Boolean function complexity

Lecturer: *Nitin Saurabh* Meeting: 5
Scribe: *Nitin Saurabh* 22.05.2019

1 Decision tree complexity

One of the simplest model of computation is the *decision tree*. The goal here is to compute a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ using queries to the input.

Definition 1 (Decision Trees)**.** *A* decision tree *for* $f\colon \{0,1\}^n \to \{0,1\}$ *is a binary tree whose internal nodes are labelled by the variables* $x_1, \ldots, x_n$ *and two edges incident on a node are labelled by* 0 *and* 1*. The leaves of the tree are also labelled with* 0 *or* 1*.*

The computation on an unknown input $x = x_1x_2\cdots x_n$ *proceeds at each node by querying the input bit* x_i *indicated by the node's label. If* $x_i = 1$ *(resp.* 0*) the computation continue in the subtree reached by taking the edge labelled* 1 *(resp.* 0*). Thus on an input we follow a path in the tree starting from the root. The label of the leaf so reached is the value of the function on that particular input.*

The depth *of a decision tree is the length of a longest path from the root to a leaf.*

For an example see Fig. 1. It computes majority on 3 variables. We say that a decision tree computes f if it agrees with $f(x)$ for all $x \in \{0,1\}^n$. Clearly there are many different decision trees that compute the same function. The complexity of a decision tree is its depth, i.e., the number of queries made on the worst case input.

Definition 2 (Decision tree complexity)**.** *The decision tree complexity* $\mathsf{D}^{\mathsf{dt}}(f)$ *of a function* f *is the minimum depth of a decision tree computing* f*.*

Since knowing all the n-bits of an input uniquely identifies the input, we have for all Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$, $\mathsf{D}^{\mathsf{dt}}(f) \leq n$. We will be interested in proving tight complexity bounds in this model. Generally the lower bound arguments proceed via what is known as an *adversary argument*. We start with a simple example.

Proposition 3. $\mathsf{D}^{\mathsf{dt}}(\mathsf{OR}_n) \geq n$.

Proof. Let T be a decision tree claiming to solve OR_n. Think of an execution of T where an adversary answers the queries made by T. Since T claims to compute OR_n, the adversary chooses to respond with 0 for the first $n-1$ queries. Thus, the value of the n-th query determines whether the OR of the input bits is 0 or 1. Therefore, any decision tree computing OR_n must make n queries in the worst case. □

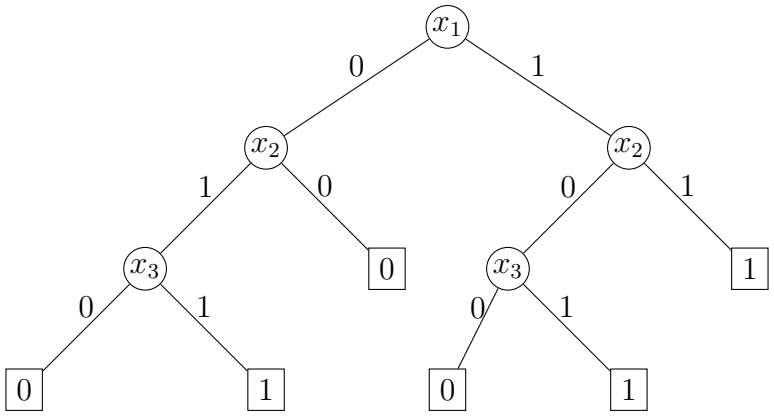

Figure 1: Decision tree computing majority on 3 variables

The high-level idea of an *adversary argument* is that an all-powerful malicious adversary pretends to choose a "hard" input. When the decision tree wants to query an input bit, the adversary sets that bit to whatever value will make the decision tree do the most work. If the decision tree doesn't query enough input bits before terminating, then there will be several different inputs, each consistent with the input bits already seen, that should result in different outputs. Whatever the decision tree outputs, the adversary can "reveal" an input that has all the queried input bits but contradicts the decision tree's output.

Given $x \in \{0,1\}^{\binom{n}{2}}$, we can associate a graph G_x on n vertices where an edge e is present iff $x_e = 1$. Consider the following indicator function for connectivity $\mathsf{Conn}_n \colon \{0,1\}^{\binom{n}{2}} \to \{0,1\}$,

$$\mathsf{Conn}_n(x) = 1 \text{ if and only if } G_x \text{ is connected.}$$

We start with an easy lower bound on its decision tree complexity.

Proposition 4. $\mathsf{D}^{\mathsf{dt}}(\mathsf{Conn}_n) \geq n^2/4$.

