Lecture 4: Decision tree complexity

Rajat Mittal

IIT Kanpur

This lecture note will introduce the first complexity measure coming from the compuational world. The complexity measure is called *decision tree complexity* and is arguably the simplest measure for a Boolean function. In particular, there are concrete techniques and results about lower bounds on this complexity measure (as opposed to circuit complexity).

We will introduce decision trees complexity measure, see some examples and construct lower bounds in the first section. The same measure can be studied in the randomized computational model as well as in quantum computing. In quantum computing, it is known as *quantum query complexity.* We will discuss the notion of algorithms in these computational paradigms and extend the definition of decision tree to these domains.

1 Deterministic decision trees

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. A *decision tree* (or a decision tree algorithm) for f, on input x, *queries* some subset of the bits of the input $(x_1, x_2, \cdots, x_n)$ and then outputs the value of $f(x)$. The k-th query of the algorithm can depend on the result of the first $k-1$ queries, this is called *adaptive* query model.

We can depict the algorithm as a binary tree, where each node contains the index queried and two children correspond to the query output being 0 or 1. The leaves denote the output of the function. For example, a decision tree for OR will look like,

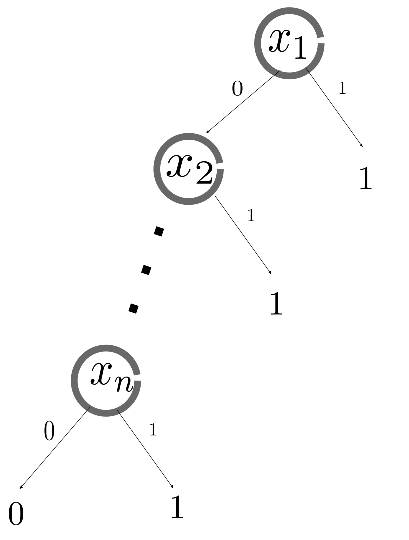

Fig. 1. A possible decision tree for OR.

The complexity of a decision tree is the depth of this tree. In the previous example, the complexity is n.

Exercise 1. Can there be more than one decision tree for a function? What about OR?

The deterministic decision tree complexity of a function f, denoted by $\mathrm{D}(f)$, is the minimum complexity of a decision tree computing f. We will look at different models in the next section and the term *deterministic* will be clear then.

Exercise 2. What is the maximum possible $\mathrm{D}(f)$ for any function f in terms of n?

We have already seen a decision tree for OR, its depth was n. Can there be a better decision tree, with complexity less than n?

We can easily construct an *adversary* argument to show that D(OR) is n. Suppose there was a decision tree, computing OR, with depth $n-1$; let us create an input where the decision tree will not output the correct value.

Look at the root node and assign it 0, follow the 0 branch and assign the next variable to be 0, keep following the path (assigning 0 to every variable) till we reach the leaf. Since the depth is less than n, there is a free variable (its value is not assigned). By fixing the value of that variable to be 0 or 1, the OR of the constructed input can be made 0 or 1. Though, the decision tree will give the same value for both of these fixings. In other words, the decision tree does not compute OR.

The adversary argument can be formalized. If any d chosen queries (even adaptively) can be answered in such a way that the function value is not determined, then d is a lower bound on the deterministic decision tree complexity. Let $l\mathrm{D}(f)$ be the optimal lower bound using this argument. This means, $l\mathrm{D}(f)$ is the maximum d such that any adaptively chosen d queries can be answered keeping the function value undetermined.

Exercise 3. Show that $l\mathrm{D}(f) \leq \mathrm{D}(f)$.

For $d = l\mathrm{D}(f) + 1$, there is a set of adaptive queries (of size at most d) which will determine the function value. These queries give a decision tree of depth d. So, the adversary argument is tight. For any $d < \mathrm{D}(f)$, any d adaptively chosen queries can be answered in such a way that the function value is not fixed.

Let us consider the decision tree complexity of Graph connectivity problem. For this problem, the input is a string of length $\binom{n}{2}$, indexed by edges, each bit specifying whether the corresponding edge is present or not. The Graph connectivity problem tell us whether the graph is connected or not. We will show that the determnistic tree complexity of this problem is $\binom{n}{2}$.

