量子計算的數學基礎 MA5501*

Chapter 4. Simon's Algorithm

§4.1 Simon's Problem

Simon's algorithm was the first quantum algorithm to show an **exponential speed-up** versus the best classical algorithm in solving a specific problem. This inspired the quantum algorithms based on the quantum Fourier transform, which is used in the most famous quantum algorithm: Shor's factoring algorithm.

Let $N = 2^n$, and identify the set $\{0, \cdots, N-1\}$ with $\{0,1\}^n$. Let $j \oplus s$ be the n-bit string obtained by bitwise adding the n-bit strings j and s mod 2; that is,

$$j \oplus s = \big((j_1 \oplus s_1)(j_2 \oplus s_2) \cdots (j_n \oplus s_n)\big)_2$$

if

$$j = (j_1 j_2 \cdots j_n)_2 \quad \text{and} \quad s = (s_1 s_2 \cdots s_n)_2 .$$

§4.1 Simon's Problem

Simon's algorithm was the first quantum algorithm to show an **exponential speed-up** versus the best classical algorithm in solving a specific problem. This inspired the quantum algorithms based on the quantum Fourier transform, which is used in the most famous quantum algorithm: Shor's factoring algorithm.

Let $N = 2^n$, and identify the set $\{0, \cdots, N-1\}$ with $\{0,1\}^n$. Let $j \oplus s$ be the n-bit string obtained by bitwise adding the n-bit strings j and s mod 2; that is,

$$j \oplus s = \big((j_1 \oplus s_1)(j_2 \oplus s_2) \cdots (j_n \oplus s_n)\big)_2$$

if

$$j = (j_1 j_2 \cdots j_n)_2 \quad \text{and} \quad s = (s_1 s_2 \cdots s_n)_2 .$$

§4.1 Simon's Problem

Simon's problem:

- **Formulation 1**: For $N = 2^n$, we are given $x = (x_0, \cdots, x_{N-1})$, with $x_i \in \{0,1\}^n$, with the property that there is a unique but unknown nonzero $s \in \{0,1\}^n$ such that $x_i = x_j$ if and only if $(i = j \text{ or } i = j \oplus s)$. Find s.
- **Formulation 2**: If $f\colon \{0,1\}^n \to \{0,1\}^n$ is either an one-to-one or a two-to-one function satisfying the property that there exists $s \in \{0,1\}^n$ such that $f(i) = f(j)$ if and only if $i = j$ or $i = j \oplus s$. Determine the class to which f belongs to.

Note that the input $x = \{x_0, \cdots, x_{N-1}\}$ now has variables x_i that themselves are n-bit strings, and one query gives such a string completely $|i\rangle|0^n\rangle \mapsto |i\rangle|x_i\rangle$.

§4.1 Simon's Problem

Simon's problem:

- **Formulation 1**: For $N = 2^n$, we are given $x = (x_0, \cdots, x_{N-1})$, with $x_i \in \{0,1\}^n$, with the property that there is a unique but unknown nonzero $s \in \{0,1\}^n$ such that $x_i = x_j$ if and only if $(i = j$ or $i = j \oplus s)$. Find s.
- **Formulation 2**: If $f\colon \{0,1\}^n \to \{0,1\}^n$ is either an one-to-one or a two-to-one function satisfying the property that there exists $s \in \{0,1\}^n$ such that $f(i) = f(j)$ if and only if $i = j$ or $i = j \oplus s$. Determine the class to which f belongs to.

Note that the input $x = \{x_0, \cdots, x_{N-1}\}$ now has variables x_i that themselves are n-bit strings, and one query gives such a string completely $|i\rangle|0^n\rangle \mapsto |i\rangle|x_i\rangle$.

§4.1 Simon's Problem

Simon's problem:

- **Formulation 1**: For $N = 2^n$, we are given $x = (x_0, \cdots, x_{N-1})$, with $x_i \in \{0,1\}^n$, with the property that there is a unique but unknown nonzero $s \in \{0,1\}^n$ such that $x_i = x_j$ if and only if $(i = j \text{ or } i = j \oplus s)$. Find s.
- **Formulation 2**: If $f\colon \{0,1\}^n \to \{0,1\}^n$ is either an one-to-one or a two-to-one function satisfying the property that there exists $s \in \{0,1\}^n$ such that $f(i) = f(j)$ if and only if $i = j$ or $i = j \oplus s$. Determine the class to which f belongs to.

Note that the input $x = \{x_0, \cdots, x_{N-1}\}$ now has variables x_i that themselves are n-bit strings, and one query gives such a string completely $|i\rangle|0^n\rangle \mapsto |i\rangle|x_i\rangle$.

