{2^{n-1}} \, D_n \end{array}\right)$$

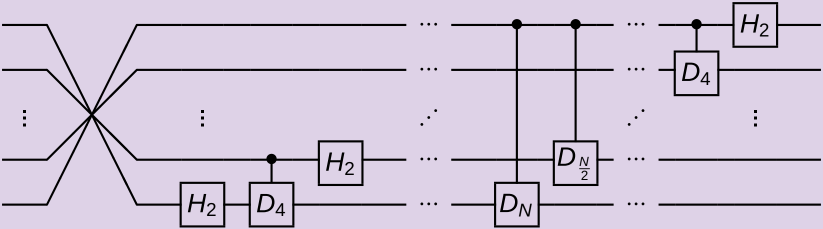
$$= (\mathbf{1}_2 \otimes \mathrm{DFT}_{2^{n-1}}) \cdot (\mathbf{1}_{2^{n-1}} \oplus D_n) \cdot (\mathrm{DFT}_2 \otimes \mathbf{1}_{2^{n-1}})$$

Factorization of the **twiddle factors**

$$D_n := \begin{pmatrix} 1 & & & \\ & \omega_{2^n} & & \\ & & \omega_{2^n}^2 & & \\ & & & \ddots & \\ & & & & \omega_{2^n}^{2^{n-1}-1} \end{pmatrix} = \begin{pmatrix} 1 & \\ & \omega_{2^n}^{2^{n-2}} \end{pmatrix} \otimes \ldots \otimes \begin{pmatrix} 1 & \\ & \omega_{2^n} \end{pmatrix}$$

Cooley-Tukey Realization of DFT

Quantum circuit for DFT_N

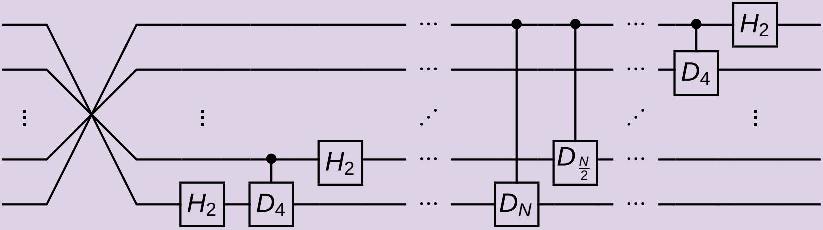


Cost

Classical Computer	Quantum Computer
$T(N) = 2\,T(N/2) + O(N)$	$T(N) = T(N/2) + O(\log N)$
$T(N) = O(N \log N)$	$T(N) = O(\log^2 N)$

Cooley-Tukey Realization of DFT

Quantum circuit for DFT$_N$



Cost

Classical Computer	Quantum Computer
$T(N) = 2\,T(N/2) + O(N)$	$T(N) = T(N/2) + O(\log N))$
$T(N) = O(N \log N)$	$T(N) = O(\log^2 N)$

Shor's Algorithm: Gate Counts

Modular exponentiation

$U: |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$, where N is the k-bit number to be factored, and x is a $2k$-bit number. Implementation using $396k(k^2 + O(k))$ elementary operations [Beckman et al.].

Quantum Fourier Transform:

DFT_{2^n} needs $\frac{1}{2}n(n-1)$ two-qubit gates and n one-qubit gates.

An upper bound on the resources for k-bit number N

About $400k^3$ operations are needed (can be improved to $O(k^2 \log k \log\log k))$). The space needed is $5k + 1$ qubits.

Example: Factoring 128-bit and 1024-bit numbers

For 128-bit we need $840 \cdot 10^6$ operations and 641 qubits. For 1024-bit number $429 \cdot 10^9$ operations and 5121 qubits.

Shor's Algorithm: Gate Counts

Modular exponentiation

$U : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$, where N is the k-bit number to be factored, and x is a $2k$-bit number. Implementation using $396k(k^2 + O(k))$ elementary operations [Beckman et al.].

Quantum Fourier Transform:

DFT_{2^n} needs $\frac{1}{2}n(n-1)$ two-qubit gates and n one-qubit gates.

An upper bound on the resources for k-bit number N

About $400k^3$ operations are needed (can be improved to $O(k^2 \log k \log\log k)$)). The space needed is $5k+1$ qubits.

Example: Factoring 128-bit and 1024-bit numbers

For 128-bit we need $840 \cdot 10^6$ operations and 641 qubits. For 1024-bit number $429 \cdot 10^9$ operations and 5121 qubits.

Shor's Algorithm: Gate Counts

Modular exponentiation

$U : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$, where N is the k-bit number to be factored, and x is a $2k$-bit number. Implementation using $396k(k^2 + O(k))$ elementary operations [Beckman et al.].

Quantum Fourier Transform:

DFT_{2^n} needs $\frac{1}{2}n(n-1)$ two-qubit gates and n one-qubit gates.

An upper bound on the resources for k-bit number N

About $400k^3$ operations are needed (can be improved to $O(k^2 \log k \log\log k))$). The space needed is $5k+1$ qubits.

Example: Factoring 128-bit and 1024-bit numbers

For 128-bit we need $840 \cdot 10^6$ operations and 641 qubits. For 1024-bit number $429 \cdot 10^9$ operations and 5121 qubits.

Shor's Algorithm: Gate Counts

Modular exponentiation

$U : |x\rangle\,|0\rangle \mapsto |x\rangle\,|a^x \bmod N\rangle$, where N is the k-bit number to be factored, and x is a $2k$-bit number. Implementation using $396k(k^2 + O(k))$ elementary operations [Beckman et al.].

Quantum Fourier Transform:

DFT_{2^n} needs $\frac{1}{2}n(n-1)$ two-qubit gates and n one-qubit gates.

