Ph/CS 219A
Quantum Computation

Lecture 10. Randomized, Reversible, and Quantum Computation

Last time we discussed classical circuits that compute Boolean functions. We considered the notion of universal gates, and the idea of uniform circuit families for the purpose of classifying the computational hardness of a language. We defined the complexity classes P, NP, co-NP, and explained the notion of a reduction and of NP-completeness.

Today we continue the discussion of classical computation, introducing the randomized and reversible computational models, and we'll begin our study of quantum computing.

See Chapter 5 of the Lecture Notes.

Randomized Computation

Suppose our computer has access to a random number generator. For each gate in the circuit, we flip coins to choose from among several possible operations. Even when the input is fixed, the circuit samples from many possible computational paths.

We won't get the same answer every time we run the circuit on input *x*, but our computation is useful if the probability of getting the right answer is high enough. It is good enough if

$$\text{Prob(right answer)} \geq 1/2 + \delta,$$

where δ is a constant independent of the input size. We can *amplify* the success probability by running the algorithm multiple times, and taking a majority vote on the outcomes.

In N trials there are 2^N possible sequence of outcomes. A majority vote gives the wrong answer only if the number N_W of wrong outcomes is greater than $N/2$. A particular sequence of outcomes with $N_W > N/2$ wrong answers occurs with probability

$$\left(\frac{1}{2}-\delta\right)^{N_W}\left(\frac{1}{2}+\delta\right)^{N-N_W} \leq \left(\frac{1}{2}-\delta\right)^{N/2}\left(\frac{1}{2}+\delta\right)^{N/2} = \frac{1}{2^N}\left(1-4\delta^2\right)^{N/2} \leq \frac{1}{2^N}\left(e^{-4\delta^2}\right)^{N/2} \Rightarrow \text{Prob(wrong majority)} \leq e^{-2N\delta^2}.$$

So probability of error is less than ε if we run the computation N times, where $N \geq \frac{1}{2\delta^2}\ln\left(1/\varepsilon\right)$.

The number of runs N does not depend on the input size if δ is a nonzero constant. A standard but arbitrary convention is to consider δ=1/6.

BPP and MA

The number of runs *N* does not depend on the input size if δ is a nonzero constant. A standard but arbitrary convention is to consider =1/6. That is, we say a randomized uniform circuit family decides a language *L* if

$$\text{Prob}(\text{accept } x \in L) \geq 2/3, \quad \text{Prob}(\text{accept } x \notin L) \leq 1/3.$$

$$\text{BPP} = \{\text{languages decided by polynomial-size uniform randomized circuit families}\}$$

BPP = "bounded-error probabilistic polynomial time." It is obvious that BPP contains P; a deterministic circuit is a special case of a randomized circuit, in which the randomness is never consulted. It is widely believed, but unproven, that BPP=P, i.e. that randomness does not expand the class of problems we can solve efficiently.

There is also a randomized version of NP, called MA = "Merlin-Arthur." (All-powerful Merlin provides a proof which mortal Arthur verifies.)

A language *L* is in MA if and only if there is a polynomial-size *randomized* verifier *V(x,y)* such that

If $x \in L$, then there exists *y* such that Prob(*V(x,y)*=1) $\geq$ 2/3 (completeness),
If $x \notin L$, then, for all *y*, Prob(*V(x,y)*= 1) $\leq$ 1/3 (soundness).

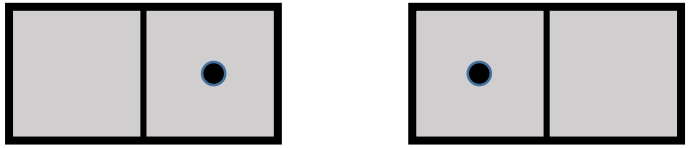
Obviously MA contains BPP (no certificate) and NP (no randomness).

Reversible Computation

The reversible model brings us a step closer to the quantum model, and is interesting in its own right.

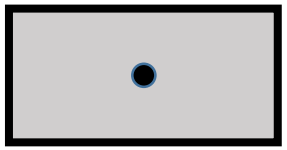
Landauer's principle: Erasing a bit at temperature T requires work $W \geq kT \log 2$ (k = Boltzmann's constant).

Why? Consider a one molecule gas in a box as a one-bit memory. Erasure is a process which maps 0→0 and 1→0, requiring no knowledge about the initial state of the memory.

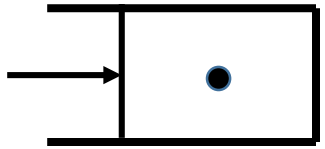


Example of a physical process that erases:

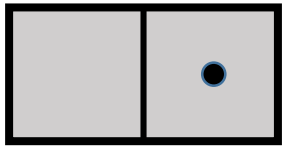
1) Remove partition.



2) Isothermal compression.



3) Replace partition.



First law of thermodynamics: $W = T \Delta S$, where $S = k \log$(number of microstates)
Number of microstates reduced by a factor of 2 → $W = T \Delta S = kT \log 2$.

The key point is that erasure always involves compression of the memory's phase space (dissipation), which necessarily requires flow of heat from the memory to its environment. Erasure carries an unavoidable thermodynamic cost.

Reversible Computation

Landauer's principle: Erasing a bit at temperature T requires work $W \geq kT \log 2$ (k = Boltzmann's constant).

Gates in a circuit which map two input bits to one output bit also reduce phase space and are therefore subject to Landauer's principle.

Example: the AND gate: 00, 01, 10 → 0, 11→ 1. Information about input is lost if output is 0. We say that the gate is *logically irreversible*. If the input bits are uniformly distributed, then the amount of lost info is

$\frac{3}{4}\log_2 3 = 1.19$ bits.

Therefore, Landauer argued, computation, like erasure, has an unavoidable thermodynamic cost per gate.