§4.2 The Quantum Algorithm

Simon's algorithm starts in a state of 2^n zero qubits $|0^n\rangle|0^n\rangle$ and apply Hadamard transforms to the first n qubits, giving

$$\frac{1}{\sqrt{2^n}} \sum_{i\in\{0,1\}^n} |i\rangle|0^n\rangle .$$

A query turns this into

$$\frac{1}{\sqrt{2^n}} \sum_{i\in\{0,1\}^n} |i\rangle|x_i\rangle .$$

Now the algorithm measures the second n-qubit register in the computational basis; this measurement is actually not necessary, but it facilitates analysis. The measurement outcome will be some value x_i and the first register will collapse to the superposition of the two indices having that x_i-value:

$$\frac{1}{\sqrt{2}}(|i\rangle + |i\oplus s\rangle)|x_i\rangle .$$

§4.2 The Quantum Algorithm

Simon's algorithm starts in a state of 2^n zero qubits $|0^n\rangle|0^n\rangle$ and apply Hadamard transforms to the first n qubits, giving

$$\frac{1}{\sqrt{2^n}} \sum_{i\in\{0,1\}^n} |i\rangle|0^n\rangle .$$

A query turns this into

$$\frac{1}{\sqrt{2^n}} \sum_{i\in\{0,1\}^n} |i\rangle|x_i\rangle .$$

Now the algorithm measures the second n-qubit register in the computational basis; this measurement is actually not necessary, but it facilitates analysis. The measurement outcome will be some value x_i and the first register will collapse to the superposition of the two indices having that x_i-value:

$$\frac{1}{\sqrt{2}}(|i\rangle + |i \oplus s\rangle)|x_i\rangle .$$

§4.2 The Quantum Algorithm

Simon's algorithm starts in a state of 2^n zero qubits $|0^n\rangle|0^n\rangle$ and apply Hadamard transforms to the first n qubits, giving

$$\frac{1}{\sqrt{2^n}}\sum_{i\in\{0,1\}^n}|i\rangle|0^n\rangle.$$

A query turns this into

$$\frac{1}{\sqrt{2^n}}\sum_{i\in\{0,1\}^n}|i\rangle|x_i\rangle.$$

Now the algorithm measures the second n-qubit register in the computational basis; this measurement is actually not necessary, but it facilitates analysis. The measurement outcome will be some value x_i and the first register will collapse to the superposition of the two indices having that x_i-value:

$$\frac{1}{\sqrt{2}}(|i\rangle+|i\oplus s\rangle)|x_i\rangle.$$

§4.2 The Quantum Algorithm

We will now ignore the second register and apply Hadamard transforms to the first n qubits. Using

$$\mathrm{H}^{\otimes n}|i\rangle = \frac{1}{\sqrt{2^n}} \sum_{j\in\{0,1\}^n} (-1)^{i\bullet j}|j\rangle,$$

and the fact that $(i \oplus s)\bullet j = (i\bullet j) \oplus (s\bullet j)$ (which is a direct consequence of $(i_k \oplus s_k)\cdot j_k = (i_k \cdot j_k)\oplus(s_k \cdot j_k)$ for all $i_k, s_k, j_k \in \{0,1\}$), we can write the resulting state as

$$\begin{aligned}
&\mathrm{H}^{\otimes n}\Big(\frac{1}{\sqrt{2}}\big(|i\rangle + |i\oplus s\rangle\big)\Big)\\
&\quad = \frac{1}{\sqrt{2^{n+1}}}\Bigg(\sum_{j\in\{0,1\}^n} (-1)^{i\bullet j}|j\rangle + \sum_{j\in\{0,1\}^n} (-1)^{(i\oplus s)\bullet j}|j\rangle\Bigg)\\
&\quad = \frac{1}{\sqrt{2^{n+1}}} \sum_{j\in\{0,1\}^n} (-1)^{i\bullet j}\big(1 + (-1)^{s\bullet j}\big)|j\rangle.
\end{aligned}$$

§4.2 The Quantum Algorithm

Note that $|j\rangle$ has nonzero amplitude if $s \bullet j = 0 \mod 2$. Measuring the state gives a uniformly random element from the set $\{j \,|\, s \bullet j = 0 \mod 2\}$. Accordingly, we get a linear equation

$$s \bullet j = 0 \mod 2$$

that gives information about s. We repeat this algorithm until we have obtained $n-1$ **independent** linear equations involving s

$$\begin{bmatrix} j_{n-1}^{(1)} & j_{n-2}^{(1)} & \cdots & j_0^{(1)} \\ j_{n-1}^{(2)} & j_{n-2}^{(2)} & \cdots & j_0^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ j_{n-1}^{(n-1)} & j_{n-2}^{(n-1)} & \cdots & j_0^{(n-1)} \end{bmatrix} \begin{bmatrix} s_{n-1} \\ s_{n-2} \\ \vdots \\ s_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mod 2\,.$$

The solutions to these equations will be 0^n and the correct s, which we can compute efficiently by a classical algorithm (Gaussian elimination modulo 2). This can be done by means of a classical circuit of size roughly $\mathcal{O}(n^3)$.