An upper bound on the resources for k-bit number N

About $400k^3$ operations are needed (can be improved to $O(k^2 \log k \log\log k)))$. The space needed is $5k + 1$ qubits.

Example: Factoring 128-bit and 1024-bit numbers

For 128-bit we need $840 \cdot 10^6$ operations and 641 qubits. For 1024-bit number $429 \cdot 10^9$ operations and 5121 qubits.

HSP: History of the Problem

D. Simon, 1994

Hidden Subgroups in $(\mathbb{Z}_2)^n$.

P. Shor, 1994

$$\left.\begin{array}{l} \text{– Factoring} \\ \text{– Discrete Logarithm} \end{array}\right\} \quad \begin{array}{l} \text{Hidden Subgroups in } \mathbb{Z}_M \\ \text{resp. } \mathbb{Z}_M \times \mathbb{Z}_M \end{array}$$

Kitaev '95, Brassard & Høyer '97, Mosca & Ekert '98

Generalization to arbitrary abelian groups.

Open Problems

Can the Hidden Subgroups Problem be solved for

- Any non-abelian group? (yes!)
- All non-abelian groups? (don't know)
- "Interesting" non-abelian groups? (progress, but still open)

HSP: History of the Problem

D. Simon, 1994

Hidden Subgroups in $(Z_2)^n$.

P. Shor, 1994

$$\left.\begin{array}{l} \text{– Factoring} \\ \text{– Discrete Logarithm} \end{array}\right\} \begin{array}{l} \text{Hidden Subgroups in } Z_M \\ \text{resp. } Z_M \times Z_M \end{array}$$

Kitaev '95, Brassard & Høyer '97, Mosca & Ekert '98

Generalization to arbitrary abelian groups.

Open Problems

Can the Hidden Subgroups Problem be solved for

- Any non-abelian group? (yes!)
- All non-abelian groups? (don't know)
- "Interesting" non-abelian groups? (progress, but still open)

HSP: History of the Problem

D. Simon, 1994

Hidden Subgroups in $(\mathbb{Z}_2)^n$.

P. Shor, 1994

$$\left.\begin{array}{l} \text{– Factoring} \\ \text{– Discrete Logarithm} \end{array}\right\} \quad \begin{array}{l} \text{Hidden Subgroups in } \mathbb{Z}_M \\ \text{resp. } \mathbb{Z}_M \times \mathbb{Z}_M \end{array}$$

Kitaev '95, Brassard & Høyer '97, Mosca & Ekert '98

Generalization to arbitrary abelian groups.

Open Problems

Can the Hidden Subgroups Problem be solved for

- Any non-abelian group? (yes!)
- All non-abelian groups? (don't know)
- "Interesting" non-abelian groups? (progress, but still open)

HSP: History of the Problem

D. Simon, 1994

Hidden Subgroups in $(Z_2)^n$.

P. Shor, 1994

$$\left.\begin{array}{ll} - & \text{Factoring} \\ - & \text{Discrete Logarithm} \end{array}\right\} \begin{array}{l} \text{Hidden Subgroups in } Z_M \\ \text{resp. } Z_M \times Z_M \end{array}$$

Kitaev '95, Brassard & Høyer '97, Mosca & Ekert '98

Generalization to arbitrary abelian groups.

Open Problems

Can the Hidden Subgroups Problem be solved for

- Any non-abelian group? (yes!)
- All non-abelian groups? (don't know)
- "Interesting" non-abelian groups? (progress, but still open)

HSP: History of the Problem

D. Simon, 1994

Hidden Subgroups in $(Z_2)^n$.

P. Shor, 1994

$$\left.\begin{array}{ll} - & \text{Factoring} \\ - & \text{Discrete Logarithm} \end{array}\right\} \quad \begin{array}{l} \text{Hidden Subgroups in } Z_M \\ \text{resp. } Z_M \times Z_M \end{array}$$

Kitaev '95, Brassard & Høyer '97, Mosca & Ekert '98

Generalization to arbitrary abelian groups.

Open Problems

Can the Hidden Subgroups Problem be solved for

- Any non-abelian group? (yes!)
- All non-abelian groups? (don't know)
- "Interesting" non-abelian groups? (progress, but still open)

HSP: History of the Problem

D. Simon, 1994

Hidden Subgroups in $(Z_2)^n$.

P. Shor, 1994

$$\left.\begin{array}{ll} - & \text{Factoring} \\ - & \text{Discrete Logarithm} \end{array}\right\} \quad \begin{array}{l} \text{Hidden Subgroups in } Z_M \\ \text{resp. } Z_M \times Z_M \end{array}$$

Kitaev '95, Brassard & Høyer '97, Mosca & Ekert '98

Generalization to arbitrary abelian groups.

Open Problems

Can the Hidden Subgroups Problem be solved for

- Any non-abelian group? (yes!)
- All non-abelian groups? (don't know)
- "Interesting" non-abelian groups? (progress, but still open)

HSP: History of the Problem

D. Simon, 1994

Hidden Subgroups in $(Z_2)^n$.

P. Shor, 1994

$$\left.\begin{array}{ll} - & \text{Factoring} \\ - & \text{Discrete Logarithm} \end{array}\right\} \quad \begin{array}{l} \text{Hidden Subgroups in } Z_M \\ \text{resp. } Z_M \times Z_M \end{array}$$

Kitaev '95, Brassard & Høyer '97, Mosca & Ekert '98

Generalization to arbitrary abelian groups.

Open Problems

Can the Hidden Subgroups Problem be solved for

- Any non-abelian group? (yes!)
- All non-abelian groups? (don't know)
- "Interesting" non-abelian groups? (progress, but still open)

The Hidden Subgroup Problem (HSP)

Definition of the problem

Given: Finite group G, finite set S, map $f : G \to S$

Promise: There exists $H \subseteq G$ where

- f constant on G/H,
- $g_1H \neq g_2H$ implies $f(g_1) \neq f(g_2)$.

