WIKIPEDIA

Simon's problem

In the computational complexity theory and quantum computing, **Simon's problem** is a computational problem that can be solved exponentially faster on a quantum computer than on a classical (or traditional) computer. Although the problem itself is of little practical value, it can be proved that a quantum algorithm can solve this problem exponentially faster than any known classical algorithm.[1]

The problem is set in the model of decision tree complexity or query complexity and was conceived by Daniel Simon in 1994.[2] Simon exhibited a quantum algorithm, usually called **Simon's algorithm**, that solves the problem exponentially faster than any deterministic or probabilistic classical algorithm, requiring exponentially less computation time (or more precisely, queries) than the best classical probabilistic algorithm.

This problem yields an oracle separation between the complexity classes BPP and BQP, unlike the separation provided by the Deutsch–Jozsa algorithm, which separates P and EQP.

Simon's algorithm was also the inspiration for Shor's algorithm. Both problems are special cases of the Abelian hidden subgroup problem, which is now known to have efficient quantum algorithms.

Contents

Problem description

Given a function (implemented by a black box or oracle) $f:\{0,1\}^n \to \{0,1\}^n$, promised to satisfy the property that, for some $s \in \{0,1\}^n$, we have, for all $x, y \in \{0,1\}^n$,

$$f(x) = f(y) \text{ if and only if } x \oplus y \in \{0^n, s\}$$

In the case $s = 0^n$, then f is *required* to be a one-to-one function (otherwise it is a two-to-one function, that is, two inputs map to the same output). Note that $x \oplus y = 0^n$ if and only if $x = y$.

So, in other words, f is a function such that $f(x) = f(x \oplus s)$, for all $x \in \{0,1\}^n$ and given some fixed and unknown $s \in \{0,1\}^n$.

The problem is to find s.

Example

For example, if $n = 3$, then the following function is an example of a function that satisfies the required and just mentioned property:

x	$f(x)$
000	101
001	010
010	000
011	110
100	000
101	110
110	101
111	010

In this case, $s = 110$ (i.e. the solution). It can easily be verified that every output of f occurs twice, and the two input strings corresponding to any one given output have bitwise XOR equal to $s = 110$. For example, the input strings 010 and 100 are both mapped (by f) to the same output string 000. If we XOR 010 and 100 we obtain s, that is

$$010 \oplus 100 = 110 = s.$$

Note that, in this example, the function f is indeed a two-to-one function.

Problem hardness

Intuitively, this is a very hard problem to solve in a "classical" way, even if one uses randomness and accepts a small probability of error. The intuition behind the hardness is reasonably simple: if you want to solve the problem classically, you need to find two different inputs x and y for which $f(x) = f(y)$. There is not necessarily any structure in the function f that would help us to find two such inputs: more specifically, we can discover something about f (or what it does) only when, for two different inputs, we obtain the same output. In any case, we would need to guess $\Omega(\sqrt{2^n})$ different inputs before being likely to find a pair on which f takes the same output, as per the birthday problem.

Overview of Simon's algorithm

Idea

The high-level idea behind Simon's algorithm is to "probe" (or "sample") a quantum circuit (see the picture below) "enough times" to find $n-1$ (linearly independent) n-bit strings, that is

$$y_1, y_2, \ldots, y_{n-1} \in \{0,1\}^n,$$

such that the following equations are satisfied

$$\begin{cases} y_1 \cdot s = 0 \\ y_2 \cdot s = 0 \\ \quad \vdots \\ y_{n-1} \cdot s = 0 \end{cases}$$

where $y_i \cdot s$ is the modulo-2 dot product; that is, $y_i \cdot s = y_{i1}s_1 \oplus y_{i2}s_2 \oplus \cdots \oplus y_{in}s_n$, and $y_{ij}, s_j \in \{0,1\}$, for $i = 1, \ldots, n-1$ and for $j = 1, \ldots, n$.

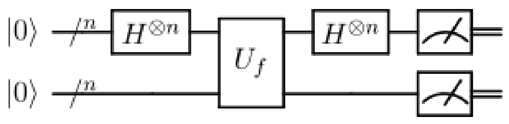
So, this linear system contains $n-1$ linear equations in n unknowns (i.e. the bits of $s \in \{0,1\}^n$), and the goal is to solve it to obtain s. Note that s is fixed for a given function f. Note also that we may not find a (unique) solution.