§4.2 The Quantum Algorithm

Note that $|j\rangle$ has nonzero amplitude if $s \bullet j = 0 \mod 2$. Measuring the state gives a uniformly random element from the set $\{j \,|\, s \bullet j = 0 \mod 2\}$. Accordingly, we get a linear equation

$$s \bullet j = 0 \mod 2$$

that gives information about s. We repeat this algorithm until we have obtained $n-1$ **independent** linear equations involving s

$$\begin{bmatrix} j_{n-1}^{(1)} & j_{n-2}^{(1)} & \cdots & j_0^{(1)} \\ j_{n-1}^{(2)} & j_{n-2}^{(2)} & \cdots & j_0^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ j_{n-1}^{(n-1)} & j_{n-2}^{(n-1)} & \cdots & j_0^{(n-1)} \end{bmatrix} \begin{bmatrix} s_{n-1} \\ s_{n-2} \\ \vdots \\ s_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mod 2\,.$$

The solutions to these equations will be 0^n and the correct s, which we can compute efficiently by a classical algorithm (Gaussian elimination modulo 2). This can be done by means of a classical circuit of size roughly $\mathcal{O}(n^3)$.

§4.2 The Quantum Algorithm

Note that $|j\rangle$ has nonzero amplitude if $s \bullet j = 0 \bmod 2$. Measuring the state gives a uniformly random element from the set $\{j \,|\, s \bullet j = 0 \bmod 2\}$. Accordingly, we get a linear equation

$$s \bullet j = 0 \bmod 2$$

that gives information about s. We repeat this algorithm until we have obtained $n-1$ **independent** linear equations involving s

$$\begin{bmatrix} j_{n-1}^{(1)} & j_{n-2}^{(1)} & \cdots & j_0^{(1)} \\ j_{n-1}^{(2)} & j_{n-2}^{(2)} & \cdots & j_0^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ j_{n-1}^{(n-1)} & j_{n-2}^{(n-1)} & \cdots & j_0^{(n-1)} \end{bmatrix} \begin{bmatrix} s_{n-1} \\ s_{n-2} \\ \vdots \\ s_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mod 2\,.$$

The solutions to these equations will be 0^n and the correct s, which we can compute efficiently by a classical algorithm (Gaussian elimination modulo 2). This can be done by means of a classical circuit of size roughly $\mathcal{O}(n^3)$.

§4.2 The Quantum Algorithm

Note that $|j\rangle$ has nonzero amplitude if $s \bullet j = 0 \mod 2$. Measuring the state gives a uniformly random element from the set $\{j \,|\, s \bullet j = 0 \mod 2\}$. Accordingly, we get a linear equation

$$s \bullet j = 0 \mod 2$$

that gives information about s. We repeat this algorithm until we have obtained $n-1$ **independent** linear equations involving s

$$\begin{bmatrix} j^{(1)}_{n-1} & j^{(1)}_{n-2} & \cdots & j^{(1)}_{0} \\ j^{(2)}_{n-1} & j^{(2)}_{n-2} & \cdots & j^{(2)}_{0} \\ \vdots & \vdots & \ddots & \vdots \\ j^{(n-1)}_{n-1} & j^{(n-1)}_{n-2} & \cdots & j^{(n-1)}_{0} \end{bmatrix} \begin{bmatrix} s_{n-1} \\ s_{n-2} \\ \vdots \\ s_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mod 2\,.$$

The solutions to these equations will be 0^n and the correct s, which we can compute efficiently by a classical algorithm (Gaussian elimination modulo 2). This can be done by means of a classical circuit of size roughly $\mathcal{O}(n^3)$.

§4.2 The Quantum Algorithm

Note that if the j's you have generated at some point span a space of size 2^k, for some $k < n-1$, then the probability that your next run of the algorithm produces a j that is linearly independent of the earlier ones, is $(2^n - 2^k)/2^n \geqslant 1/2$. Hence an expected number of $\mathcal{O}(n)$ runs of the algorithm suffices to find $n-1$ linearly independent j's. Simon's algorithm thus finds s using an expected number of $\mathcal{O}(n)$ x_i-queries and polynomially many other operations.

Figure 1: Quantum circuit for Simon's algorithm

§4.2 The Quantum Algorithm

Note that if the j's you have generated at some point span a space of size 2^k, for some $k < n-1$, then the probability that your next run of the algorithm produces a j that is linearly independent of the earlier ones, is $(2^n - 2^k)/2^n \geqslant 1/2$. Hence an expected number of $\mathcal{O}(n)$ runs of the algorithm suffices to find $n-1$ linearly independent j's. Simon's algorithm thus finds s using an expected number of $\mathcal{O}(n)$ x_i-queries and polynomially many other operations.