Exercise 4. What will the strategy for adversary?

You might want to keep the graph *barely connected*. The intution would be to answer 0 except if the tree becomes disconnected. Let us make it more formal. Say, at any point, not queried edges are blue and queried and kept are red. The goal is to keep the graph (with blue and red edges) barely connected. That means when a blue edge is queried, we will make it red if and only if the removal of it makes the graph disconnected.

In other words, the answer will be 0 if there is a path between the vertices of the queried edge using red and blue edges (ignoring the queried edge). From the strategy, if we include the last edge, the graph will be connected.

Exercise 5. Prove, using induction, if all edges are not queried then the red edges can't make a cycle.

The only thing remainig to be proven is that we don't get a spanning tree before the last edge. Suppose we do get a spanning tree and some blue edges are left. Let e be one of the blue edges, we know that e will create a cycle with the spanning tree.

Exercise 6. Show that the last edge added to the cycle before e should not have been added.

So, this problem also turns out to have full $\mathrm{D}(f)$ (equal to the number of variables). You might wonder if all problems have full $\mathrm{D}(f)$. You will show in the assignment that for addressing function on $n + 2^n$ bits, the deterministic decision tree complexity is $n + 1$.

Relation with degree: We will try to relate the two complexity measures seen, degree and deterministic decision tree. The easy direction is to show $\mathrm{D}(f)$ is greater than degree. Assume that f is from $\{0,1\}^n$ to $\{0,1\}$, this does not change the degree or $\mathrm{D}(f)$.

Exercise 7. Given a decision tree for a Boolean function $f : \{0,1\}^n \to \{0,1\}$, can you construct a polynomial representation for it?

The technique is very similar to interpolation (as done for truth table). We will create a indicator polynomial for every leaf. Summing up indicator polynomials for leaves labelled 1 will give us the polynomial. You will complete this argument in the assignment.

Exercise 8. Do you know a function where $\mathrm{D}(f) = \mathit{deg}(f)$?

In the other direction, it is known that $\mathrm{D}(f) = O(\mathit{deg}(f)^3)$. The proof of this will be covered later in the course. Though, this relation *need not* be tight. We only know a function f such that $\mathrm{D}(f) = \Omega(\mathit{deg}(f)^2)$, and nothing better.

The degree lower bound allows us to lower bound the deterministic tree complexity of composed functions. Clearly, $\mathrm{D}(f \circ g) \geq \mathit{def}(f \circ g)$.

Exercise 9. Show that

$$\mathit{def}(f \circ g) = \mathit{def}(f)\mathit{def}(g).$$

2 Randomized decision tree and quantum query complexity

The decision tree model can be extended to two different paradigms of computation, randomized and quantum. In both of these paradigms, the task is to come up with a *bounded error algorithm.* If you are not familiar, it is described in the next section.

2.1 Randomized and quantum algorithms

Most of the introductory algorithms you might have seen are all deterministic algorithms, e.g., sorting and network flows. That means, we *always* get the *exact* answer when the algorithm ends. The task there is to put a bound on the running time of such an algorithm.

We can relax algorithm's task a bit, with the input x, algorithm A can also use a random string (say $r \in \{0,1\}^k$ for some k chosen by algorithm). Hence, algorithm A is a function of the combined input x and r. A randomized/probabilistic algorithm is called to be successful, if for every x it outputs the correct answer *for most of the random strings r.*

Definition 1 (Bounded error randomized algorithm). *A randomized algorithm A accepts the function f in the bounded error model if,*

– *If $f(x) = 1$, then* $\Pr_r(A(x,r) = 1) \geq \frac{2}{3}$.
– *If $f(x) = 0$, then* $\Pr_r(A(x,r) = 0) \geq \frac{2}{3}$.

There are many ways to come up with a random string in the classical case, coin-toss, clock or many others. For the quantum computer, applying a Hadamard on $|0\rangle$ and measuring in standard basis works. In case of quantum computing, there need not be a random string, the probability will be taken over the inherent randomness of quantum states and measurements.