Proof. We want to give a strategy that forces a decision tree to query many edges before reaching an answer. We know that a graph is connected iff for every cut in the graph there exists at least one edge crossing the cut. Our strategy would be to force the decision tree to query every edge in a particular cut. To maximize the number of queries, consider a partition of $V = V_1 \dot\cup V_2$ such that $|V_1| = |V_2|$. Thus, the size of the cut is $n^2/4$.

The strategy is to reply with $x_e = 0$ for every edge in the cut until the very last edge and for every edge not in the cut reply with $x_e = 1$. Clearly the decision tree has to query the last edge in the cut to know whether the graph is connected or not. □

We now build on this idea to show that in fact it is not possible to improve on the trivial upper bound. This time our strategy would be to build a graph that is minimally connected, i.e., a spanning tree.

Theorem 5. $\mathsf{D}^{\mathsf{dt}}(\mathsf{Conn}_n) \geq \binom{n}{2}$.

Proof. Consider the strategy given as follows. We maintain connected components of the partial graph given by the queried edges. We start with n components, i.e., each node in different components. It captures the fact that no queries have been made yet. For each query we do the following.

Check if the queried edge belongs to the same connected component. If yes, then return 0. Otherwise, it connects two distinct components, say C_1 and C_2. Now check if it is the last edge to be queried in one of the two following cuts: (i) C_1 and $V \setminus C_1$, and (ii) C_2 and $V \setminus C_2$. If yes, then return 1, else return 0.

We claim that the partial graph maintained by the above strategy is a forest and furthermore, this forest does not turn into a single connected tree until the last edge is queried. It is easily seen that the theorem follows from the claim, since a query algorithm has to query all the edges before reaching a decision.

It is clear that the partial graph maintained is a forest since we never add an edge that is contained within a component; hence the graph maintained is acyclic. Moreover, whenever we add an edge we reduce the number of components by 1. Thus, the graph becomes connected when $(n-1)$-th edge is added to the partial graph.

We now prove that every other edge has been queried before the $(n-1)$-th edge is added. Suppose not. Then there exists an edge e that has not been queried and is not present in the spanning tree constructed after the $(n-1)$-th edge is added. Adding the edge e to the spanning tree creates a unique cycle containing that edge. Let e' be another edge on this cycle that was the last edge, among the edges on the cycle, to be added to the tree. It is now easily seen that when e' was being added to the tree, e was also an edge present in the cut that led to the inclusion of e'. Thus, it contradicts the choice of e' being the last edge to be queried in that cut. □

Definition 6. *A Boolean function* $f\colon \{0,1\}^n \to \{0,1\}$ *is called* evasive *if* $\mathsf{D}^{\mathsf{dt}}(f) = n$.

Rephrasing the previous theorem, we see that Conn_n is an evasive function.

2 Certificate complexity

Let $a \in \{0,1\}^n$. How many input bits of a one must query in order to ascertain the function value $f(a)$? In the previous section we saw that $\mathsf{D}^{\mathsf{dt}}(f)$ bits suffices to know the value at any input. However, in a decision tree, queries are constrained in some sense, for example, the variable at the root is queried on every input. The question therefore now is: *Can we do better if we relax this and allow for each input to choose its own smallest set of bits to be queried?* This leads to the notion of *certificate complexity.*

Definition 7. *A* certificate *for a Boolean function f is a subset $S \subseteq [n]$ of variables with an assignment $\alpha\colon S \to \{0,1\}$ such that for all $x, y \in \{0,1\}^n$ with $x|_S = y|_S = \alpha$, $f(x) = f(y)$. The size of a certificate is $|S|$. A certificate is called a* 1*-certificate if the function value is* 1 *on all inputs consistent with the certificate. Similarly, when the function value is* 0*, its called a* 0*-certificate.*

The certificate complexity of f at x, denoted $\mathsf{Cert}(f,x)$, is the size of the smallest certificate consistent with x.

The certificate complexity of f, denoted $\mathsf{Cert}(f)$, is defined to be $\max_x \mathsf{Cert}(f,x)$.

The 1-certificate complexity of f, denoted $\mathsf{Cert}^1(f)$, is defined to be $\max_{x:\ f(x)=1} \mathsf{Cert}(f,x)$.

The 0-certificate complexity of f, denoted $\mathsf{Cert}^0(f)$, is defined to be $\max_{x:\ f(x)=0} \mathsf{Cert}(f,x)$.