Figure 1: Quantum circuit for Simon's algorithm

§4.2 The Quantum Algorithm

Note that if the j's you have generated at some point span a space of size 2^k, for some $k < n-1$, then the probability that your next run of the algorithm produces a j that is linearly independent of the earlier ones, is $(2^n - 2^k)/2^n \geqslant 1/2$. Hence an expected number of $\mathcal{O}(n)$ runs of the algorithm suffices to find $n-1$ linearly independent j's.

Simon's algorithm thus finds s using an expected number of $\mathcal{O}(n)$ x_i-queries and polynomially many other operations.

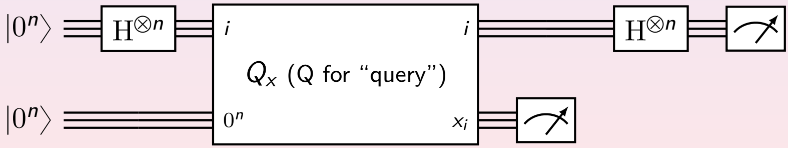

Figure 1: Quantum circuit for Simon's algorithm

§4.2 The Quantum Algorithm

Example

Let f be a periodic function of 2 qubits given by

$$f(x_1, x_2) = (x_1 \oplus x_2, x_1 \oplus x_2) \qquad \forall\, x_1, x_2 \in \{0, 1\}\,.$$

The quantum circuit to solve the problem is:

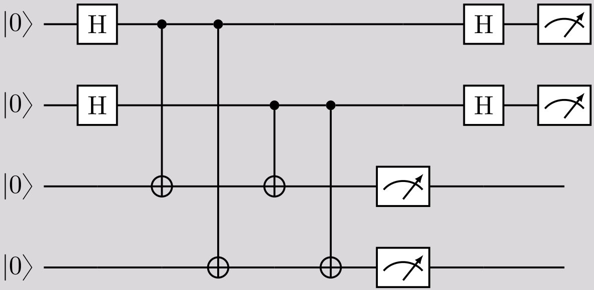

Figure 2: Quantum circuit for Simon's algorithm in this example

§4.2 The Quantum Algorithm

Example (Cont'd)

To check the four CNOT operations indeed provide the oracle U_f, we note that by writing $|x\rangle = |x_1x_2\rangle$ and $|y\rangle = |y_1\rangle|y_2\rangle$, we have

$$\begin{aligned}
&\mathbf{CNOT}_{2,4}\mathbf{CNOT}_{2,3}\mathbf{CNOT}_{1,4}\mathbf{CNOT}_{1,3}|x\rangle|y\rangle \\
&\quad= \mathbf{CNOT}_{2,4}\mathbf{CNOT}_{2,3}\mathbf{CNOT}_{1,4}\mathbf{CNOT}_{1,3}|x_1\rangle|x_2\rangle|y_1\rangle|y_2\rangle \\
&\quad= \mathbf{CNOT}_{2,4}\mathbf{CNOT}_{2,3}\mathbf{CNOT}_{1,4}|x_1\rangle|x_2\rangle|x_1 \oplus y_1\rangle|y_2\rangle \\
&\quad= \mathbf{CNOT}_{2,4}\mathbf{CNOT}_{2,3}|x_1\rangle|x_2\rangle|x_1 \oplus y_1\rangle|x_1 \oplus y_2\rangle \\
&\quad= \mathbf{CNOT}_{2,4}|x_1\rangle|x_2\rangle|x_1 \oplus x_2 \oplus y_1\rangle|x_1 \oplus y_2\rangle \\
&\quad= |x_1\rangle|x_2\rangle|x_1 \oplus x_2 \oplus y_1\rangle|x_1 \oplus x_2 \oplus y_2\rangle = |x\rangle|y \oplus f(x)\rangle \\
&\quad= Q_f|x\rangle|y\rangle\,.
\end{aligned}$$

§4.3 Classical Algorithms for Simon's Problem

§4.3.1 Upper bound

Let us first sketch a classical randomized algorithm that solves Simon's problem using $\mathcal{O}(\sqrt{2^n})$ queries. Our algorithm will make T randomly chosen distinct queries $i_1, \cdots, i_T$, for some T to be determined later. If there is a collision among those queries (that is, $x_{i_k} = x_{i_\ell}$ for some $k \neq \ell$), then we are done, because then we know $i_k = i_\ell \bmod s$, equivalently $s = i_k \oplus i_\ell$. How large should T be such that we are likely to see a collision in case $s \neq 0^n$ (there will not be any collisions if $s = 0^n$)? There are $C_2^T = \dfrac{T(T-1)}{2} \approx T^2/2$ pairs in our sequence that could be a collision, and the probability for a fixed pair to form a collision is $1/2^{n-1}$; thus the expected number of collisions in our sequence will be roughly $T^2/2^n$. If we choose $T = \sqrt{2^n}$, we expect to have roughly 1 collision in our sequence, which is good enough to find s.

