Lecture notes

January 8, 2024

1 Lecture 1

Logistics. There will be 4 homework assignments. All homework will be announced during lectures and released on the course website: https://wdaochen.com/teaching.html. The first homework will be shorter and released on Friday 12th January. It will be due by the drop deadline of 22nd January. All assessment will be via homework.

Office hours: Fridays 2:00pm - 3:00pm, ICICS X553.

TA: Xingyu Zhou (away this week, will be grading homework).

Brief introduction. Quantum computing is computation using the laws of quantum mechanics that were discovered in the early 20th century. Sometimes the power of quantum computing is described as: n quantum bits or qubits can be in the 2^n states of n bits at the same time. Therefore, it can solve a problem by simply trying 2^n, i.e., a large number for n large, possible solutions at the same time. But this is explanation is not really correct. In fact, employing the same logic, we may also say that n classical bits can be in 2^n states at the same time in a classical randomized computation where the n bits are determined by n random coin flips. In fact, quantum computation is similar to randomized computation in many ways. However, we will see that it subsumes randomized computation and can be provably more powerful than randomized computation in the so-called "query model" of computation. Much of this course will focus on the query model, which is *not* the same as the standard model of computation, known as the Turing model. We do this for two reasons:

1. Quantum-over-classical speedups (or simply quantum speedups) can be proven rigorously in the query model. In contrast, there is no rigorous proof of quantum speedup in the Turing model. This is because it is notoriously difficult to prove classical lower bounds in the Turing model.

2. Quantum speedups in the query model often translates into "apparent speedups" in the Turing model, where "apparent speedups" means quantum algorithms that's faster than any *known* classical algorithm. For example, the apparent quantum speedup for factoring discovered by Peter Shor was motivated by a quantum speedup in the query model for the so-called Simon's problem.

Let's get started! An alphabet is a finite non-empty set. $\mathbb{N}$ denotes the positive integers (no zero).
Query complexity: Let $n, m \in \mathbb{N}$, $\Sigma := \{0, 1, \ldots, m-1\}$, Γ be an alphabet, and

$$f \colon D \subseteq \Sigma^n \to \Gamma, \tag{1}$$

Mainly deal with case where $\Gamma = \{0, 1\}$.

Remark 1. *(Remarks are informal.)*

1. *Main question of query complexity: given $x \in D$, how many bits of x need to be read (queried) to compute f? (At most n.)*

2. *Three models of query computation: deterministic $D(f)$, randomized $R(f)$, quantum $Q(f)$. Quantum speedup means $Q(f) < R(f)$. Note that this means to establish a quantum speedup, we not only need to give a quantum algorithm but a classical lower bound. Therefore, a portion of this class will be devoted to* classical *analysis.*

3. *Comment: Say this only if someone asks. I think the point is easier to appreciate once the students have become more familiar with the query model. The access to the input $x \in D$ is different in the Turing model. In the Turing model, x would be the classical input to a (quantum) circuit and time complexity is the time needed to generate that circuit. The $x \in \{0,1\}^n$ in the query model more appropriately translates to a function $x \colon [n] \to \{0,1\}$ whose circuit can be contructed using the input y in the Turing model. Hopefully this makes more sense when we discuss how the query speedup for OR translates to an apparent speedup for 3SAT. (Note that in the Turing machine model, the output of the Turing machine on 1^λ is the* (quantum) circuit *that computes the answer to the problem on all input $y \in \{0,1\}^\lambda$.)*

Example 1. *1. $f \colon \{0,1\}^3 \to \{0,1\}$, $f(x) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$. Intuitively, how many bits of the input $x \in \{0,1\}$ do we need to read to determine f. Could always read all 3 bits but here there's a cleverer way. Read the bit x_1, then read x_2 if $x_1 = 1$ and x_3 if $x_1 = 0$.*

2. $f = \mathrm{OR}_n \colon \{0,1\}^n \to \{0,1\}$

$$\mathrm{OR}_n(x) = x_1 \vee x_2 \vee \cdots \vee x_n. \tag{2}$$

Computing OR_n *models search among* n *elements since computing* $\mathrm{OR}_n(x)$ *is equivalent to finding a* 1 *among the* n *bits of* x*. Facts:*

(a) $D(\mathrm{OR}_n) = R(\mathrm{OR}_n) = \Theta(n)$ [1] *Intuition: for a "worst-case" input* x*, you might query the first* $n-1$ *bits and they all show up as* 0 *so still need to made one more query to devide whether* $\mathrm{OR}(x) = 0$ *or* $\mathrm{OR}(x) = 1$*.*

(b) $Q(\mathrm{OR}_n) = \Theta(\sqrt{n})$ *– due to Grover search, will see later.*

(Note that computing OR_{2^n} *relates to solving SAT on* n *Boolean variables since latter is about seach for a satisfying assignment among* 2^n *possible assignments.)*

The definitions of query complexity are dependent on the domain of the f under consideration, i.e., they are dependent on D, m, and n (left implicit).

