Algorithms, Complexity and the Quantum Fourier Transform

Artur Ekert and Alastair Kay

Some problems are easy to solve and some are really hard. A quantum computer offers an enormous gain in the use of computational resources such as time and memory—though only in certain types of computation. This brings us to computational complexity which is all about bounds on resources computers can use. Quantum Fourier Transform appears as an important example.

I. ALGORITHMS AND THEIR COMPLEXITY

In order to solve a particular problem, computers follow a precise set of instructions that can be mechanically applied to yield the solution to any given instance of the problem. A specification of this set of instructions is called an algorithm. Examples of algorithms are the procedures taught in elementary schools for adding and multiplying whole numbers; when these procedures are mechanically applied, they always yield the correct result for any pair of whole numbers. Any algorithm can be represented by a family of Boolean circuits $(C_1, C_2, C_3, ...)$, where the network C_n acts on all possible input instances of size n bits. Any useful algorithm should have such a family specified by an example circuit C_n and *a simple rule* explaining how to construct the circuit C_{n+1} from the circuit C_n. These are called *uniform* families of circuits or networks.

When we build circuits, we care about the economy of resources. After all, each gate we use may cost us money ($\equiv$ energy $\equiv$ time etc.). Circuits with logarithmic $O(\log n)$, linear $O(n)$, or polynomial size (number of gates) are considered reasonable, whereas circuits with exponential size, $O(c^n)$, are viewed as rather expensive. In the language of network complexity - an algorithm is said to be efficient if it has a uniform and polynomially-bounded family of circuits ($O(n^c)$ for some constant c). Let us take a look at some examples.

II. QUANTUM FOURIER TRANSFORM

The quantum Hadamard transform has, obviously, a uniform family of networks whose size is growing as n with the number of input qubits. Another good example of a uniform family of networks is the quantum Fourier transform (QFT) on n qubits defined in the computational basis as the unitary operation

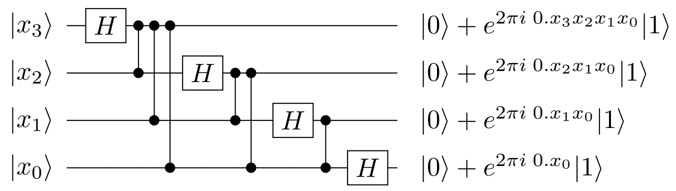
$$|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_y e^{i\frac{2\pi}{2^n}xy}|y\rangle.$$

This expression uses the binary (for the state vectors) and decimal (in the exponent) forms of the numbers x, y interchangeably. After dropping the normalisation factor, it can be written in the tensor product form

$$|x\rangle \mapsto (|0\rangle + e^{2\pi i\ 0.x_0}|1\rangle)(|0\rangle + e^{2\pi i\ 0.x_1x_0}|1\rangle)\dots$$
$$\dots(|0\rangle + e^{2\pi i\ 0.x_{n-1}\dots x_1x_0}|1\rangle).$$

where $x = \sum_k x_k 2^k$, $|x\rangle \equiv |x_{n-1}\rangle \dots |x_1\rangle|x_0\rangle$ and we use the binary point notation, e.g. $0.x_2x_1x_0$ is equal to $x_2/2 + x_1/4 + x_0/8$.

Suppose we want to *construct* such a unitary evolution on n qubits using our repertoire of quantum logic gates. We can start with a single qubit and notice that in this case the QFT is reduced to applying a Hadamard gate. Then we can take two qubits and notice that the QFT can be implemented with two Hadamard gates and the controlled phase shift $B(\pi/2)$ in between. Progressing this way we can construct the three qubit QFT and the four qubit QFT, whose network looks like this:


$|x_3\rangle$
$|x_2\rangle$
$|x_1\rangle$
$|x_0\rangle$
H
H
H
H
$|0\rangle + e^{2\pi i\ 0.x_3x_2x_1x_0}|1\rangle$
$|0\rangle + e^{2\pi i\ 0.x_2x_1x_0}|1\rangle$
$|0\rangle + e^{2\pi i\ 0.x_1x_0}|1\rangle$
$|0\rangle + e^{2\pi i\ 0.x_0}|1\rangle$

The order of the qubits at the output is reversed and there are three different types of the $B(\phi)$ gates in the network above: $B(\pi/2)$, $B(\pi/4)$ and $B(\pi/8)$. First H is applied to the first qubit (counting from the top) followed by the controlled phase shifts $B(\pi/2)$, $B(\pi/4)$ and $B(\pi/8)$, then H is applied to the second qubit followed by the controlled phase shifts $B(\pi/2)$ and $B(\pi/4)$, then H is applied to the third qubit followed by the controlled phase shift $B(\pi/2)$, and finally H is applied to the fourth qubit. The general case of n qubits requires a trivial extension of the network following the same sequence pattern of gates H and B. The QFT network operating on n qubits contains n Hadamard gates H and $n(n-1)/2$ phase shifts B, in total $n(n+1)/2$ elementary gates. Hence, the QFT has a uniform family of networks whose size grows only as a quadratic function of the size of the input, i.e. $O(n^2)$. Changing from one set of gates to another can only affect the network size by a multiplicative constant which does not affect the quadratic scaling with n. Thus the complexity of the QFT is $O(n^2)$ no matter which set of adequate gates we use.