§4.3 Classical Algorithms for Simon's Problem

§4.3.1 Upper bound

Let us first sketch a classical randomized algorithm that solves Simon's problem using $\mathcal{O}(\sqrt{2^n})$ queries. Our algorithm will make T randomly chosen distinct queries $i_1, \cdots, i_T$, for some T to be determined later. If there is a collision among those queries (that is, $x_{i_k} = x_{i_\ell}$ for some $k \neq \ell$), then we are done, because then we know $i_k = i_\ell \bmod s$, equivalently $s = i_k \oplus i_\ell$. How large should T be such that we are likely to see a collision in case $s \neq 0^n$ (there will not be any collisions if $s = 0^n$)? There are $C_2^T = \dfrac{T(T-1)}{2} \approx T^2/2$ pairs in our sequence that could be a collision, and the probability for a fixed pair to form a collision is $1/2^{n-1}$; thus the expected number of collisions in our sequence will be roughly $T^2/2^n$. If we choose $T = \sqrt{2^n}$, we expect to have roughly 1 collision in our sequence, which is good enough to find s.

§4.3 Classical Algorithms for Simon's Problem

§4.3.1 Upper bound

Let us first sketch a classical randomized algorithm that solves Simon's problem using $\mathcal{O}(\sqrt{2^n})$ queries. Our algorithm will make T randomly chosen distinct queries $i_1, \cdots, i_T$, for some T to be determined later. If there is a collision among those queries (that is, $x_{i_k} = x_{i_\ell}$ for some $k \neq \ell$), then we are done, because then we know $i_k = i_\ell \bmod s$, equivalently $s = i_k \oplus i_\ell$. How large should T be such that we are likely to see a collision in case $s \neq 0^n$ (there will not be any collisions if $s = 0^n$)? There are $C_2^T = \dfrac{T(T-1)}{2} \approx T^2/2$ pairs in our sequence that could be a collision, and the probability for a fixed pair to form a collision is $1/2^{n-1}$; thus the expected number of collisions in our sequence will be roughly $T^2/2^n$. If we choose $T = \sqrt{2^n}$, we expect to have roughly 1 collision in our sequence, which is good enough to find s.

§4.3 Classical Algorithms for Simon's Problem

Of course, an expected value of 1 collision does not mean that we will have at least one collision with high probability, but a slightly more involved calculation shows the latter statement as well.

§4.3 Classical Algorithms for Simon's Problem

§4.3.2 Lower bound

Simon proved that any classical randomized algorithm that finds s with high probability needs to make $\Omega(\sqrt{2^n})$ queries, so the above classical algorithm is essentially optimal. This was the first proven exponential separation between quantum algorithms and classical bounded-error algorithms (let us stress again that this does not prove an exponential separation in the usual circuit model, because **we are counting queries rather than ordinary operations** here). Simon's algorithm inspired Shor to his factoring algorithm.

We prove a lower bound for the decision version of Simon's problem:

Given: input $x = (x_0, \cdots, x_{N-1})$, where $N = 2^n$ and $x_i \in \{0,1\}^n$.

Promise: there exists $s \in \{0,1\}^n$ such that $x_i = x_j$ if and only if $(i = j \text{ or } i = j \oplus s)$.

Task: decide whether $s = 0^n$.

§4.3 Classical Algorithms for Simon's Problem

§4.3.2 Lower bound

Simon proved that any classical randomized algorithm that finds s with high probability needs to make $\Omega(\sqrt{2^n})$ queries, so the above classical algorithm is essentially optimal. This was the first proven exponential separation between quantum algorithms and classical bounded-error algorithms (let us stress again that this does not prove an exponential separation in the usual circuit model, because **we are counting queries rather than ordinary operations** here). Simon's algorithm inspired Shor to his factoring algorithm.

We prove a lower bound for the decision version of Simon's problem:

Given: input $x = (x_0, \cdots, x_{N-1})$, where $N = 2^n$ and $x_i \in \{0,1\}^n$.

Promise: there exists $s \in \{0,1\}^n$ such that $x_i = x_j$ if and only if $(i = j$ or $i = j \oplus s)$.

Task: decide whether $s = 0^n$.