Simon's quantum circuit

The quantum circuit (see the picture) is the implementation (and visualization) of the quantum part of Simon's algorithm.

A quantum state of all zeros is first prepared (this can easily be done). Half of this state is then transformed using a Hadamard transform. The result is then fed into an "oracle" (or "black box"), which knows how to compute f. After that, part of the output produced by the "oracle" is transformed using another Hadamard transform. Finally, a measurement on the overall resulting quantum state is performed. It is during this measurement that we retrieve the n-bit strings, $y_1, y_2, \ldots, y_{n-1} \in \{0,1\}^n$, mentioned in the previous sub-section.

Simon's algorithm can be thought of as an iterative algorithm (which makes use of a quantum circuit) followed by a (possibly) "classical" algorithm to find the solution to a linear system of equations.



Quantum circuit representing/implementing Simon's algorithm

Simon's algorithm

In this section, each part of Simon's algorithm is explained (in detail). It may be useful to look at the picture of Simon's quantum circuit above while reading each of the following sub-sections.

Input

Simon's algorithm starts with the input $|0^n\rangle \otimes |0^n\rangle = |0^n\rangle|0^n\rangle$, where $|0^n\rangle$ is the quantum state with n zeros.

(The symbol $\otimes$ is the typical symbol used to represent the tensor product. To not clutter the notation, the symbol $\otimes$ is sometimes omitted: for example, in the previous sentence, $|0^n\rangle \otimes |0^n\rangle$ is equivalent to $|0^n\rangle|0^n\rangle$. In this article, it is (often) used to remove ambiguity or to avoid confusion.)

Example

So, for example, if $n = 2$, then the initial input is

$$|0^2\rangle \otimes |0^2\rangle = |00\rangle \otimes |00\rangle = (|0\rangle \otimes |0\rangle) \otimes (|0\rangle \otimes |0\rangle) = |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle = |0000\rangle = |0^4\rangle = |0^{2n}\rangle.$$

First Hadamard transformation

After that, the input (as described in the previous sub-section) is transformed using a Hadamard transform. Specifically, the Hadamard transform $H^{\otimes n}$ (note that the tensor product can also be applied to matrices) is applied to the first n qubits, that is, to the "partial" state $|0^n\rangle$, so that the composite state after this operation is

$$|\Psi\rangle = \left(H^{\otimes n}|0^n\rangle\right) \otimes |0^n\rangle = \left(\sum_{x\in\{0,1\}^n} \frac{1}{\sqrt{2^n}}|x\rangle\right) \otimes |0^n\rangle = \left(\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\right) \otimes |0^n\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{x\in\{0,1\}^n} (|x\rangle \otimes |0^n\rangle)$$

where $x \in \{0,1\}^n$ denotes any n-bit string (i.e. the summation is over any n-bit string). The term $\frac{1}{\sqrt{2^n}}$ can be factored out of the summation because it does not depend on x (i.e. it is a constant with respect to x). Note that $\frac{1}{\sqrt{2^n}} = \frac{1}{2^{\frac{n}{2}}}$.

Example

Suppose (again) $n = 2$, then the input is $|0000\rangle$ and the Hadamard transform $H^{\otimes 2}$ is

$$H^{\otimes 2} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} \otimes \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} & \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\\ \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} & -\left(\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\right)\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1 & 1 & 1 & 1\\1 & -1 & 1 & -1\\1 & 1 & -1 & -1\\1 & -1 & -1 & 1\end{bmatrix}$$

If we now apply $H^{\otimes 2}$ to the first $|00\rangle$, i.e. to the state

$$|00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

we obtain

$$H^{\otimes 2}|00\rangle = \frac{1}{2}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$
$$= \frac{1}{2}\left(\begin{bmatrix}1\\0\\0\\0\end{bmatrix}+\begin{bmatrix}0\\1\\0\\0\end{bmatrix}+\begin{bmatrix}0\\0\\1\\0\end{bmatrix}+\begin{bmatrix}0\\0\\0\\1\end{bmatrix}\right) = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right) = \frac{1}{2^{2/2}}\sum_{x\in\{0,1\}^2}|x\rangle$$