Problem: Find generators for H.

Note

- This is a natural generalization of Simon's problem.
- There $G = F_2^n$ and in addition we know $|H| = |\langle s\rangle| = 2$.
- In fact, the HSP for *any* subgroup $H \leq \mathbb{F}_2^n$ can be solved efficiently.

Mathematically

$$\begin{tikzcd} G \arrow[r, "f"] \arrow[d, "\pi"'] & S \\ G/H \arrow[ur, "i"'] & \end{tikzcd}$$

The Hidden Subgroup Problem (HSP)

Definition of the problem

Given: Finite group G, finite set S, map $f : G \to S$

Promise: There exists $H \subseteq G$ where

- f constant on G/H,
- $g_1H \neq g_2H$ implies $f(g_1) \neq f(g_2)$.

Problem: Find generators for H.

Note

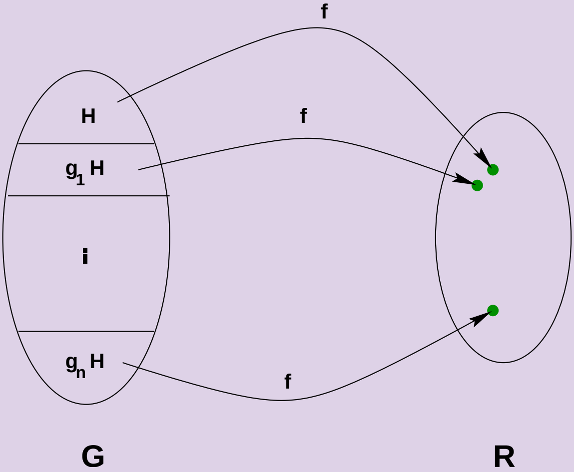
- This is a natural generalization of Simon's problem.
- There $G = F_2^n$ and in addition we know $|H| = |\langle s \rangle| = 2$.
- In fact, the HSP for *any* subgroup $H \leq \mathrm{F}_2^n$ can be solved efficiently.

Mathematically

$$\begin{tikzcd} G \arrow[r, "f"] \arrow[d, "\pi"'] & S \\ G/H \arrow[ur, "i"'] & \end{tikzcd}$$

HSP: Separation of Pre-images

Visualization of the cosets of G



Examples for Hidden Subgroup Problems

Simon's Problem

Find hidden subgroup in $G = Z_2^n$ of a *black-box* function $f : G \to \{0,1\}^m$, where $m \le n$ and $x \in y + H \Leftrightarrow f(x) = f(y)$.

Factoring

Find hidden subgroup in $G = Z_M$ with respect to the function

$$f(x) := a^x \bmod N.$$

Discrete logarithm problem

Find hidden subgroup in $G = Z_M \times Z_M$ with respect to the function

$$f(x, y) := a^x b^{-y} \bmod p.$$

The graph isomorphism problem

Can be reduced to the problem of finding certain hidden subgroups of order 2 in the non-abelian group $G = S_n \wr S_2$.

Examples for Hidden Subgroup Problems

Simon's Problem

Find hidden subgroup in $G = Z_2^n$ of a *black-box* function $f : G \to \{0,1\}^m$, where $m \leq n$ and $x \in y + H \Leftrightarrow f(x) = f(y)$.

Factoring

Find hidden subgroup in $G = Z_M$ with respect to the function

$$f(x) := a^x \bmod N.$$

Discrete logarithm problem

Find hidden subgroup in $G = Z_M \times Z_M$ with respect to the function

$$f(x,y) := a^x b^{-y} \bmod p.$$

The graph isomorphism problem

Can be reduced to the problem of finding certain hidden subgroups of order 2 in the non-abelian group $G = S_n \wr S_2$.

Examples for Hidden Subgroup Problems

Simon's Problem

Find hidden subgroup in $G = Z_2^n$ of a *black-box* function $f : G \to \{0,1\}^m$, where $m \le n$ and $x \in y + H \Leftrightarrow f(x) = f(y)$.

Factoring

Find hidden subgroup in $G = Z_M$ with respect to the function

$$f(x) := a^x \bmod N.$$

Discrete logarithm problem

Find hidden subgroup in $G = Z_M \times Z_M$ with respect to the function

$$f(x,y) := a^x b^{-y} \bmod p.$$

The graph isomorphism problem

Can be reduced to the problem of finding certain hidden subgroups of order 2 in the non-abelian group $G = S_n \wr S_2$.

Examples for Hidden Subgroup Problems

Simon's Problem

Find hidden subgroup in $G = Z_2^n$ of a *black-box* function $f : G \to \{0,1\}^m$, where $m \leq n$ and $x \in y + H \Leftrightarrow f(x) = f(y)$.

Factoring

Find hidden subgroup in $G = Z_M$ with respect to the function

$$f(x) := a^x \bmod N.$$

Discrete logarithm problem

Find hidden subgroup in $G = Z_M \times Z_M$ with respect to the function

$$f(x,y) := a^x b^{-y} \bmod p.$$

The graph isomorphism problem

Can be reduced to the problem of finding certain hidden subgroups of order 2 in the non-abelian group $G = S_n \wr S_2$.