Definition 1 (Deterministic decision tree (or query algorithm))**.** *Comment: draw figure A deterministic decision tree is a* m*-ary tree* T *with a unique vertex labelled as "root", together with the following additional data:*

1. *Each leaf*[2] *of* T *is labelled by an element in* Γ*.*
2. *Each non-leaf vertex of* T *is labelled by an element in* $[n] \coloneqq \{1, 2, \ldots, n\}$*.*
3. *For all non-leaf vertices* v*, the* m *edges between* v *and its* m *children are each labelled by a unique element in* $\{0, 1, \ldots, m-1\}$*. (For each non-leaf vertex of* T*, its neighbors going away from the root are known as its children. Each non-leaf vertex of* T *has exactly* m *children.)*

Definition 2 (Deterministic query computation)**.** *Let* T *be a deterministic decision tree and* $x \in D$*. We write* $T(x) \in \{0,1\}$ *for the bit output by the following procedure*

> *Set* v_{current} *to be the root vertex. Then, repeat the following until the label of a leaf is output:*
>
> 1. *If* v_{current} *is a leaf then output its label.*
> 2. *Otherwise, let* $i \in [n]$ *be the label of* v_{current} *and let* v *be the child of* v_{current} *such that the edge* $\{v_{\text{current}}, v\}$ *is labelled by* x_i*. Set* $v_{\text{current}} = v$*.*

We say that a deterministic decision tree T *computes* f *if*

$$\forall x \in D,\, T(x) = f(x). \tag{3}$$

[The "for all" quantifier means this is sometimes referred to as "T *computes* f *in the worst case".]*

Definition 3 (Deterministic query complexity)**.** *Given a deterministic decision tree (DDT)* T*, its depth* $\mathrm{depth}(T)$ *is the maximum length of a root-to-leaf path in* T*. Then*

$$D(f) \coloneqq \min_{T \text{ DDT},\ T \text{ computes } f} \mathrm{depth}(T) \tag{4}$$

Definition 4 (Randomized decision tree (or query algorithm))**.** *A randomized decision tree is a probability distribution* $\mathcal{T}$ *over deterministic decision tree.*

Definition 5 (Randomized query computation)**.** *Given* $x \in D$ *and a randomized decision tree* $\mathcal{T}$*, we write* $\mathcal{T}(x)$ *for the random variable on* $\{0,1\}$ *defined by: for all* $i \in \Gamma$*,*

$$\Pr[\mathcal{T}(x) = i] \coloneqq \Pr[T(x) = i \mid T \leftarrow \mathcal{T}]. \tag{5}$$

Let $\epsilon \in (0, 1/2)$*. We say that a randomized decision tree* $\mathcal{T}$ *computes* f *with (two-sided) bounded-error* ϵ *if*

$$\forall x \in D,\ \Pr[\mathcal{T}(x) = f(x)] \geq 1 - \epsilon \tag{6}$$

Note that

$$\Pr[\mathcal{T}(x) = f(x)] = \Pr[T(x) = f(x) \mid T \leftarrow \mathcal{T}] = \sum_T \Pr[T \mid T \leftarrow \mathcal{T}] \cdot \mathbb{1}[T(x) = f(x)] \tag{7}$$

[1] Asymptotic notation: $f(n) = \Theta(n)$ means there exists constants $0 < c_1 < c_2$ and $n_0 \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ s.t. $n > n_0$: $c_1 n \leq f(n) \leq c_2 n$.
[2] A leaf of T is a vertex of degree 1.

Definition 6 (Randomized query complexity)**.** *Given a randomized decision tree (RDT) T, its depth is defined by*

$$\mathrm{depth}(\mathcal{T}) \coloneqq \max\{\mathrm{depth}(T) \mid \Pr[T \mid T \leftarrow \mathcal{T}] > 0\}. \tag{8}$$

Then for $\epsilon \in (0, 1/2)$,

$$R_\epsilon(f) \coloneqq \min\{\mathrm{depth}(\mathcal{T}) \mid \mathcal{T} \text{ RDT}, \ \mathcal{T} \text{ computes } f \text{ with bounded-error } \epsilon\}. \tag{9}$$

Also standard to write

$$R(f) \coloneqq R_{1/3}(f). \tag{10}$$