§4.3 Classical Algorithms for Simon's Problem

§4.3.2 Lower bound

Simon proved that any classical randomized algorithm that finds s with high probability needs to make $\Omega(\sqrt{2^n})$ queries, so the above classical algorithm is essentially optimal. This was the first proven exponential separation between quantum algorithms and classical bounded-error algorithms (let us stress again that this does not prove an exponential separation in the usual circuit model, because **we are counting queries rather than ordinary operations** here). Simon's algorithm inspired Shor to his factoring algorithm.

We prove a lower bound for the decision version of Simon's problem:

Given: input $x = (x_0, \cdots, x_{N-1})$, where $N = 2^n$ and $x_i \in \{0,1\}^n$.

Promise: there exists $s \in \{0,1\}^n$ such that $x_i = x_j$ if and only if $(i = j \text{ or } i = j \oplus s)$.

Task: decide whether $s = 0^n$.

§4.3 Classical Algorithms for Simon's Problem

Consider the input distribution μ that is defined as follows. With probability $1/2$, x is a uniformly random permutation of $\{0,1\}^n$; this corresponds to the case $s = 0^n$. With probability $1/2$, we pick a nonzero string s at random, and for each pair $(i; i \oplus s)$, we pick a unique value for $x_i = x_{i\oplus s}$ at random. If there exists a randomized T-query algorithm that achieves success probability $\geqslant 2/3$ under this input distribution μ, then there also is deterministic T-query algorithm that achieves success probability $\geqslant 2/3$ under μ (because the behavior of the randomized algorithm is an average over a number of deterministic algorithms). Now consider a deterministic algorithm with error $\leqslant 1/3$ under μ, that makes T queries to x. We want to show that $T = \Omega(\sqrt{2^n})$.

§4.3 Classical Algorithms for Simon's Problem

Consider the input distribution μ that is defined as follows. With probability $1/2$, x is a uniformly random permutation of $\{0,1\}^n$; this corresponds to the case $s = 0^n$. With probability $1/2$, we pick a nonzero string s at random, and for each pair $(i; i \oplus s)$, we pick a unique value for $x_i = x_{i \oplus s}$ at random. If there exists a randomized T-query algorithm that achieves success probability $\geqslant 2/3$ under this input distribution μ, then there also is deterministic T-query algorithm that achieves success probability $\geqslant 2/3$ under μ (because the behavior of the randomized algorithm is an average over a number of deterministic algorithms). Now consider a deterministic algorithm with error $\leqslant 1/3$ under μ, that makes T queries to x. We want to show that $T = \Omega(\sqrt{2^n})$.

§4.3 Classical Algorithms for Simon's Problem

Consider the input distribution μ that is defined as follows. With probability $1/2$, x is a uniformly random permutation of $\{0,1\}^n$; this corresponds to the case $s = 0^n$. With probability $1/2$, we pick a nonzero string s at random, and for each pair $(i; i \oplus s)$, we pick a unique value for $x_i = x_{i \oplus s}$ at random. If there exists a randomized T-query algorithm that achieves success probability $\geqslant 2/3$ under this input distribution μ, then there also is deterministic T-query algorithm that achieves success probability $\geqslant 2/3$ under μ (because the behavior of the randomized algorithm is an average over a number of deterministic algorithms). Now consider a deterministic algorithm with error $\leqslant 1/3$ under μ, that makes T queries to x. We want to show that $T = \Omega(\sqrt{2^n})$.

§4.3 Classical Algorithms for Simon's Problem

First consider the case $s = 0^n$. We can assume the algorithm never queries the same point twice. Then the T outcomes of the queries are T distinct n-bit strings, and each sequence of T strings is equally likely. Now consider the case $s \neq 0^n$. Suppose the algorithm queries the indices $i_1, \cdots, i_T$ (this sequence depends on x) and gets outputs $x_{i_1}, \cdots, x_{i_T}$. Call a sequence of queries $i_1, \cdots, i_T$ good if it shows a collision (that is, $x_{i_k} = x_{i_\ell}$ for some $k \neq \ell$), and bad otherwise. If the sequence of queries of the algorithm is good, then we can find s, since $i_k \oplus i_\ell = s$. On the other hand, if the sequence is bad, then each sequence of T distinct outcomes is equally likely - just as in the $s = 0^n$ case! We will now show that the probability of the bad case is very close to 1 for small T.

§4.3 Classical Algorithms for Simon's Problem

First consider the case $s = 0^n$. We can assume the algorithm never queries the same point twice. Then the T outcomes of the queries are T distinct n-bit strings, and each sequence of T strings is equally likely. Now consider the case $s \neq 0^n$. Suppose the algorithm queries the indices $i_1, \cdots, i_T$ (this sequence depends on x) and gets outputs $x_{i_1}, \cdots, x_{i_T}$. Call a sequence of queries $i_1, \cdots, i_T$ good if it shows a collision (that is, $x_{i_k} = x_{i_\ell}$ for some $k \neq \ell$), and bad otherwise. If the sequence of queries of the algorithm is good, then we can find s, since $i_k \oplus i_\ell = s$. On the other hand, if the sequence is bad, then each sequence of T distinct outcomes is equally likely - just as in the $s = 0^n$ case! We will now show that the probability of the bad case is very close to 1 for small T.