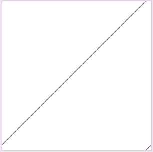
Duality of the Fourier Transform

Basic identity

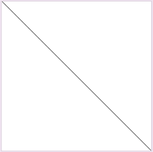
$$\mathrm{DFT}_A\left(\frac{1}{\sqrt{|U|}}\sum_{\mathbf{x}\in U+c}|\mathbf{x}\rangle\right)=\frac{1}{\sqrt{|U^\perp|}}\sum_{\mathbf{y}\in U^\perp}\varphi_{c,y}\cdot|\mathbf{y}\rangle$$

Geometric interpretation

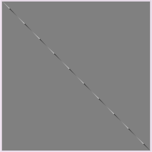
Shifted Line



DFT $\longrightarrow$



Amplitude



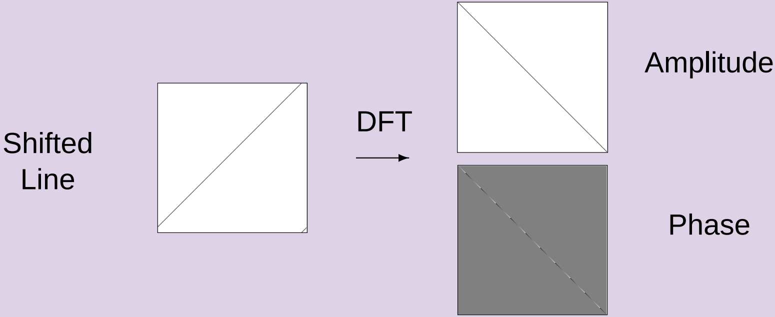
Phase

Duality of the Fourier Transform

Basic identity

$$\mathrm{DFT}_A\left(\frac{1}{\sqrt{|U|}}\sum_{\mathbf{x}\in U+c}|\mathbf{x}\rangle\right) = \frac{1}{\sqrt{|U^\perp|}}\sum_{\mathbf{y}\in U^\perp}\varphi_{c,y}\cdot|\mathbf{y}\rangle$$

Geometric interpretation



Generalized Fourier Transforms

Definition of DFT_G

Any isomorphism $\Phi : \mathbb{C}[G] \longrightarrow \bigoplus_{k=1}^{m} \mathbb{C}^{d_k \times d_k}$ of the group algebra and a direct sum of irreducible matrix algebras.

Some properties of DFT_G

- Defined for arbitrary finite groups.
- The isomorphism Φ is realized by a unitary matrix

$$\mathrm{DFT}_G = \frac{1}{\sqrt{|G||H|}} \sum_{\rho,i,j} \sqrt{d_\rho} \sum_{h \in H} \rho_{ij}(gh) \left|\rho, i, j\right\rangle \left\langle g\right| .$$

- DFT_G decomposes the regular representation ϕ of G:

$$\phi^{\mathrm{DFT}_G} = \mathrm{DFT}_G{}^{\dagger} \, \phi \, \mathrm{DFT}_G = \bigoplus_{k=1}^{m} 1_{d_{\rho_k}} \otimes \rho_k .$$

Generalized Fourier Transforms

Definition of DFT_G

Any isomorphism $\Phi : \mathbb{C}[G] \longrightarrow \bigoplus_{k=1}^{m} \mathbb{C}^{d_k \times d_k}$ of the group algebra and a direct sum of irreducible matrix algebras.

Some properties of DFT_G

- Defined for arbitrary finite groups.
- The isomorphism Φ is realized by a unitary matrix

$$\mathrm{DFT}_G = \frac{1}{\sqrt{|G||H|}} \sum_{\rho,i,j} \sqrt{d_\rho} \sum_{h \in H} \rho_{ij}(gh) \, |\rho, i, j\rangle \langle g| \, .$$

- DFT_G decomposes the regular representation ϕ of G:

$$\phi^{\mathrm{DFT}_G} = {\mathrm{DFT}_G}^{\dagger} \, \phi \, \mathrm{DFT}_G = \bigoplus_{k=1}^{m} 1_{d_{\rho_k}} \otimes \rho_k \, .$$

Hidden Subgroup Problems: Standard Algorithm

Quantum algorithm

Given: finite group G with hidden subgroup $H \leq G$.
Task: Find a set of generators for H.

Repeat the following steps $poly(n)$ many times:

1. Initialize two quantum registers: $|0\rangle\,|0\rangle$
2. Equal distribution on first register: $\frac{1}{\sqrt{|G|}} \sum_{x \in G} |x\rangle\,|0\rangle$
3. Compute f in superposition: $\frac{1}{\sqrt{|G|}} \sum_{x \in G} |x\rangle\,|f(x)\rangle$
4. Measure second register: $\frac{1}{\sqrt{|H|}} \sum_{x \in cH} |x\rangle\,|f(c)\rangle$
5. Compute DFT_G on first register: $$\frac{1}{\sqrt{|G||H|}} \sum_{\rho,i,j} \sqrt{d_\rho} \sum_{h \in H} \rho_{ij}(ch)\,|\rho, i, j\rangle$$
6. Measure first register: Sample (ρ, j) with probability $\sum_i \frac{d_\rho}{|G|} \left| \sum_{h \in H} \rho_{ij}(ch) \right|^2$.

Further classical post-processing necessary!

Searching for a satisfying assignment

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, find an $x \in \{0,1\}^n$ such that $f(x) = 1$. Such an x is also called "satisfying assignment" and f itself is also called "predicate". Note that this search problem includes NP-complete problems such as 3-SAT.

How the search problem is specified

Given: List X of $N = 2^n$ items and a predicate f on X which is given by the operator

$$V_f : |x\rangle \mapsto (-1)^{f(x)} |x\rangle .$$

Task: Find satisfying element x, i. e., $f(x) = 1$ (we assume precisely one such element exists).

Searching for a satisfying assignment

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, find an $x \in \{0,1\}^n$ such that $f(x) = 1$. Such an x is also called "satisfying assignment" and f itself is also called "predicate". Note that this search problem includes NP-complete problems such as 3-SAT.