For example, $\mathsf{Cert}(\mathsf{AND}_n) = n$, $\mathsf{Cert}^0(\mathsf{AND}_n) = 1$, and $\mathsf{Cert}^1(\mathsf{AND}_n) = n$. The following proposition is an easy observation now.

Proposition 8. *Let $f\colon \{0,1\}^n \to \{0,1\}$. Then, for all $x \in \{0,1\}^n$, $\mathsf{Cert}(f,x) \leq \mathsf{D}^{\mathsf{dt}}(f)$. Therefore, $\mathsf{Cert}(f) \leq \mathsf{D}^{\mathsf{dt}}(f)$.*

Remark 2.1. 1*-certificate complexity can be viewed as a* non-deterministic *analog of (deterministic) decision trees. Similarly,* 0*-certificate complexity can be viewed as a* co-non-deterministic *analog of deterministic decision trees.*

A natural question now is how the decision tree and certificate complexities are related. We saw that $\mathsf{Cert}(f) \leq \mathsf{D}^{\mathsf{dt}}(f)$, but can decision tree depth be exponentially larger than the certificate complexity? Consider the following example,

$$\mathsf{Tribes}_{\sqrt{n},\sqrt{n}} := \bigvee_{i=1}^{\sqrt{n}} \left(\bigwedge_{j=1}^{\sqrt{n}} x_{ij} \right). \tag{1}$$

It is a read-once DNF (every variable appears exactly once) defined on n variables. It belongs to a class of functions known as *Tribes* functions. It is easily seen that $\mathsf{Cert}^1(\mathsf{Tribes}_{\sqrt{n},\sqrt{n}}) = \mathsf{Cert}^0(\mathsf{Tribes}_{\sqrt{n},\sqrt{n}}) = \sqrt{n}$, but $\mathsf{D}^{\mathsf{dt}}(\mathsf{Tribes}_{\sqrt{n},\sqrt{n}}) = n$. So this example shows a quadratic gap between certificate complexity and decision tree depth.

Theorem 9. *Let $f\colon \{0,1\}^n \to \{0,1\}$. Then, $\mathsf{D}^{\mathsf{dt}}(f) \leq \mathsf{Cert}^0(f) \cdot \mathsf{Cert}^1(f)$.*

Proof. We provide a decision tree algorithm that will decide f in $\mathsf{Cert}^0(f) \cdot \mathsf{Cert}^1(f)$ queries. The algorithm exploits the following crucial property of 0-certificates and 1-certificates.

Claim 2.1. *Let (S,α) be a 1-certificate and (T,β) be a 0-certificate for f. Then $S \cap T \neq \emptyset$. Furthermore, $\exists\ i \in S \cap T$ such that $\alpha(i) \neq \beta(i)$.*

Proof. Suppose they are not contradicting on one of the intersecting variables. Then we can make an input that is consistent with both (S,α) and (T,β), thereby contradicting the fact these are certificates for f. □

We now describe the decision tree algorithm to compute $f(x)$. It maintains a set $\mathcal{X} \subseteq \{0,1\}^n$ consisting of all inputs that are consistent with the replies to queries made so far. Initially $\mathcal{X} = \{0,1\}^n$.

1. Repeat the following $\mathsf{Cert}^1(f)$ times:
   If the function is constant on $\mathcal{X}$, then return this value and stop. Otherwise, pick a 0-certificate consistent with queries made so far and query all the variables in this certificate. If the queried values agree with the assignment given by the certificate then return 0 and stop. If not, then prune $\mathcal{X}$ to be the remaining set of inputs consistent with the answers to the queried variables.

2. Pick a $y \in \mathcal{X}$ and return $f(y)$.

It is clear that the algorithm queries at most $\mathsf{Cert}^1(f) \cdot \mathsf{Cert}^0(f)$ variables, since the algorithm runs for $\mathsf{Cert}^1(f)$ times and each time it queries $\mathsf{Cert}^0(f)$ variables.

We now need to show its correctness. Again it is clear that if it outputs in Stage 1, then the output is correct. So we consider the case when it returns the answer in Stage 2. Using Claim 2.1, we see that after querying a 0-certificate in Stage 1 we reduce the number of unknown variables in every 1-certificate by 1. Therefore, after $\mathsf{Cert}^1(f)$ steps, we would have queried all possible variables appearing in all 1-certificates. Thus, at the end of the Stage 1, after $\mathsf{Cert}^1(f)$ steps, we know whether our input contains a 1-certificate or not. In other words, all the remaining inputs in $\mathcal{X}$ have the same function value. □

Since $\mathsf{Cert}^1(f)$ and $\mathsf{Cert}^0(f)$ can be viewed as non-deterministic decision tree complexities. Theorem 9 can also be viewed as showing "$\mathsf{P} = \mathsf{NP} \cap \text{co-}\mathsf{NP}$" for decision tree *depth*.

