Boolean function complexity

Lecturer: *Nitin Saurabh* Meeting: 1
Scribe: *Nitin Saurabh* 17.04.2019

Let $\{0,1\}^* := \cup_{n\in\mathbb{N}}\{0,1\}^n$. A *Boolean* function $f(x_1,\ldots,x_n)$ over n variables is a mapping $f\colon \{0,1\}^n \to \{0,1\}$. Usually, we interpret 0 as False and 1 as True. Some simple examples are,

$$\begin{aligned}
\mathsf{OR}(x_1,\ldots,x_n) = 1 &\text{ iff } x_1 \vee x_2 \vee \cdots \vee x_n = 1 \text{ iff at least one of the } x_i \text{ equals 1,}\\
\mathsf{AND}(x_1,\ldots,x_n) = 1 &\text{ iff } x_1 \wedge x_2 \wedge \cdots \wedge x_n = 1 \text{ iff all } x_i\text{'s equal 1,}\\
\mathsf{MAJ}(x_1,\ldots,x_n) = 1 &\text{ iff } \sum_{i=1}^{n} x_i \geq n/2, \text{ and}\\
\mathsf{Parity}(x_1,\ldots,x_n) = 1 &\text{ iff } \sum_{i=1}^{n} x_i \equiv 1 \pmod 2.
\end{aligned}$$

In fact, any property can be encoded as a Boolean function. For example, consider the CLIQUE problem where given a graph G and an integer k, one asks if G contains a clique of size k.

To encode a graph on n vertices, we may define Boolean variables $x_e \in \{0,1\}$ for each possible edge e in G. Recall there are $\binom{n}{2}$ possible edges on a graph over n vertices. We interpret Boolean values of a variable x_e as whether the edge e is present in the graph or not. That is, $x_e = 1$ iff e is in the graph G. Thus, every input in $x \in \{0,1\}^{\binom{n}{2}}$ defines a graph G_x on n vertices. We can then define $\text{CLIQUE}_k^n\colon \{0,1\}^{\binom{n}{2}} \to \{0,1\}$,

$$\text{CLIQUE}_k^n(x_1,\ldots,x_{\binom{n}{2}}) = 1 \text{ iff the graph } G_x \text{ has a clique of size } k.$$

In general, corresponding to any property we can define a Boolean function f, $f(x) = 1$ iff the graph G_x has that property.

We typically study a family of Boolean functions $\{f_n\}_n$ where $f_n\colon \{0,1\}^n \to \{0,1\}$. This corresponds to studying a language $L \subseteq \{0,1\}^*$ such that $L_n := L \cap \{0,1\}^n = \{x \in \{0,1\}^n \mid f_n(x) = 1\}$.

1 Basic Definitions

In the following we represent OR over 2 bits by $\vee$, AND over 2 bits by $\wedge$, and the negation of a bit by $\neg$. Recall, $\neg x = 1 - x$ for $x \in \{0,1\}$.

Definition 1. *(De Morgan Circuits) A* De Morgan *circuit $\mathcal{C}$ over n variables $\{x_1,\ldots,x_n\}$ is a directed acyclic graph (DAG) with n sources (nodes with indegree zero) and one sink (node with outdegree zero). Further they satisfy the following properties :*

- *All nodes have indegree 0, 1, or 2.*
- Input *nodes, i.e., nodes with indegree 0, are labelled with variables* $\{x_1, \ldots, x_n\}$ *or constants* $\{0, 1\}$.
- *All non-input nodes are called* gates *and are labelled with one of* $\vee$, $\wedge$ *or* $\neg$. *Gates with indegree 1 are labelled with* $\neg$, *whereas gates with indegree 2 are labelled with* $\vee$ *or* $\wedge$.
- *The sink node, i.e., the gate with outdegree 0, is designated as the* output.

Remark 1.1. *Though we assume the circuit produces 1 bit of output; it is easy to generalize the definition to circuits with more than one bit of output.*

By the *fan-in* (resp. *fan-out*) of a gate we will mean the number of incoming (resp. outgoing) edges incident on that gate. There are two important measures associated with circuits, namely its *size* and *depth*.