How the search problem is specified

Given: List X of $N = 2^n$ items and a predicate f on X which is given by the operator

$$V_f : |x\rangle \mapsto (-1)^{f(x)} |x\rangle .$$

Task: Find satisfying element x, i. e., $f(x) = 1$ (we assume precisely one such element exists).

Computing a Function into the Phase

Another way of computing f

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the following operation computes f into the phases:

$$V_f \left|x\right\rangle = (-1)^{f(x)} \left|x\right\rangle$$

Question: What is the relation between U_f and V_f?

Realizing V_f from U_f

$$\begin{aligned} U_f \left|x\right\rangle \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) &= \frac{1}{\sqrt{2}} U_f \left|x\right\rangle \left|0\right\rangle - \frac{1}{\sqrt{2}} U_f \left|x\right\rangle \left|1\right\rangle \\ & \frac{1}{\sqrt{2}} \left|x\right\rangle \left|f(x)\right\rangle - \frac{1}{\sqrt{2}} \left|x\right\rangle \left|1 \oplus f(x)\right\rangle \\ & \frac{1}{\sqrt{2}} (-1)^{f(x)} \left|x\right\rangle \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) \end{aligned}$$

Computing a Function into the Phase

Another way of computing f

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the following operation computes f into the phases:

$$V_f \left|x\right\rangle = (-1)^{f(x)} \left|x\right\rangle$$

Question: What is the relation between U_f and V_f?

Realizing V_f from U_f

$$\begin{aligned} U_f \left|x\right\rangle \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) &= \frac{1}{\sqrt{2}} U_f \left|x\right\rangle \left|0\right\rangle - \frac{1}{\sqrt{2}} U_f \left|x\right\rangle \left|1\right\rangle \\ & \frac{1}{\sqrt{2}} \left|x\right\rangle \left|f(x)\right\rangle - \frac{1}{\sqrt{2}} \left|x\right\rangle \left|1 \oplus f(x)\right\rangle \\ & \frac{1}{\sqrt{2}} (-1)^{f(x)} \left|x\right\rangle \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle) \end{aligned}$$

Grover's Algorithm

The diffusion operator

$$D_n := \begin{pmatrix} -1 + \frac{2}{2^n} & \frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \frac{2}{2^n} & -1 + \frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{2^n} & \frac{2}{2^n} & \cdots & -1 + \frac{2}{2^n} \end{pmatrix}$$

Inversion about the average

One application of the operator $-D_n V_f$ when applied to the equal superposition of basis states does the following:

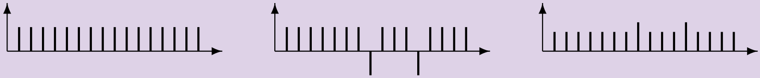
Grover's Algorithm

The diffusion operator

$$D_n := \begin{pmatrix} -1+\frac{2}{2^n} & \frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \frac{2}{2^n} & -1+\frac{2}{2^n} & \cdots & \frac{2}{2^n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{2^n} & \frac{2}{2^n} & \cdots & -1+\frac{2}{2^n} \end{pmatrix}$$

Inversion about the average

One application of the operator $-D_nV_f$ when applied to the equal superposition of basis states does the following:

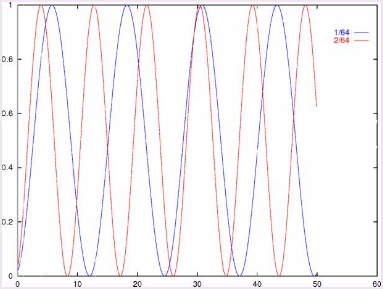


Grover's Algorithm

Grover's algorithm

First prepare the equal superposition. Then iterate the operator $-D_n S_f$ a number of $O(\sqrt{2^n})$ times. Afterwards measure the system in the computational basis. With high probability the result will be the solution x for which $f(x) = 1$.

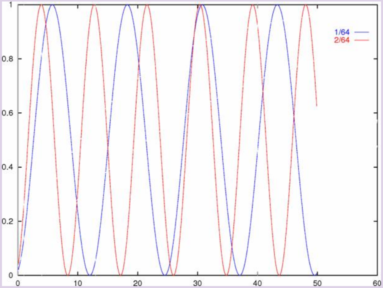
Success probability after several iterations



Grover's algorithm

First prepare the equal superposition. Then iterate the operator $-D_n S_f$ a number of $O(\sqrt{2^n})$ times. Afterwards measure the system in the computational basis. With high probability the result will be the solution x for which $f(x) = 1$.

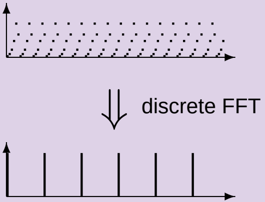
Success probability after several iterations



Two Different Types of Quantum Algorithms

Factoring

classical: $O(e^{(c+o(1))\sqrt[3]{\log n(\log\log n)^2}})$
quantum: $O(\mathrm{poly}(\log n))$



- feature extraction using signal transforms
- leads to the idea of "hidden subgroup problems"
- highly regular, in general huge speed-ups can be expected

Searching

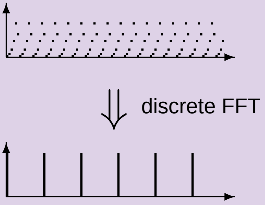
classical: $O(N)$
quantum: $O(\sqrt{N})$ (and this is optimal)

- increase the amplitude of target states via correlations
- in general a square-root speed-up can be expected

Two Different Types of Quantum Algorithms

Factoring

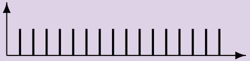
classical: $O(e^{(c+o(1))\sqrt[3]{\log n(\log\log n)^2}})$
quantum: $O(\mathrm{poly}(\log n))$

