CS6840: Advanced Complexity Theory Mar 6, 2012

Lecture 35 : Size and Depth complexity of Boolean Circuits

Lecturer: Jayalal Sarma M.N. *Scribe: Dinesh K.*

THEME: Circuit Complexity
LECTURE PLAN: Notion of size and depth of circuits. Shannon's and Lupanov's bound on size of circuits computing any boolean function.

In the previous lecture, we saw what boolean functions, family of boolean functions are, defined function computation and argued that boolean circuits are natural notions for computing boolean functions. We also saw the notion of complete basis and characterisation of complete basis due Emil Post.

In this lecture, we shall see two important parameters of a circuit called *size complexity* and *depth complexity* and shall look into Lupanov's (upper) bound and Shannon's (lower) bound on the size of a circuit computing a function.

Before delving deep, let's have a look at the following definitions.

1 Definitions

Definition 1. (Family of circuits) Let $\{C_n\}_{n\geq 0}$ be an infinite collection where each c_i is a boolean circuit (computing a boolean function) on the i input lines.

This definition is similar to our notion of family of functions defined in the previous lecture.

Definition 2. Let $L \subseteq \Sigma^*$, $x \in \Sigma^*$. Let the characteristic function of L on inputs of length n be defined as ,

$$\chi_L^{(n)}(x) = \begin{cases} 1 & \text{if } x \in L \cap \{0,1\}^n, \\ 0 & \text{otherwise} \end{cases}$$

Definition 3. $L \subseteq \Sigma^*$ is computed [1] by a family of circuits if $\exists\{C_n\}_{n\geq 0}$ such that

$$x \in L \iff C_{|x|}(x) = 1$$

Essentially C_n computes the function $\chi_L^{(n)}(x)$

[1] When we say L is computed by a circuit family, we do not worry about how to get the circuit family. The issue of whether the family exists or not shall be addressed in the subsequent lectures.

Observe that the circuit computing a function on n input can be drastically different from a circuit computing the same function on $n+1$ inputs. This leads to the notion of *non-uniform* computation which can be stated informally as the case where there is no single circuit that decides the language, yet for strings of the same length there is a single device (circuit) that decides them.

2 Parameters of Interest

To understand what parameters of circuit are we interested in, let us try to gain some motivation from the parallel computation world. In parallel computation, the aim is to decompose the jobs and then each job is processed by one processor.

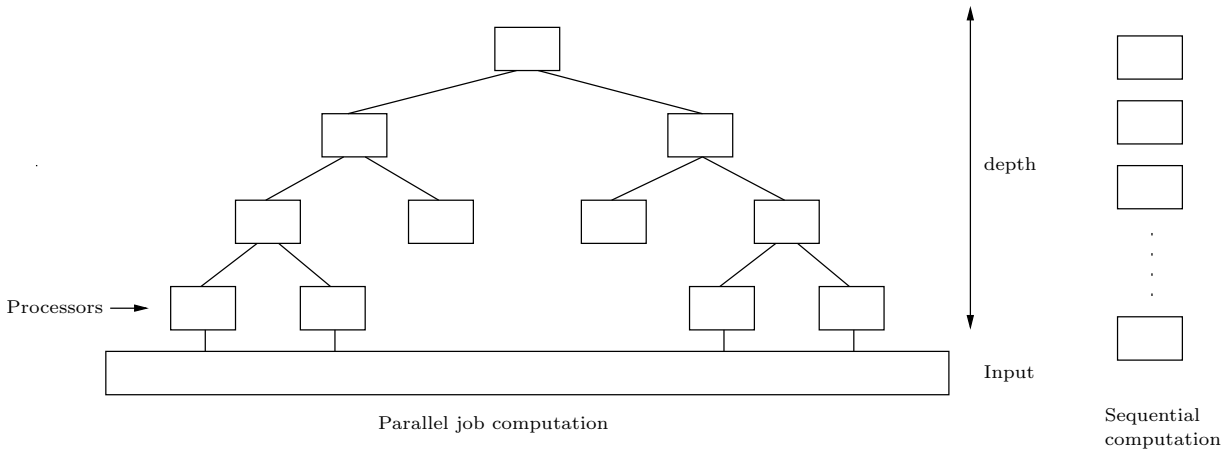

Figure 1: Parallel and Sequential computation

But there can be parallelism in the outcomes of these processors. So we again have a processors that receives the processed input and carry on the processing (figure 1) thereby leading levels of computation. In case of sequential computation, no such decomposition is possible : each processor in the same level will be executed one after the other and will be repeated for all the levels.

What would be the "parallel" time taken to finish the job ? This is nothing but the time required to get the output. Since input to each level is dependent on the output of the previous level, the delay caused will be proportional to the length of the longest path.

Circuits essentially tries to abstract this idea.