III. COMPLEXITY REVISITED

There are basically three different types of Boolean networks: classical deterministic, classical probabilistic, and quantum. They correspond to, respectively, deterministic, randomised, and quantum algorithms.

Classical deterministic networks are based on logical connectives such as AND, OR, and NOT and are required to always deliver correct answers. If a problem admits a deterministic uniform network family of polynomial size, we say that the problem is in the class P.

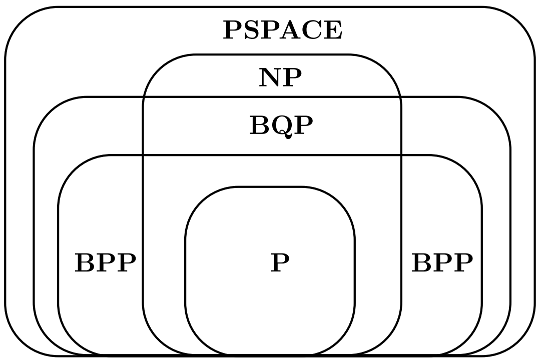
Probabilistic networks have additional "coin flip" gates which do not have any inputs and emit one uniformly-distributed random bit when executed during a computation. Despite the fact that probabilistic networks may generate erroneous answers they may be very useful. A good example is primality testing – given an n-bit number, x, decide whether or not x is prime. Until 2002, when it was shown that primality testing is in P, the smallest known uniform deterministic network family that could solve this problem was of size $O(n^{d \log\log n})$, which was not polynomially bounded. In contrast a probabilistic algorithm that can solve the same problem with a uniform probabilistic network family of size $O(n^3 \log(1/\epsilon))$, where ϵ is the probability of error, has been known since 1976. *N.B.* ϵ does not depend on n and we can choose it as small as we wish and still get an efficient algorithm. The $\log(1/\epsilon)$ factor can be explained by Imagining a probabilistic circuit that computes a Boolean function and that errs with probability smaller than $\frac{1}{2}-\delta$ for fixed $\delta > 0$. If you run r of these networks in parallel (so that the size of the overall network is increased by factor r) and then use the majority voting for the final "yes" or "no" answer your overall probability of error will be bounded by $\epsilon = \exp(-\delta^2 r)$. (This follows directly from the Chernoff bound.) Hence r is of the order $\log(1/\epsilon)$. If a problem admits such a family of networks then we say the problem is in the class BPP (stands for "bounded-error probabilistic polynomial").

It is an open question whether randomness adds any power to computation. It is very likely that the primality-testing is not an exception and any randomised algorithm can be derandomised. Even though it is yet to be demonstrated, there are many indications that P equals BPP.

Last but not least we have quantum algorithms, or families of quantum networks, which are believed to be more powerful than their probabilistic counterparts. The example here is the factoring problem – given an n-bit number x, find a list of prime factors of x. The smallest known uniform probabilistic network family which solves the problem is of size $O(2^{d\sqrt{n \log n}})$. One reason why quantum computation is such a fashionable field today is the discovery, by Peter Shor, of a uniform family of quantum networks of $O(n^2 \log\log n \log(1/\epsilon))$ in size, that solve the factoring problem. If a problem admits a uniform quantum network family of polynomial size that for any input gives the right answer with probability larger than $\frac{1}{2}+\delta$ for fixed $\delta > 0$ then we say the problem is in the class BQP (stands for "bounded-error quantum polynomial").

Quantum networks are potentially more powerful because of multiparticle quantum interference, an inherently quantum phenomenon which makes the quantum theory radically different from any classical statistical theory. We assemble n qubits, prepare them in the initial input state $|0\rangle|0\rangle \ldots |0\rangle$ and evolve the state through a prescribed sequence of quantum operations. Finally we measure the qubits and the outcome of the measurement is the output of the computation. Any attempt to simulate this process on a classical computer requires keeping track of all of the $O(2^n)$ components of the state vector. Even for a modest number of qubits, say $n = 100$, we need to follow the evolution of $2^{100} \approx 10^{30}$ numbers, which is far beyond the computational capacity of any foreseeable classical computer.

The fact that it is impossible, in general, to simulate an evolution of a quantum system on a classical computer in an efficient way led Richard Feynman to point out that if it requires that much computation to work out what will happen in, say a quantum multi-particle interference experiment, then the very act of setting up such an experiment and measuring the outcome is equivalent to performing a complex computation. Thus what at the first glance looks like an obstacle is, in fact, an opportunity!



Here is a simplified diagram showing some of the complexity classes and their presumed relationships. We have added two more classes, NP and $PSPACE$, for it is difficult to talk about computational complexity without mentioning NP. Sometimes the problem may be hard to solve, but a solution can be easily verified once found, e.g. factoring. Such problems form a complexity class called NP. The most famous conjecture of complexity theory asserts that NP problems do really exist i.e. recognising the answer efficiently does not imply that we are able to find it efficiently. Thus we believe that

$$P \neq NP$$

but the statement still awaits proof. Let us mention in passing that most NP problems are believed to be outside BQP, meaning that even a quantum computer would require more than a polynomial number of steps to solve them. We do know though that BQP cannot extend outside the class known as PSPACE, which contains problems that a conventional computer can solve using only a polynomial amount of memory but an unlimited number of steps.