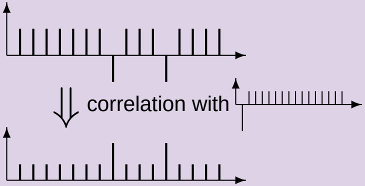


- feature extraction using signal transforms
- leads to the idea of "hidden subgroup problems"
- highly regular, in general huge speed-ups can be expected

Searching

classical: $O(N)$
quantum: $O(\sqrt{N})$ (and this is optimal)

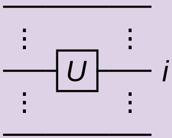


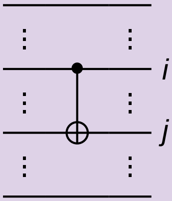


- increase the amplitude of target states via correlations
- in general a square-root speed-up can be expected

Quantum Gates and Circuits

Elementary quantum gates

$U^{(i)} =$ 

$\mathrm{CNOT}^{(i,j)} =$ 

Universal set of gates

Theorem (Barenco et al., 1995):

$$\mathcal{U}(2^n) = \langle U^{(i)}, \mathrm{CNOT}^{(i,j)} \quad : \quad i,j = 1,\ldots,n, \quad i \neq j \rangle$$

How to prove this result?

Next: breaking down the proof into several small steps

Examples for quantum circuits

Realizing a cyclic shift

How to realize $P_n : x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1

Purely classical realization

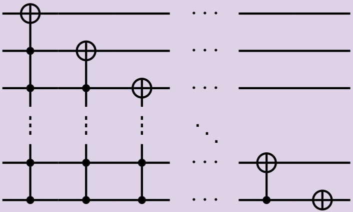
Solution 2

Genuinely quantum realization

Realizing a cyclic shift

How to realize $P_n : x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1



Purely classical realization

Solution 2

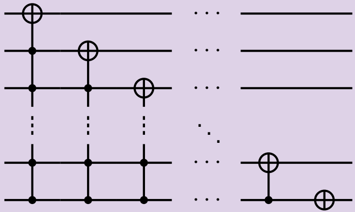
Genuinely quantum realization

Examples for quantum circuits

Realizing a cyclic shift

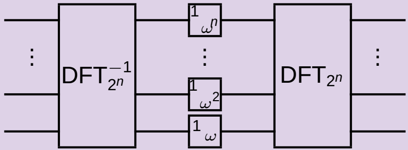
How to realize $P_n : x \mapsto x + 1 \mod 2^n$, which cyclically shifts the basis states of an n qubit register?

Solution 1



Purely classical realization

Solution 2



Genuinely quantum realization

Universality of CNOT and Local Gates

Proof outline

- Given a unitary matrix $U \in \mathcal{U}(2^n)$.
- Write $U = U_1 \cdot \ldots \cdot U_M$, where U_i acts on pairs of states
- Factorize each U_i using multiply-controlled $\Lambda_n(V)$ gates, where $V \in \mathcal{U}(2)$.
- Write each V in the form $(A^\dagger \sigma_x A)(B^\dagger \sigma_x B)$ with $A, B \in \mathcal{U}(2)$.
- Use this to write each $\Lambda_n(V)$ in terms of local gates and $\Lambda_n(\sigma_x)$ (i. e., generalized CNOTs).
- Implement $\Lambda_n(\sigma_x)$ recursively using $\Lambda_k(\sigma_x)$, where $k < n$.

Literature

A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo et al.
Elementary gates for quantum computation.
Physical Review A, 52(5):3457–3467, 1995.

Universality of CNOT and Local Gates

Proof outline

- Given a unitary matrix $U \in \mathcal{U}(2^n)$.
- Write $U = U_1 \cdot \ldots \cdot U_M$, where U_i acts on pairs of states
- Factorize each U_i using multiply-controlled $\Lambda_n(V)$ gates, where $V \in \mathcal{U}(2)$.
- Write each V in the form $(A^\dagger \sigma_x A)(B^\dagger \sigma_x B)$ with $A, B \in \mathcal{U}(2)$.
- Use this to write each $\Lambda_n(V)$ in terms of local gates and $\Lambda_n(\sigma_x)$ (i. e., generalized CNOTs).
- Implement $\Lambda_n(\sigma_x)$ recursively using $\Lambda_k(\sigma_x)$, where $k < n$.

Literature

A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo et al.
Elementary gates for quantum computation.
Physical Review A, 52(5):3457–3467, 1995.

Rotation on a subspace spanned by the states s_1 and s_2

$$T(s_1,s_2) = \begin{array}{l} \begin{array}{ccccccccccc} & & & s_1 & & & & s_2 & & & \end{array} \\ \left(\begin{array}{ccccccccccc} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & * & & & & * & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & * & & & & * & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{array}\right) \begin{array}{c} \\ \\ \\ s_1 \\ \\ \\ \\ s_2 \\ \\ \\ \\ \end{array} \end{array}$$

Theorem

Every $U \in \mathcal{U}(2^n)$ can be written in the form

$$U = \prod_{s_1,s_2 \in \{0,1\}^n} T(s_1,s_2).$$

Operations on Subspaces

Rotation on a subspace spanned by the states s_1 and s_2

$$T(s_1,s_2) \quad = \quad \begin{array}{cl} \begin{array}{ccccccccccc} & & & s_1 & & & & s_2 & & & \\ & & & \downarrow & & & & \downarrow & & & \end{array} & \\ \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & * & & & & * & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & * & & & & * & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix} & \begin{array}{l} \\ \\ \\ \leftarrow s_1 \\ \\ \\ \\ \leftarrow s_2 \\ \\ \\ \\ \end{array} \end{array}$$