3 Sensitivity vs. Block sensitivity

Definition 10. *The* sensitivity *of a Boolean function f at an input $x \in \{0,1\}^n$, denoted $\mathsf{s}(f,x)$, is the number of neighbors y of x in the hypercube graph such that $f(y) \neq f(x)$. That is,*

$$\mathsf{s}(f,x) = |\{i \in [n] \mid f(x) \neq f(x^i)\}|,$$

where x^i denotes the input obtained from x by flipping the i-th bit while keeping every other bit fixed.

The sensitivity *of f, denoted $\mathsf{s}(f)$, is defined to be $\max_x \mathsf{s}(f,x)$.*

For example, $\mathsf{s}(\mathsf{OR}_n, 0^n) = n$, $\mathsf{s}(\mathsf{OR}_n, x) = 1 \ \forall x \in \{0,1\}^n \setminus 0^n$, and $\mathsf{s}(\mathsf{OR}_n) = n$. Also, $\mathsf{s}(\mathsf{Parity}_n) = n$.

Nisan generalized the definition of sensitivity to *block sensitivity* in order to characterize the complexity of computing functions on a CREW PRAM.

For an input $x \in \{0,1\}^n$ and $S \subseteq [n]$, let x^S denote the input obtained from x by flipping the bits indexed by S while leaving the other bits unchanged. We say that a set (block) S is a sensitive block for f at x if $f(x) \neq f(x^S)$.

Definition 11. *The* block sensitivity *of f at x, denoted $\mathsf{bs}(f,x)$, is the largest number t such that there exists t disjoint sets $S_1, \ldots, S_t \subseteq [n]$ such that $f(x^{S_i}) \neq f(x)$, for all $1 \leq i \leq t$.*

The block sensitivity *of f, denoted $\mathsf{bs}(f)$, is defined to be $\max_x \mathsf{bs}(f,x)$.*

In the definition of block sensitivity when the disjoint sets are all taken to be singleton sets, i.e., $|S_i| = 1$, we obtain sensitivity. Therefore, $\mathsf{s}(f, x) \leq \mathsf{bs}(f, x)$. In fact, we have the following chain of inequalities.

Proposition 12. *Let f be any Boolean function. Then,*

$$\mathsf{s}(f, x) \leq \mathsf{bs}(f, x) \leq \mathsf{Cert}(f, x) \leq \mathsf{D}^{\mathsf{dt}}(f).$$

Proof. We just saw the first inequality and third inequality is the Proposition 8. The second inequality follows from the observation that a certificate must query at least one variable from each sensitive block. □

The largest known gap between sensitivity and block sensitivity is quadratic.

Example 13 (Rubinstein 1995)**.** *We divide the set of n variables into $\sqrt{n}$ disjoint (consecutive) subsets of $\sqrt{n}$ variables each. That is, for $1 \leq j \leq \sqrt{n}$, $S_j = \{x_{1+(j-1)\sqrt{n}}, \ldots, x_{j\sqrt{n}}\}$.*

For $x \in \{0,1\}^n$, define $R(x) = 1$ if and only if there exists at least one subset S_j where two consecutive variables are set to 1 and other $\sqrt{n} - 2$ variables are set to 0.

Clearly we have

$$\mathsf{bs}(R) \geq \mathsf{bs}(R, 0^n) \geq n/2,$$

where the sensitive blocks at 0^n are given by $n/2$ mutually disjoint sets of two consecutive variables.

On the other hand, we have $\mathsf{s}(R, x) \leq \sqrt{n}$ for all $x \in R^{-1}(1)$ and $\mathsf{s}(R, x) \leq 2$ for all $x \in R^{-1}(0)$.

Therefore, $\mathsf{bs}(R) \geq \mathsf{s}(R)^2/2$, i.e., we have a quadratic gap between block sensitivity and sensitivity.

It is not known whether block sensitivity can be upper bounded by a polynomial in sensitivity. This is well known as *sensitivity conjecture*.

Conjecture 14 (Sensitivity conjecture)**.** *Does there exists a universal constant $c > 0$ such that for all Boolean functions f,*

$$\mathsf{bs}(f) = O(\mathsf{s}(f)^c)?$$