Definition 2. *The* size *of a circuit is defined to be the number of* $\vee$ *and* $\wedge$ *gates that it contains. The* depth *of a circuit is the length of a longest path from an input gate to the output gate.*

We now define a restriction of circuits.

Definition 3. *(De Morgan Formulas) A* formula *is a circuit where every gate has fan-out at most 1. That is, the underlying undirected graph is a tree.*

The size *of a formula is defined to be the number of leaves in its tree, and the* depth *of a formula is the depth of its tree.*

The crucial difference between formulas and circuits is that of the restriction on fan-out. In the circuit model, a result computed at some gate can be reused many times by increasing the fan-out of the gate. However in formulas, we need to recompute a function if we wish to use its result again.

For example, we can compute Parity_n on n bits with a circuit of size $O(n)$. However we will prove in the following lectures that Parity_n requires a formula of size $\Omega(n^2)$.

Remark 1.2. *Sometimes the size of a circuit is also measured by the number of edges (or, wires) in the underlying graph. Note that this measure is only quadratically larger than the measure in Definition 2. Further, in the case of formulas it's only linearly large.*

The set of operations (or, gates) allowed in a Boolean circuit is given by a *basis*.

Definition 4. *A* basis *is a finite set consisting of Boolean functions.*

For example,

- De Morgan basis : $\{\vee, \wedge, \neg\}$
- Monotone basis : $\{\vee, \wedge\}$ (not universal)
- Full binary basis : all Boolean functions over 2 variables

We will mostly be working with De Morgan basis.

1.1 Uniformity vs. Non-uniformity

In a *uniform* model of computation, a single algorithm works for *all* inputs of arbitrary length, i.e., all inputs in $\{0,1\}^*$. For example, Turing Machines, Finite automata, etc.

On the other hand, in a *non-uniform* computational model, for every input length n, we have a single algorithm working on inputs of only that length. Algorithms across different input lengths might be completely different in their behaviour. For example, circuits, formulas, etc.

In particular, we say that a sequence of circuits $\{C_n\}_{n\in\mathbb{N}}$ decides a language $L \subseteq \{0,1\}^*$ if for all n, C_n computes f_n where $f_n \colon \{0,1\}^n \to \{0,1\}$ is such that $f_n(x) = 1 \Leftrightarrow x \in L$ for any $x \in \{0,1\}^n$.

Observe that the non-uniform model of computation is more powerful than the uniform model. Since we are allowed to use different circuits for different input lengths and at each input length there are only finitely many inputs, circuits can even compute undecidable languages.

Thus, it follows that lower bounds in the non-uniform model imply lower bounds in the uniform model. While the upper bounds in the uniform model imply upper bounds in the non-uniform model. To illustrate, we formally note that functions computed by Turing machines can be computed efficiently by Turing circuits.

Theorem 5. *Any Turing machine running in time $t(n)$ over inputs of length n can be simulated by a circuit of size $O(t(n)^2)$.*

An immediate and easy corollary is that proving super-polynomial lower bound on the circuit size of any language in NP implies $\mathsf{P} \neq \mathsf{NP}$.

In the following lectures we will see some polynomial lower bounds against De Morgan formulas. We now show that most Boolean functions are hard to compute.

2 Lower bounds

Let $\mathsf{L}(f)$ denote the minimal size of a De Morgan formula computing a Boolean function f. We define $\mathsf{L}(n) := \max_{f\colon \{0,1\}^n \to \{0,1\}} \mathsf{L}(f)$. That is, $\mathsf{L}(n)$ is the smallest number t such that every Boolean function on n variables can be computed by a formula of size at most t.