Theorem

Every $U \in \mathcal{U}(2^n)$ can be written in the form

$$U = \prod_{s_1,s_2 \in \{0,1\}^n} T(s_1, s_2).$$

Conditional gates with multiple controls

Let $U \in \mathcal{U}(2)$. Then $\Lambda_k(U) \in \mathcal{U}(2^{k+1})$ is defined by

$$\Lambda_k := \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \boxed{U} \end{pmatrix} = \mathbf{1}_{2^{k+1}-2} \oplus U.$$

Alternative description of $\Lambda_k(U)$

$$\Lambda_k(U)\,|x_1,\ldots,x_n\rangle\,|y\rangle = \begin{cases} |x_1,\ldots,x_n\rangle\,|y\rangle & \text{if } \exists i : x_i \neq 1 \\ |x_1,\ldots,x_n\rangle\,U\,|y\rangle & \text{if } \forall i : x_i = 1 \end{cases}$$

Elementary Quantum Gates

Conditional gates with multiple controls

Let $U \in \mathcal{U}(2)$. Then $\Lambda_k(U) \in \mathcal{U}(2^{k+1})$ is defined by

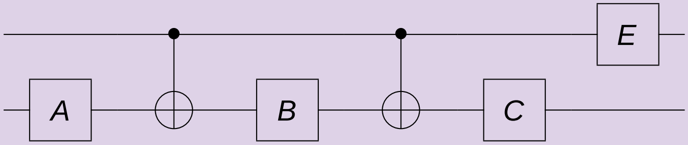
$$\Lambda_k := \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \boxed{U} \end{pmatrix} = \mathbf{1}_{2^{k+1}-2} \oplus U.$$

Alternative description of $\Lambda_k(U)$

$$\Lambda_k(U)\,|x_1,\ldots,x_n\rangle\,|y\rangle = \begin{cases} |x_1,\ldots,x_n\rangle\,|y\rangle & \text{if } \exists i : x_i \neq 1 \\ |x_1,\ldots,x_n\rangle\,U\,|y\rangle & \text{if } \forall i : x_i = 1 \end{cases}$$

Realizing a singly controlled $\Lambda_1(U)$ gate

$\Lambda_1(U) =$

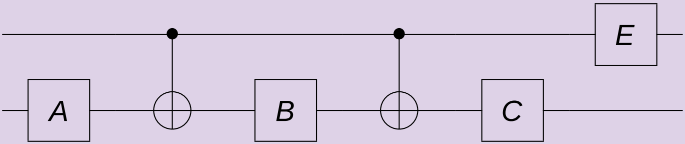


Description of the local gates used in this circuit

- $U = \exp(i\phi)W$,
- $W \in \mathrm{SU}(2)$,
- $E = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\phi) \end{pmatrix}$,
- $ABC = I$, $A\sigma_x B\sigma_x C = W$.
- The matrices A,B, and C are obtained from the decomposition $U = (A^\dagger\sigma_x A)(B^\dagger\sigma_x B)$.

Elementary Quantum Gates

Realizing a singly controlled $\Lambda_1(U)$ gate

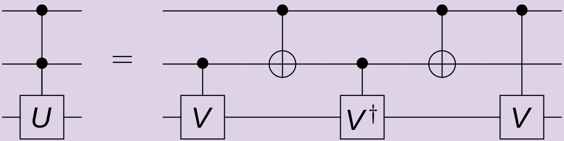
$\Lambda_1(U) =$ 

Description of the local gates used in this circuit

- $U = \exp(i\phi)W$,
- $W \in \mathrm{SU}(2)$,
- $E = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\phi) \end{pmatrix}$,
- $ABC = I$, $A\sigma_x B\sigma_x C = W$.
- The matrices A,B, and C are obtained from the decomposition $U = (A^\dagger\sigma_x A)(B^\dagger\sigma_x B)$.

Elementary Quantum Gates

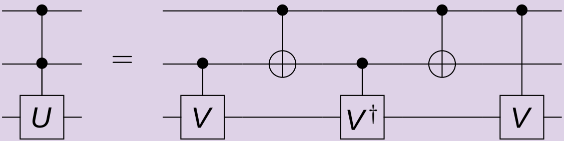
Realizing a two-fold controlled gate $\Lambda_2(U)$



Comments

- Here we have to find a unitary V with $V^2 = U$.
- This idea can be generalized to arbitrary $\Lambda_k(U)$, however, the complexity obtained from this factorization scales exponentially with k.
- Need better break-down strategy.

Realizing a two-fold controlled gate $\Lambda_2(U)$

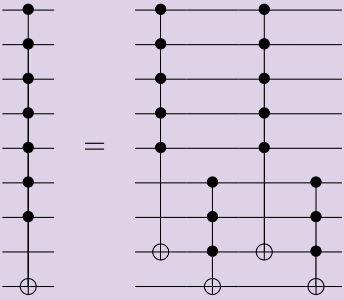


Comments

- Here we have to find a unitary V with $V^2 = U$.
- This idea can be generalized to arbitrary $\Lambda_k(U)$, however, the complexity obtained from this factorization scales exponentially with k.
- Need better break-down strategy.

Efficient Break-Down Strategies

$\Lambda_{n-2}(\sigma_x)$



$\Longrightarrow$ linear in n

$\Lambda_{n-1}(W)$ with $W \in \mathrm{SU}(2)$

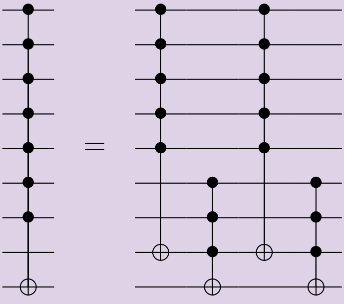
$\Longrightarrow$ linear in n

Important result about multiply-controlled gates

Let $U \in \mathcal{U}(2)$. Then any $\Lambda_{n-1}(U)$ gate operating on n qubits can be implemented using at most $O(n)$ elementary gates.

