CS903: Quantum computation and Information theory (Special Topics In TCS)

Lecture 1: Perspective

1 Coverage

In this course we will study:

1. Quantum mechanics
2. Computer Science/classical complexity theory
3. Quantum computation
4. Noise and Information theory
5. Quantum Information and quantum error correction
6. Quantum cryptography

2 Quantum mechanics

What it is:

- A mathematical framework or set of rules for the construction of physical theories.
- It is *hard*, be warned and study every day. The mathematics is not hard though and is based on linear algebra and probability theory.
- It is an axiomatic system; a set of four postulates.
- Came about in early 1920's primarily because of inadequacies of *classical* (anything that is not quantum) theories
- Physical models of almost everything under (and also inside) the sun seem to be based on principles of quantum mechanics. These include: structure of atoms, nuclear fusion in stars, superconductors, structure of the DNA, elementary particles of nature ... Only gravitation appear to have resisted all attempts to be completely incorporated in to the framework of quantum mechanics.
- No deviation has ever been found.
- Most believe, that a "theory of everything", should one ever be found, will have to be built within the frameworks of quantum mechanics.

What it is not: A complete physical theory of the world.

For example, the rules of quantum mechanics do not tell us how electrons act with photons. *Quantum electrodynamics* is a theory (a set of rules), phrased within the rules of QM, which tells us how electrons and photons should be described and how should they interact.

3 Computer Science

3.1 It is ancient

Perhaps the most ancient of all (though it may not be a science in the strict sense)! Sophisticated algorithmic ideas can be found in the texts of the Egyptian, Babylonian and the Indus valley civilizations. Some of the first algorithms taught in CSL101/CSL102 are Euclid's GCD and the sieve of Eratosthenes from the Greeks (300 BC approx) and the fast powering (by Acharya Pingala in the Hindu Chandah-sutra, 200 BC approx). These are some of the first examples of formal algorithms (inductive/primitive recursive).

3.2 The Church-Turing thesis

Modern computer science has its foundations in formal logic. After the success of set theory, the great mathematician David Hilbert formulated a set of 30 problems with the aim to find 'mechanical procedures' to determine whether a mathematical statement is a theorem or not. Alan Turing (1936) developed in detail the notion of what we would now call a programmable computer, a model of computation now known as **Turing machine**. In particular he showed:

1. There can be a **Universal Turing machine** which can be used to simulate any Turing machine.
2. The *Universal Turing machine* completely captures what it means to perform a task by algorithmic means.
3. Hilbert's 23^{rd} problem (*The Entscheidungs problem*) is not computable by an *Universal Turing machine*. (Mathematician's could relax, they were not to be rendered job-less, after all).

The above led to an assertion called the **Church-Turing thesis:** *If an algorithm can be performed on any piece of hardware (including a modern computer) then there is an equivalent algorithm for a Universal Turing machine which performs the same task.*

3.3 The RAM model and the modern computer

Not long after Turing's paper, John Von Neumann put forth the **RAM model** (a model of the stored program computer) and the other theories necessary to put together, in a practical fashion, all the components required for a computer to be fully as capable as a *Universal Turing Machine*.

Actually, Von Neumann did a *bit* more than just describe the RAM model and its equivalence with *Turing machines*. He showed that within the classical model *information* can be harnessed in on/off switches called *bits*. He also showed how *noise* can be contained (by footing a small power bill) and bits can be realized using principles of *thermodynamics* and *information theory*.

In 1947 Bardeen, Brattain and Shockley developed the transistor and hardware development (of the Intel variety) took off.

3.4 Efficient computation and complexity theory

3.4.1 Strong Church-Turing thesis

With the advent of electronic computers people noticed that some computations took inordinate amount of resources in terms of *time* (to run) and *space* (in bits). With this came the notion of *efficiency*.

Roughly speaking, an *efficient* algorithm is one which runs in time polynomial in the size of the problem solved. In contrast, an *inefficient* algorithm takes super-polynomial (typically exponential) time.

What was noticed through the 1960's and the 1970's was that it seemed as though the *Turing machine* model of computation was at least as powerful as any other model of computation, in the sense that a problem which could be solved efficiently in any other model of computation could also be solved efficiently using the Turing machine model, by using the Turing machine to simulate the other model of computation. This led to the strengthened version of **Church-Turing thesis:**

Any algorithmic process can be simulated efficiently using a Turing machine.

3.4.2 Probabilistic version

In the mid 1970's Solovay and Strassen showed that it is possible to test whether an integer is a prime or composite using a *randomized* algorithm in which a probabilistic coin toss forms an essential part. The Solovay-Strassen test could determine that a number was prime with a certain probability or composite with certainty. By repeating the test a few times it was possible to determine whether the number was prime or composite with near certainty. The Solovay-Strassen randomized algorithm is efficient and till last year (Manindra Agarwal et. al., 2003) no deterministic polynomial time algorithm was known for the problem. It turns out that randomization facilitates efficient algorithm design in many situations. This led to the *strong Church-Turing thesis:*

Any algorithmic process can be simulated efficiently using a probabilistic Turing machine.