Theorem 6 (Riordan-Shannon (1942))**.** *For every constant $\varepsilon > 0$ and sufficiently large n,*

$$\mathsf{L}(n) \geq (1-\varepsilon)\frac{2^n}{\log n}.$$

Proof. The proof uses a simple *counting argument* :

- Count how many different Boolean functions on n variables can be computed by a formula of size t.
- Compare it with the total number 2^{2^n} of Boolean functions on n variables.

Without loss of generality we assume that all negations are only applied to the leaf nodes. We now count the number of different formulas of size at most t.

Observe that a formula is fixed given the structure of the underlying full binary tree and a labeling of the nodes in the tree. The number of full binary trees with t leaves is at most 4^{t-1}. (This number is given by the Catalan number.)

We now count in how many ways we can convert a full binary tree into a De Morgan formula. Since each leaf node can be labelled by a literal in the set $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ or constants in $\{0, 1\}$. The number of ways we can label the leaf nodes is at most $(2n + 2)^t$. The internal nodes can be labelled by one of $\vee$ or $\wedge$. Thus, there are 2^{t-1} ways to label the internal nodes. Therefore, the number of ways a tree can be converted into a De Morgan formula is at most $2^{t-1}(2n + 2)^t$.

Hence, the total number of different formulas of size at most t is at most

$$4^{t-1} \cdot 2^{t-1} \cdot (2n + 2)^t \leq (16n)^t.$$

Comparing the above number with 2^{2^n} gives the desired lower bound. $\square$

Remark 2.1. *We note that the proof not only says that there exists a hard function, but in fact* almost all *functions over n variables are hard. To observe this, note that $(16n)^t \ll 2^{2^n}$ when $t = 2^n/10 \log n$.*

Another point to remark is that the lower bound is asymptotically tight. Lupanov (1960) showed that $\mathsf{L}(n) \leq (1 + o(1))\frac{2^n}{\log n}$.

We now prove a similar result for circuits.

Let $\mathsf{C}(f)$ denote the minimal size of a De Morgan circuit computing a Boolean function f. We define $\mathsf{C}(n) := \max_{f\colon \{0,1\}^n \to \{0,1\}} \mathsf{C}(f)$. That is, $\mathsf{C}(n)$ is the smallest number t such that every Boolean function on n variables can be computed by a circuit of size at most t.

Theorem 7 (Shannon (1949))**.** *For every sufficiently large n, $\mathsf{C}(n) > 2^n/n$.*

Proof. Again the proof uses counting argument similar to the proof of Theorem 6. We start with counting the number of different circuits over n variables and of size at most t. Let $g_1, g_2, \ldots, g_t$ denote the gates of the circuit. To describe a circuit, it suffices to label each gate with one of $\vee$, $\wedge$, or $\neg$, and further describe the two incoming edges to them. Thus, the number of such descriptions is at most

$$\left(3\binom{t - 1 + 2n}{2}\right)^t \leq \left(3\frac{(t + 2n)^2}{2}\right)^t \leq 2^t(t + 2n)^{2t}.$$

We now assume, without loss of generality, that no two gates in the circuit computes the same function. Thus, permuting the labels of the t gates gives us a different description of a circuit computing the same Boolean function. Therefore, the total number of different Boolean functions computed by circuits of size at most t is at most

$$\frac{2^t(t + 2n)^{2t}}{t!} \leq \frac{2^t 3^t (t + 2n)^{2t}}{t^t} = (6t)^t \left(1 + \frac{2n}{t}\right)^{2t} \leq (6t)^t e^{4n}.$$

Setting $t = \frac{2^n}{n}$, we have

$$(6t)^t e^{4n} \leq 2^{2^n(1-\frac{\log n}{n})+3\frac{2^n}{n}+O(\log n)}.$$

Since there are 2^{2^n} distinct Boolean function on n variables, we obtain the desired lower bound. □

Remark 2.2. *Again we note that the proof shows that almost all Boolean functions over n variables require large circuits.*

In the next lecture we will see an asymptotically matching upper bound. That is, for every Boolean function f on n variables, $\mathsf{C}(f) \leq (1 + o(1))\frac{2^n}{n}$.