§4.3 Classical Algorithms for Simon's Problem

First consider the case $s = 0^n$. We can assume the algorithm never queries the same point twice. Then the T outcomes of the queries are T distinct n-bit strings, and each sequence of T strings is equally likely. Now consider the case $s \neq 0^n$. Suppose the algorithm queries the indices $i_1, \cdots, i_T$ (this sequence depends on x) and gets outputs $x_{i_1}, \cdots, x_{i_T}$. Call a sequence of queries $i_1, \cdots, i_T$ good if it shows a collision (that is, $x_{i_k} = x_{i_\ell}$ for some $k \neq \ell$), and bad otherwise. If the sequence of queries of the algorithm is good, then we can find s, since $i_k \oplus i_\ell = s$. On the other hand, if the sequence is bad, then each sequence of T distinct outcomes is equally likely - just as in the $s = 0^n$ case! We will now show that the probability of the bad case is very close to 1 for small T.

§4.3 Classical Algorithms for Simon's Problem

If $i_1, \cdots, i_{k-1}$ is bad, then we have excluded **at most** C_2^{k-1} possible values of s (namely all values $i_j \oplus i_{j'}$ for all distinct $j, j' \in [k-1]$), and all other values of s are equally likely. The probability that the next query i_k makes the sequence good, is the probability that $x_{i_k} = x_{i_j}$ for some $j < k$, equivalently, that the set $S = \{i_k \oplus i_j \,|\, j < k\}$ happens to contain the string s. However, S has only $k-1$ members, while there are $2^n - 1 - C_2^{k-1}$ equally likely remaining possibilities for s. This means that the probability that the sequence is still bad after query i_k is made, is very close to 1. In formulas:

$$\Pr[i_1, \cdots, i_T \text{ is bad}] = \prod_{k=2}^{T} \Pr[i_1, \cdots, i_k \text{ is bad} | i_1, \cdots i_{k-1} \text{ is bad}]$$

$$= \prod_{k=2}^{T} \Big(1 - \frac{k-1}{2^n - 1 - C_2^{k-1}}\Big) \geqslant 1 - \sum_{k=2}^{T} \frac{k-1}{2^n - 1 - C_2^{k-1}},$$

here we used the fact that $(1-a)(1-b) \geqslant 1 - (a+b)$ if $a, b \geqslant 0$.

§4.3 Classical Algorithms for Simon's Problem

If $i_1, \cdots, i_{k-1}$ is bad, then we have excluded **at most** C_2^{k-1} possible values of s (namely all values $i_j \oplus i_{j'}$ for all distinct $j, j' \in [k-1]$), and all other values of s are equally likely. The probability that the next query i_k makes the sequence good, is the probability that $x_{i_k} = x_{i_j}$ for some $j < k$, equivalently, that the set $S = \{i_k \oplus i_j \,|\, j < k\}$ happens to contain the string s. However, S has only $k-1$ members, while there are $2^n - 1 - C_2^{k-1}$ equally likely remaining possibilities for s. This means that the probability that the sequence is still bad after query i_k is made, is very close to 1. In formulas:

$$\Pr[i_1, \cdots, i_T \text{ is bad}] = \prod_{k=2}^{T} \Pr[i_1, \cdots, i_k \text{ is bad} | i_1, \cdots i_{k-1} \text{ is bad}]$$

$$= \prod_{k=2}^{T} \Big(1 - \frac{k-1}{2^n - 1 - C_2^{k-1}}\Big) \geqslant 1 - \sum_{k=2}^{T} \frac{k-1}{2^n - 1 - C_2^{k-1}},$$

here we used the fact that $(1-a)(1-b) \geqslant 1 - (a+b)$ if $a, b \geqslant 0$.

§4.3 Classical Algorithms for Simon's Problem

If $i_1, \cdots, i_{k-1}$ is bad, then we have excluded **at most** C_2^{k-1} possible values of s (namely all values $i_j \oplus i_{j'}$ for all distinct $j, j' \in [k-1]$), and all other values of s are equally likely. The probability that the next query i_k makes the sequence good, is the probability that $x_{i_k} = x_{i_j}$ for some $j < k$, equivalently, that the set $S = \{i_k \oplus i_j \,|\, j < k\}$ happens to contain the string s. However, S has only $k-1$ members, while there are $2^n - 1 - C_2^{k-1}$ equally likely remaining possibilities for s. This means that the probability that the sequence is still bad after query i_k is made, is very close to 1. In formulas:

$$\Pr[i_1, \cdots, i_T \text{ is bad}] = \prod_{k=2}^{T} \Pr[i_1, \cdots, i_k \text{ is bad} | i_1, \cdots i_{k-1} \text{ is bad}]$$

$$= \prod_{k=2}^{T} \left(1 - \frac{k-1}{2^n - 1 - C_2^{k-1}}\right) \geqslant 1 - \sum_{k=2}^{T} \frac{k-1}{2^n - 1 - C_2^{k-1}},$$

here we used the fact that $(1-a)(1-b) \geqslant 1 - (a+b)$ if $a, b \geqslant 0$.