To obtain the final composite quantum state, we can now tensor product $H^{\otimes 2}|00\rangle$ with $|00\rangle$, that is

$$\left(H^{\otimes 2}|00\rangle\right)\otimes|00\rangle = \left(\frac{1}{2}\sum_{x\in\{0,1\}^2}|x\rangle\right)\otimes|00\rangle = \frac{1}{2}\left(|00\rangle+|01\rangle+|10\rangle+|11\rangle\right)\otimes|00\rangle$$
$$= \frac{1}{2}\left(|00\rangle\otimes|00\rangle+|01\rangle\otimes|00\rangle+|10\rangle\otimes|00\rangle+|11\rangle\otimes|00\rangle\right) = \frac{1}{2}\sum_{x\in\{0,1\}^2}\left(|x\rangle\otimes|00\rangle\right).$$

Oracle

We then call the oracle or black-box (U_f in the picture above) to compute the function f on the transformed input $|\Psi\rangle = \frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}\left(|x\rangle\otimes|0^n\rangle\right)$, to obtain the state

$$|\Psi\rangle' = \frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}\left(|x\rangle\otimes|f(x)\rangle\right)$$

Second Hadamard transformation

We then apply the Hadamard transform $H^{\otimes n}$ to the states of the first n qubits of the state $|\Psi\rangle'$, to obtain

$$|\Psi\rangle'' = \frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}\left(\left(H^{\otimes n}|x\rangle\right)\otimes|f(x)\rangle\right)$$
$$= \frac{1}{2^{n/2}}\sum_{x\in\{0,1\}^n}\left(\left(\frac{1}{2^{n/2}}\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}|y\rangle\right)\otimes|f(x)\rangle\right) = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\left(\sum_{y\in\{0,1\}^n}\left((-1)^{x\cdot y}|y\rangle\otimes|f(x)\rangle\right)\right)$$

where $(-1)^{x\cdot y}$ can either be -1 or 1, depending on $x\cdot y = x_1y_1+\cdots+x_ny_n$, where $x_i, y_i\in\{0,1\}$, for $i=1,\ldots,n$. So, for example, if $x=101$ and $y=111$, then $x\cdot y = 1*1+0*1+1*1 = 2$, which is an even number. Thus, in this case, $(-1)^{x\cdot y} = (-1)^{1*1+0*1+1*1} = (-1)^2 = 1$. Note that $x\cdot y$ is always a non-negative number.

Intuition behind this inverse Hadamard transformation that is applied here can be found on CMUs lecture notes (https://www.cs.cmu.edu/~odonnell/quantum18/lecture11.pdf)

Let's now rewrite

$$|\Psi\rangle'' = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\left(\sum_{y\in\{0,1\}^n}\left((-1)^{x\cdot y}|y\rangle\otimes|f(x)\rangle\right)\right)$$

as follows

$$|\Psi\rangle'' = \sum_{y\in\{0,1\}^n}\left(|y\rangle\otimes\left(\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\left((-1)^{x\cdot y}|f(x)\rangle\right)\right)\right)$$

This manipulation will be convenient to understand the explanations in the next sections. Note that we have switched the order of the summations.

Measurement

After having performed all previously described operations, at the end of the circuit, a measurement is performed.

There are now two possible cases we need to consider separately

- $x \oplus y = 0^n$ or
- $x \oplus y = s$, where $s \neq 0^n$.

First case

Let's first analyze the (special) case $x \oplus y = 0^n$, which means that f is (by requirement) a one-to-one function (as explained above in the "problem description").

Let's keep in mind that the quantum state before the measurement is

$$\sum_{y \in \{0,1\}^n} |y\rangle \otimes \left(\frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |f(x)\rangle \right) \right)$$

Now, the probability that the measurement results in each string $y \in \{0,1\}^n$ is

$$p_y = \left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |f(x)\rangle \right) \right\|^2 = \frac{1}{2^n}$$

One way to see that the previous equation is true is to note that

$$\left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |f(x)\rangle \right) \right\|^2 = \left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |x\rangle \right) \right\|^2$$

because the two vectors only differ in the ordering of their entries, given that f is one-to-one.