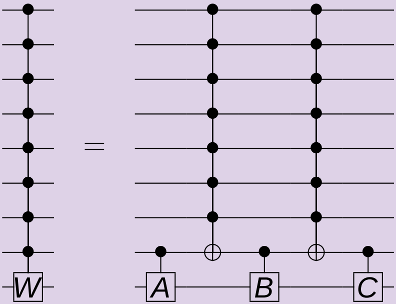
Efficient Break-Down Strategies

$\Lambda_{n-2}(\sigma_x)$



$\Longrightarrow$ linear in n

$\Lambda_{n-1}(W)$ with $W \in \mathrm{SU}(2)$



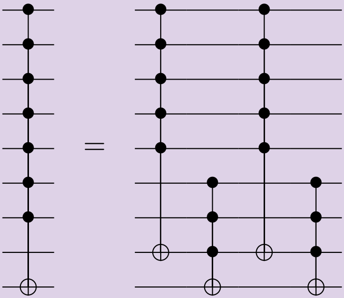
$\Longrightarrow$ linear in n

Important result about multiply-controlled gates

Let $U \in \mathcal{U}(2)$. Then any $\Lambda_{n-1}(U)$ gate operating on n qubits can be implemented using at most $O(n)$ elementary gates.

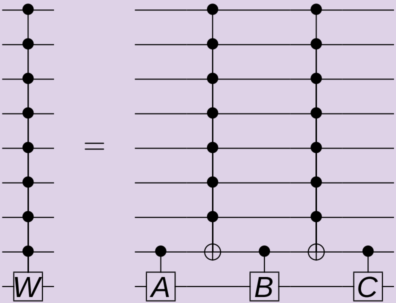
Efficient Break-Down Strategies

$\Lambda_{n-2}(\sigma_x)$



$\Longrightarrow$ linear in n

$\Lambda_{n-1}(W)$ with $W \in \mathrm{SU}(2)$



$\Longrightarrow$ linear in n

Important result about multiply-controlled gates

Let $U \in \mathcal{U}(2)$. Then any $\Lambda_{n-1}(U)$ gate operating on n qubits can be implemented using at most $O(n)$ elementary gates.

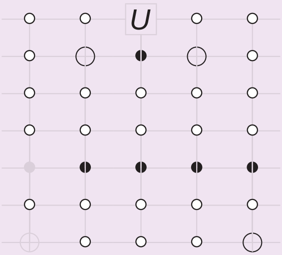
Operations on a two-dimensional subspace

Consider a subspace with basis $|i\rangle$, $|j\rangle$:

- $\Lambda_{n-1}(U)$, if i and j differ in only one bit
- Use Gray code sequence to connect i and j.

Example: $n = 7$, $i = 5$, $j = 100$

0000101
0000100
0100100
1100100



$\Longrightarrow$ Complexity is $O(n^3)$ elementary gates.

Elementary Quantum Gates

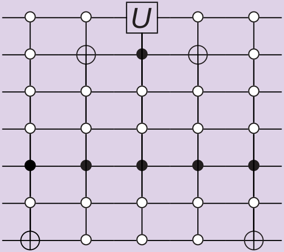
Operations on a two-dimensional subspace

Consider a subspace with basis $|i\rangle$, $|j\rangle$:

- $\Lambda_{n-1}(U)$, if i and j differ in only one bit
- Use Gray code sequence to connect i and j.

Example: $n = 7$, $i = 5$, $j = 100$

0000101
0000100
0100100
1100100



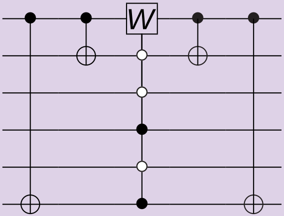
$\Longrightarrow$ Complexity is $O(n^3)$ elementary gates.

Saving Even More Gates and the Final Result

The Cybenko trick (2001)

We can save almost all of the permutation gates necessary for the Gray code by using CNOT gates at the different positions:

0000101
1000101
1100101
1100100



$\Longrightarrow$ Complexity of $O(n)$ gates for $W \in \mathrm{SU}(n)$

Theorem

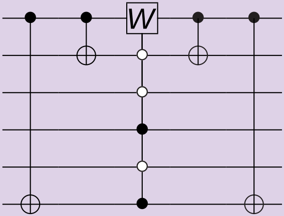
Any unitary transformation on n qubits can be implemented using at most 4^n elementary gates. This bound is tight and for almost all elements of $\mathcal{U}(2^n)$ we need $\Theta(4^n)$ gates.

Saving Even More Gates and the Final Result

The Cybenko trick (2001)

We can save almost all of the permutation gates necessary for the Gray code by using CNOT gates at the different positions:

$$\left.\begin{array}{l}0000101\\1000101\end{array}\right\}$$
$$\begin{array}{l}1100101\\1100100\end{array}$$



$\Longrightarrow$ Complexity of $O(n)$ gates for $W \in \mathrm{SU}(n)$

Theorem

Any unitary transformation on n qubits can be implemented using at most 4^n elementary gates. This bound is tight and for almost all elements of $\mathcal{U}(2^n)$ we need $\Theta(4^n)$ gates.

Conclusions

- Shor's algorithm for factoring integers
  - Reducing factoring to order finding
  - Setting up a periodic state
  - Efficient extraction of the period by computing a QFT
- Grover's algorithm: searching N items in time $O(\sqrt{N})$.
- Elementary quantum gates:
  - Controlled NOT gate
  - Local unitary transformations

Dedicated to Thomas Beth, *16.11.1949, † 17.08.2005