§4.3 Classical Algorithms for Simon's Problem

If $i_1, \cdots, i_{k-1}$ is bad, then we have excluded **at most** C_2^{k-1} possible values of s (namely all values $i_j \oplus i_{j'}$ for all distinct $j, j' \in [k-1]$), and all other values of s are equally likely. The probability that the next query i_k makes the sequence good, is the probability that $x_{i_k} = x_{i_j}$ for some $j < k$, equivalently, that the set $S = \{i_k \oplus i_j \,|\, j < k\}$ happens to contain the string s. However, S has only $k-1$ members, while there are $2^n - 1 - C_2^{k-1}$ equally likely remaining possibilities for s. This means that the probability that the sequence is still bad after query i_k is made, is very close to 1. In formulas:

$$\Pr[i_1, \cdots, i_T \text{ is bad}] = \prod_{k=2}^{T} \Pr[i_1, \cdots, i_k \text{ is bad} | i_1, \cdots i_{k-1} \text{ is bad}]$$

$$= \prod_{k=2}^{T} \Big(1 - \frac{k-1}{2^n - 1 - C_2^{k-1}}\Big) \geqslant 1 - \sum_{k=2}^{T} \frac{k-1}{2^n - 1 - C_2^{k-1}},$$

here we used the fact that $(1-a)(1-b) \geqslant 1 - (a+b)$ if $a, b \geqslant 0$.

§4.3 Classical Algorithms for Simon's Problem

Note that $2^n - 1 - C_2^{k-1} \approx 2^n$ as long as $k \ll \sqrt{2^n}$ and $\sum_{k=2}^{T}(k-1) = \frac{T(T-1)}{2} \approx T^2/2$. Hence we can approximate the last term in the formula by $1 - T^2/2^{n+1}$ if $k \ll \sqrt{2^n}$. Accordingly, if $T \ll \sqrt{2^n}$ then with probability nearly 1 (probability taken over the distribution μ) the algorithm's sequence of queries is bad. If it gets a bad sequence, it cannot "see" the difference between the $s = 0^n$ case and the $s \neq 0^n$ case, since both cases result in a uniformly random sequence of T distinct n-bit strings as answers to the T queries. This shows that T has to be $\sqrt{2^n}$ in order to enable the algorithm to get a good sequence of queries with high probability.

§4.3 Classical Algorithms for Simon's Problem

Note that $2^n - 1 - C_2^{k-1} \approx 2^n$ as long as $k \ll \sqrt{2^n}$ and $\sum_{k=2}^{T}(k-1) = \frac{T(T-1)}{2} \approx T^2/2$. Hence we can approximate the last term in the formula by $1 - T^2/2^{n+1}$ if $k \ll \sqrt{2^n}$. Accordingly, if $T \ll \sqrt{2^n}$ then with probability nearly 1 (probability taken over the distribution μ) the algorithm's sequence of queries is bad. If it gets a bad sequence, it cannot "see" the difference between the $s = 0^n$ case and the $s \neq 0^n$ case, since both cases result in a uniformly random sequence of T distinct n-bit strings as answers to the T queries. This shows that T has to be $\sqrt{2^n}$ in order to enable the algorithm to get a good sequence of queries with high probability.

§4.3 Classical Algorithms for Simon's Problem

Note that $2^n - 1 - C_2^{k-1} \approx 2^n$ as long as $k \ll \sqrt{2^n}$ and $\sum_{k=2}^{T}(k-1) = \dfrac{T(T-1)}{2} \approx T^2/2$. Hence we can approximate the last term in the formula by $1 - T^2/2^{n+1}$ if $k \ll \sqrt{2^n}$. Accordingly, if $T \ll \sqrt{2^n}$ then with probability nearly 1 (probability taken over the distribution μ) the algorithm's sequence of queries is bad. If it gets a bad sequence, it cannot "see" the difference between the $s = 0^n$ case and the $s \neq 0^n$ case, since both cases result in a uniformly random sequence of T distinct n-bit strings as answers to the T queries. This shows that T has to be $\sqrt{2^n}$ in order to enable the algorithm to get a good sequence of queries with high probability.