The value of the right-hand side, that is

$$\left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |x\rangle \right) \right\|^2$$

is more easily seen to be $\frac{1}{2^n}$.

Thus, when $x \oplus y = 0^n$, the outcome is simply a uniformly distributed n-bit string.

Second case

Let's now analyze the case $x \oplus y = s$, where $s \neq 0^n$. Note that, in this case, f is a two-to-one function, that is, there are two inputs that map to the same output of f.

The analysis performed in the first case is still valid for this second case, that is, the probability to measure any given string $y \in \{0,1\}^n$ can still be represented as

$$p_y = \left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |f(x)\rangle \right) \right\|^2$$

However, in this second case, we still need to figure out what this value of p_y is. Let's see why in the following explanations.

Let $A = range(f)$. Let $z \in A$ (i.e. z is some output of the function f), then note that, for every $x_1 \in \{0,1\}^n$, there is one (and only one) $x_2 \in \{0,1\}^n$, such that $f(x_1) = f(x_2) = z$; moreover, we also have $x_1 \oplus x_2 = s$, which is equivalent to $x_2 = s \oplus x_1$ (see "the problem description" section above for a review of this concept).

Hence, we have

$$p_y = \left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |f(x)\rangle \right) \right\|^2 = \left\| \frac{1}{2^n} \sum_{z \in A} \left(\left((-1)^{x_1 \cdot y} + (-1)^{x_2 \cdot y} \right) |z\rangle \right) \right\|^2$$

Given that $x_2 = s \oplus x_1$, then we can rewrite the coefficient $(-1)^{x_1 \cdot y} + (-1)^{x_2 \cdot y}$ as follows

$$(-1)^{x_1 \cdot y} + (-1)^{x_2 \cdot y} = (-1)^{x_1 \cdot y} + (-1)^{(x_1 \oplus s) \cdot y}$$

Given that $(x_1 \oplus s) \cdot y = (x_1 \cdot y) \oplus (s \cdot y)$, then we can further write the expression above as

$$(-1)^{x_1 \cdot y}(1 + (-1)^{y \cdot s})$$

So, p_y can further be written as

$$p_y = \left\| \frac{1}{2^n} \sum_{z \in A} \left((-1)^{x_1 \cdot y}(1 + (-1)^{y \cdot s}) |z\rangle \right) \right\|^2$$

Odd number

Now, if $y \cdot s = y_1 s_1 + \cdots + y_n s_n$ is an odd number, then $(-1)^{y \cdot s} = -1$. In that case,

$$(-1)^{x_1 \cdot y}(1 + (-1)^{y \cdot s}) = (-1)^{x_1 \cdot y}(1 - 1) = 0$$

Consequently, we have

$$p_y = \left\| \frac{1}{2^n} \sum_{z \in A} \left((-1)^{x_1 \cdot y}(1 + (-1)^{y \cdot s}) |z\rangle \right) \right\|^2 = 0$$

Given that $p_y = 0$, then we never have this case, that is, no string $y \in \{0,1\}^n$ is seen (after the measurement) in this case.

(This is the case where we have *destructive interference*.)

Even number

If, instead, $y \cdot s$ is an even number (e.g. zero), then $(-1)^{y \cdot s} = 1$. In that case,

$$(-1)^{x_1 \cdot y}(1 + (-1)^{y \cdot s}) = (-1)^{x_1 \cdot y} 2$$

So, we have

$$p_y = \left\| \frac{1}{2^n} \sum_{z \in A} \left((-1)^{x_1 \cdot y}(2) |z\rangle \right) \right\|^2 = \left\| \frac{2}{2^n} \sum_{z \in A} \left((-1)^{x_1 \cdot y} |z\rangle \right) \right\|^2 = \left\| \frac{1}{2^{n-1}} \sum_{z \in A} \left((-1)^{x_1 \cdot y} |z\rangle \right) \right\|^2$$

Note that $\frac{2}{2^n} = 2 * 2^{-n} = 2^1 * 2^{-n} = 2^{-n+1} = 2^{-(n-1)} = \frac{1}{2^{n-1}}$.

(This is the case where we have *constructive interference*.)