3.4.3 Decision problems and complexity classes

- A decision problem is said to be in the class **P** if there is a Turing machine which can solve the problem in time polynomial in the length of the input.
- A decision problem is said to be in the class **NP** if there exists a Turing machine that can verify a proposed solution in polynomial time.
- A decision problem is said to be in the class **BPP** if there exists a probabilistic Turing machine which can solve the problem with a bounded probability in time polynomial in the length of the input. (the probability bound is arbitrary. *Chernoff* bound can be used to amplify the probability to near 1 with only a few repetitions of the algorithm)
- A decision problem is said to be in the class **PSPACE** if it can be solved on a Turing machine using only a polynomial number or working bits.
- A decision problem is said to be in the class **L** if it can be solved on a Turing machine using only a logarithmic number or working bits.

- A decision problem is said to be in the class **EXP** if there is a Turing machine which can solve the problem in time exponential in the length of the input.

Clearly

$$\mathbf{L} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$$

[Why? Prove them.]

Although each of these inclusions are widely believed to be strict, none of them has ever been proved to be strict. Though the *time hierarchy theorem* tells us that $\mathbf{P} \subset \mathbf{EXP}$ and the *space hierarchy theorem* tells us that $\mathbf{L} \subset \mathbf{PSPACE}$.

Also, it is obvious that

$$\mathbf{P} \subseteq \mathbf{BPP}$$

We will study these complexity classes and their properties in more detail later.

4 Quantum computation

Why should we at all think of doing computation using Quantum mechanics? Three reasons:

4.1 Complexity

1. In 1982 Feynman and Benioff independently pointed out that quantum systems can be used to do computation. Feynman's main motivation was simulation of physical systems.

2. Feynman also pointed out that it seems impossible to be able to efficiently simulate even small quantum systems on the Turing machine using only polynomial amount of resources. He speculated that a 'quantum mechanical computer' will be more powerful than a Universal Turing machine.

3. In 1985 David Deutsch tried to define a computational device that would be capable of efficiently simulating an arbitrary physical system (quantum mechanical). He defined the 'Universal Quantum Computer' and formalized the notion of quantum computation. **BQP** is the name of the class of problems that a quantum computer can solve efficiently.

4. Once we define Quantum computation we will be able to show (straightforward) that a classical Turing machine can simulate a quantum computer. So, we get nothing more in terms of computability.

5. In 1994 Peter Shor showed that factoring (finding the prime factors of an integer) can be done efficiently (n^3 for an n bit number) on a quantum computer. Shor's solution is based on 'quantum Fourier transform' and follows an earlier work by Simon. The best known classical algorithm (*the number field sieve*) works in time $exp(cn^{(1/3)} \log n^{(2/3)})$. The problem is believed to be in the class **NPI**, a non-empty class of intermediate hard problems if $\mathbf{P} \neq \mathbf{NP}$.

6. In 1995 Lov Grover (an IITD alumnus) showed that searching in an unstructured domain of size n can be speeded up to $O(\sqrt{n})$ using quantum computation. If search could be speeded up further then it would have been a great way to search for solutions **NPC** problems, whose solutions can anyway be verified efficiently. However, this was not to be - Bennet, Bernstein, Brassard and Vazirani (1997) proved that Grover's algorithm was actually optimal.

7. Shor's and Grover's algorithms strengthens Feynman's hypothesis that quantum computers are indeed more powerful than Universal Turing machines (even probabilistic).

8. It is also known now that $\mathbf{P} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}$. Hence if it is proved that quantum computers are strictly more powerful than classical computers then it would follow that $\mathbf{P} \neq \mathbf{PSPACE}$. This would be great because computer scientists have been trying to prove this for a long time now. Looking at it pessimistically, this may take some doing. After all, if the classical complexity theory community could not find a proof in all these years, it may not be easy to show that $\mathbf{P} \subset \mathbf{BQP}$.

   If, on the other hand, it turns out that quantum computers are no more powerful than the classical ones, then there would be little reason (other than technological ones) to do algorithm design using quantum mechanical principles.

4.2 Technology

Ever since Shockley's transistor, hardware development has happened at a phenomenal pace. In the early 90's most of us were using 33 MHz PC's with 20 MB hard-disks. In sharp contrast, today most of us have desktops with 3 GHz CPU's, a couple of GB of RAM and $>$ 80 GB hard-disks. This development has been codified as the Moore's law, which states that *computer power will roughly double for constant cost roughly once every two years.*

Amazingly, Moore's law has approximately held true till now. However, this dream run may soon end because conventional methods of fabrication of computer technology are running in to problems of size, and quantum effects are beginning to interfere in the functioning of electronic devices as they are made smaller. An obvious solution is to switch over to computing using quantum mechanical principles.

4.3 Taste

Finally, there may be a reason of taste and belief.

The analytical tools used by Physicists and Engineers to model natural and engineering systems appear to be based on theories like: thermodynamics, entropy and information theory, quantum mechanics, Fourier (and other similar) transforms, ... In contrast, traditional CS appears to be largely based on formal logic and combinatorics. Quantum computation may turn out to be one way to reconcile! Consider the following quote:

"All of this will lead to theories [of computation] which are much less rigidly of an all-or-none nature than past and present formal logic. They will be of a much less combinatorial, and much more analytical, character. In fact, there are numerous indications to make us believe that this

new system of formal logic will move closer to another discipline that has been little linked in the past with logic. This is thermodynamics, primarily in the form it was received from Boltzmann, and is that part of theoretical physics which comes nearest in some of its aspects to manipulating and measuring information"

- John Von Neumann, *Collected Works*, Vol. 5, pg. 304.