Definition 2 (Bounded error quantum algorithm). *An quantum algorithm A accepts the function $f : \{0,1\}^n \to \{0,1\}$ in the bounded error model if,*

– *If $f(x) = 1$, then* $\Pr(A(x) = 1) \geq \frac{2}{3}$.
– *If $f(x) = 0$, then* $\Pr(A(x) = 0) \geq \frac{2}{3}$.

Note that the probability is computed only over the random strings used in the algorithm. In other words, the probability condition is true for all inputs x (and not just on a high fraction of inputs).

For instance, if we just want our algorithm to work on large fraction of inputs, since number of primes are small, it is easy to come up with an algorithm for testing if a number is prime. We will not call such an algorithm a bounded error algorithm.

The constant $\frac{2}{3}$ in the definition is not important, any constant strictly greater than $\frac{1}{2}$ will work. In this case, we will repeat the algorithm k times and take the majority vote to decide the output. Suppose the original algorithm (the one we are repeating) gives the correct answer with probability more than $\frac{1}{2}+\epsilon$. We assume that ϵ is a constant. Then after repeating it k times, using Chernoff bound, the probability that we get the wrong answer is less than

$$e^{-2\epsilon^2 k}.$$

Exercise 10. Read about Chernoff bound.

Note 1. The number of times we need to repeat the algorithm, k, in bounded error algorithm is independent of size of input and only depends on probability we want to achieve. So, k will be a constant.

2.2 Decision tree/query model

Exercise 11. What should be the extension of decision tree in randomized computation model.

In a deterministic decision tree, once we query a bit, there are two possible queries in the next step (depeding on the query output). For the randomized case, we can decide on the bit to query depending upon the output of the previous queries and a random string. Finally the computation should answer correctly according to the bounded error model.

There is an easier way to think about a randomized decision tree.

Exercise 12. Using induction, show that a randomized decision tree is equivalent to having a probability distribution over the space of all decision trees.

Definition 3 (Randomized decision tree for f). *A randomized decision tree R is a probability distribution μ over the space of decision trees. R accepts the function f in the bounded error model if,*

$$\Pr_{D\ \mu}(D(x) = f(x)) \geq \frac{2}{3}.$$

The depth (complexity) of R is the depth of the largest decision tree with nonzero probability.

Let $\mathrm{cost}(R, x)$ be the expected number of queries on x by the randomized decision tree R. The cost of a randomized decision tree is the cost of R on the worst case input x.

Like in the case of deterministic decision tree complexity, the randomized decision tree complexity of an f, denoted $\mathrm{R}(f)$, is the minimum possible complexity of a randomized decision tree computing f.

$$\mathrm{R}(f) = \min_{\textit{randomized decision tree } R} \max_x \mathrm{cost}(R, x).$$

In many places the cost of R on x is defined to be the worst case length (over all decision trees with positive probability) of the path taken by the decision tree on x (instead of the expected path length defined by us). Since we are working with constant error (2/3), it is known that this cost do *not* give rise to a different complexity measure (both measures are related by constant).

In the quantum case, we allow quantum queries and accept the result in the bounded error model. Since quantum allows us to do queries in superposition, it does not give us a tree structure. So, we call such an algorithm a *quantum query algorithm.* The complexity of a quantum query algorithm is the number of queries done on the worst case input.

Define $\mathrm{cost}(Q, x)$ to be the maximum queries needed to compute x by a quantum query algorithm Q. The *quantum query complexity* of an f, denoted $\mathrm{Q}(f)$, is the minimum possible complexity of a quantum query algorithm computing f.

$$\mathrm{Q}(f) = \min_{\textit{algorithm } Q} \max_x \mathrm{cost}(Q, x).$$

3 Assignment

Exercise 13. Show that the decision tree of any nonconstant symmetric function is n.

Exercise 14. Show that $\mathrm{D}(\mathrm{ADDR}_m) = m + 1$.

Exercise 15. Show that $\mathrm{D}(f \circ g) \leq \mathrm{D}(f)\mathrm{D}(g)$.

Exercise 16. Show that $\mathrm{D}(f) \geq deg(f)$. This will be done by making a polynomial from a decision tree with degree at most the depth of the decision tree.