Is this why you feel heat being dissipated when you rest your computer on your lap?

Note that at 300K, $kT = 4 \times 10^{-21}$ Joules. For 3 billion transistors on a CPU and a clock speed of 3 GHz, fewer than 10^{19} bit ops per second (an overestimate for various reasons, but an upper bound), so Landauer says we need power at least 40 milliwatts. Actually, your laptop is burning about 100 W.

Heat dissipation is a serious constraint on today's computing technology (we don't want the chip to melt), but currently the fundamental thermodynamic limit it not much of a limitation. Maybe some day. Anyway, Landauer was *wrong*. A *reversible* computer can simulate the circuit model efficiently. Erasure is not necessary for computation.

Reversible Computation

A *reversible* computer can simulate a Boolean circuit efficiently. In principle we can compute for free. (From a thermodynamic viewpoint, erasure is hard, but computing is easy.)

A reversible computer evaluates invertible functions taking *n* bits to *n* bits. $f : \{0,1\}^n \to \{0,1\}^n$
Because the function is 1-to-1, it permutes the 2^n bit strings of length *n*.
The number of such permutations is

$(2^n)! \approx (2^n / e)^{2^n}$ (Stirling approximation). A tiny fraction of the $\left(2^{2^n}\right)^n = \left(2^n\right)^{2^n}$ functions taking *n* bits to *n* bits.

An invertible function can encode a Boolean function. $\tilde{f}(x, y) = (x, y \oplus f(x))$.
Here *x* is *n* bits and *y, f(x)* are each one bit. XOR = $\oplus$ is addition mod 2.

Are there universal reversible gates? That is, simple operations that we can compose to construct any invertible function? There are, but 2 → 2 bit gates do not suffice. We need 3→3 bit gates for universality.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = M \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix}; \quad M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

These account for all 4!= 24 reversible 2→2 gates. All are linear over the binary field. We need nonlinear operations to achieve multiplication, e.g. the AND gate.

Reversible Computation

2 → 2 bit gates do not suffice for universal reversible computing. We need 3→3 bit gates.

Toffoli gate. It is reversible and can be used to build up arbitrary permutations of n-bit strings (see the Chapter 5 Lecture Notes).

We can see that it simulates NOT, AND, though it produces some "junk" we need to discard. If we set x=y=1, the Toffoli gate flips z.

If we set z=0, then the Toffoli gate deposits $x \wedge y$ in the third register. And $x \vee y = \neg [(\neg x) \wedge (\neg y)]$.

Also useful to note: if we set x = 1, then the Toffoli becomes the (linear) controlled-NOT (CNOT) gate $(y,z) \rightarrow (y, y \oplus z)$. In particular for $z=0$, the CNOT gate *copies* the bit y into the third register.

Landauer might still object. It we simulate AND gates using Toffoli gates, we generate junk in each step. Eventually we will run out of memory and will need to erase all that junk, finally paying the price for the gates we have executed.

But the erasure can be avoided. We simulate the circuit using Toffoli gates, copy the final output bit, then run the circuit backward to return our memory to its pristine initial state! In effect that doubles the circuit size, but that is not such a hefty price to pay, and in fact we can reduce that cost substantially. (See notes.)

Quantum Circuits

A more powerful computational model (or so we believe).

1) Qubits $\mathcal{H}_n = (\mathcal{H}_2)^{\otimes n} = \text{span}\{|x\rangle, \quad x = 0,1,2,\ldots,2^n - 1\}.$ Preferred decomposition into small subsystems, because "physics is local."

2) Initialization $|000\ldots0\rangle.$ We can cool a register close to absolute zero, relatively easily.

3) Universal set of unitary quantum gates $\{U_1, U_2, \ldots, U_{n_G}\}$.

Finite instruction set. Each acts on a constant number of qubits (e.g. two). *Universal* means we can approximate any *n*-qubit unitary to high accuracy.

4) Classical control. A classical computer builds a circuit and directs its execution.

5) Readout of one or more qubits in the standard basis $\{|0\rangle, |1\rangle\}$. Hence the quantum model is a randomized computational model. (Measurements can be delayed until the end of the computation.)

This model (1)--(5) can be simulated by a randomized classical computer, but not *efficiently.* Reversible classical computation is a special case, because permutations of basis states are unitary. Randomized computation is a special case, because we can flip a coin by measuring an *X* eigenstate in the *Z* basis.

There is a quantum analog of BPP: BQP = bounded-error quantum polynomial time.

$$\text{BQP} = \{\text{languages decided by polynomial-size uniform quantum circuit families}\}$$

Quantum Circuits

There is a quantum analog of BPP: BQP = bounded-error quantum polynomial time.

$$\mathrm{BQP} = \{\text{languages decided by polynomial-size uniform quantum circuit families}\}$$

Evidently, BQP contains BPP, but we hope it is larger!

There is also a quantum analog of MA, called QMA = quantum Merlin-Arthur, in which the verifier is a uniform family of poly-size quantum circuits, and the witness is a quantum state. Evidently QMA contains MA, but we expect it is larger. We may also define a class QCMA = quantum-classical Merlin-Arthur, where the verifier is quantum but the witness is classical. Then $\mathrm{BPP} \subseteq \mathrm{BQP} \subseteq \mathrm{QCMA} \subseteq \mathrm{QMA}.$

The quantum model raises many questions to be addressed in the next few lectures.

-- How do we construct universal quantum gate sets, and what is the cost of simulating one set of universal gates with another?

-- How hard is it to simulate the quantum circuit model using the classical circuit model?

-- How much error can we tolerate in approximating the ideal quantum gates of the model by realistic quantum gates using actual hardware?

-- How large a quantum circuit is needed to approximate accurately a generic unitary transformation?