So, in summary, for this second case, we have the following probabilities

$$p_y = \left\| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left((-1)^{x \cdot y} |f(x)\rangle \right) \right\|^2 = \begin{cases} \frac{1}{2^{n-1}} & \text{if } y \cdot s \text{ is even} \\ 0 & \text{if } y \cdot s \text{ is odd} \end{cases}$$

Classical post-processing

When we run the circuit (operations) above, there are two cases:

- in the (special) case where $x \oplus y = 0^n$, the measurement results in each string $y \in \{0,1\}^n$ with probability $p_y = \frac{1}{2^n}$
- in the case $x \oplus y = s$ (where $s \neq 0^n$), the probability to obtain each string $y \in \{0,1\}^n$ is given by

$$p_y = \begin{cases} \frac{1}{2^{n-1}} & \text{if } y \cdot s \text{ is even} \\ 0 & \text{if } y \cdot s \text{ is odd} \end{cases}$$

Thus, in both cases, the measurement results in some string $y \in \{0,1\}^n$ that satisfies $s \cdot y = 0$, and the distribution is uniform over all of the strings that satisfy this constraint.

Is this enough information to determine s? The answer is "yes", provided that the process (above) is repeated several times (and a small probability of failure is accepted). Specifically, if the above process is run $n-1$ times, we get $n-1$ strings $y_1, y_2, \ldots, y_{n-1} \in \{0,1\}^n$, such that

$$\begin{cases} y_1 \cdot s &= 0 \\ y_2 \cdot s &= 0 \\ &\vdots \\ y_{n-1} \cdot s &= 0 \end{cases}$$

This is a system if $n-1$ linear equations in n unknowns (i.e. the bits of s), and the goal is to solve it to obtain s. Note that each of the $y_1, y_2, \ldots, y_{n-1} \in \{0,1\}^n$ that we obtain after each measurement (for each "round" of the process) is, of course, the result of a measurement, so it is known (at the end of each "round").

We only get a unique non-zero solution s if we are "lucky" and $y_1, y_2, \ldots, y_{n-1} \in \{0,1\}^n$ are linearly independent. The probability that $y_1, y_2, \ldots, y_{n-1}$ are linearly independent is at least

$$\prod_{k=1}^{\infty} \left(1 - \frac{1}{2^k}\right) = 0.288788\cdots > \frac{1}{4}$$

If we have linear independence, we can solve the system to get a candidate solution $s' \neq 0^n$ and test that $f(0^n) = f(s')$. If $f(0^n) = f(s')$, we know that $s = s'$, and the problem has been solved. If $f(0^n) \neq f(s')$, it must be that $s = 0^n$ (because, if this were not so, the unique non-zero solution to the linear equations would have been s). Either way, once we have linear independence, we can solve the problem.

Complexity

Simon's algorithm requires $O(n)$ queries to the black box, whereas a classical algorithm would need at least $\Omega(2^{n/2})$ queries. It is also known that Simon's algorithm is optimal in the sense that *any* quantum algorithm to solve this problem requires $\Omega(n)$ queries.[3][4]

See also

- Deutsch–Jozsa algorithm

References

1. Arora, Sanjeev and Barak, Boaz. *Computational Complexity: A Modern Approach*. Cambridge University Press.
2. Simon, D.R. (1994), "On the power of quantum computation" (http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.51.5477&rep=rep1&type=pdf), *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*: 116–123, retrieved 2011-06-06
3. Koiran, P.; Nesme, V.; Portier, N. (2007), "The quantum query complexity of the Abelian hidden subgroup problem" (http://perso.ens-lyon.fr/pascal.koiran/Publis/lip.05-17.ps), *Theoretical Computer Science*, **380** (1–2): 115–126, doi:10.1016/j.tcs.2007.02.057 (https://doi.org/10.1016%2Fj.tcs.2007.02.057), retrieved 2011-06-06
4. Koiran, P.; Nesme, V.; Portier, N. (2005), "A quantum lower bound for the query complexity of Simon's Problem" (http://perso.ens-lyon.fr/pascal.koiran/Publis/icalp05.ps), *Proc. ICALP*, **3580**: 1287–1298, arXiv:quant-ph/0501060 (https://arxiv.org/abs/quant-ph/0501060), Bibcode:2005quant.ph..1060K (https://ui.adsabs.harvard.edu/abs/2005quant.ph..1060K), retrieved 2011-06-06