Definition 4. (Depth) Depth of any circuit C is a function $\mathcal{D}$ that takes in circuit and gives the length of the longest path from root (output) to any leaf (input).

Definition 5. (Size) Size of a circuit C is a function $\mathcal{S}$ that takes a circuit and gives the number of gates (or devices) in the circuit.

Note that the size of a circuit can also be thought as the time taken by a sequential machine to compute the same function.

Example 6. *Consider the parity function* $f : \{0,1\}^n \to \{0,1\}$ *where* $f(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$. *The circuit corresponding to this function is as shown in figure 2.*

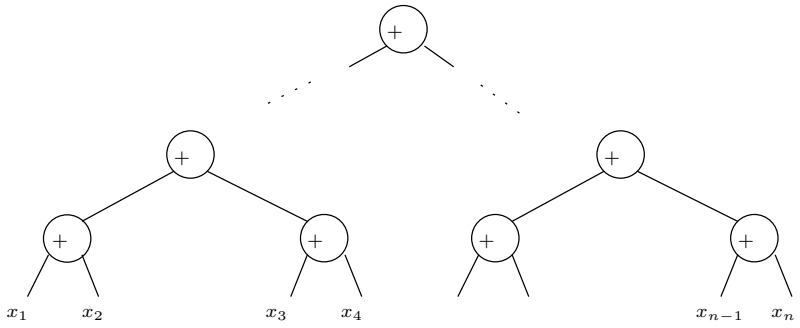

Figure 2: The parity function

Note that we have exploited the associativity property of $\oplus$. *This circuit has a size* $= n - 1$ *and depth* $= \lceil \log_2 n \rceil$.

The next natural question to ask is

Will the parameters $\mathcal{S}$ and $\mathcal{D}$ depend on the complete basis chosen ?

The answer is yes and we now give a more accurate definition of size and depth in terms of languages.

Definition 7. Size $S_\Omega(L)$ of a language L is the minimum sized[2] function of a circuit family that computes L for the complete basis Ω.

Definition 8. Depth $D_\Omega(L)$ of a language L is the minimum depth function of a circuit family that computes L for the complete basis Ω.

Definition 9. Circuit Complexity of a language L for a complete basis Ω is defined as the tuple $(\mathcal{S}_\Omega(L), \mathcal{D}_\Omega(L))$ which corresponds to the size and depth complexities respectively.

Claim 10. *Let* Ω, Ω' *be two finite complete basis of boolean functions of fixed arity. Then for any* $L \in \Sigma^*$,

$$\mathcal{S}_\Omega(L) = \Theta(\mathcal{S}_{\Omega'}(L)), \mathcal{D}_\Omega(L) = \Theta(\mathcal{D}_{\Omega'}(L))$$

[2]By minimum sized, we mean asymptotic size i.e. constant factors are ignored

Proof Idea Let $\{C_n\}_{n\geq 0}$ be the minimum sized circuit computing L for the complete basis Ω. When the basis is changed, one need to express boolean functions in Ω in terms of boolean functions in Ω'. This is always possible since Ω' is complete. Now a circuit in the basis Ω' can be obtained by replacing the gates in Ω by their equivalent circuits in Ω'.

Note that the basis is finite and is independent of n. So the blow up or shrinkage of each gate C_n will only be by a constant. Hence in the worst case all the gates in $\{C_n\}$ gets scaled by a constant factor and the size of the resultant circuit can be at most $k\times$ size of original circuit. In case of depth, the length of the longest path gets scaled up (or down) by a factor independent of n. □

Now we define the complexity class corresponding to the circuits computing a function.

Definition 11. For functions $f, g : \mathbb{N} \to \mathbb{N}$

$$SIZE(f(n)) = \{L | L \text{ is computed by a family of circuits of size } f(n)\}$$
$$DEPTH(g(n)) = \{L | L \text{ is computed by a family of circuits of depth } g(n)\}$$

3 Size of Circuits computing Boolean functions

Consider a boolean function $f : \{0,1\}^n \to \{0,1\}$ defined on the variables $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ and the complete basis $\Omega = \{\wedge, \vee, \neg\}$ where each function is having an arity 2. By claim 10, we have already seen that fixing the basis to be finite gives us the guarantee that the same result holds in other bases asymptotically. From now on we shall be working on this complete basis.

We would like to know what would be the size of the circuit computing f. How large or small can it be ? Before answering them let us see what would be the trivial circuit computing f.

An easy way to look at f is to express it as sum-of-product terms (which can be figured out from the truth table of f). Hence

$$f(x_1, x_2, \ldots, x_n) = \bigvee_{f(a_1,a_2,\ldots,a_n)=1} (x_1^{a_1} \wedge x_2^{a_2} \wedge \ldots \wedge x_n^{a_n})$$

where,

$$x_i^{a_i} = \begin{cases} \overline{x_i} & \text{if } a_i = 0, \\ x_i & \text{if } a_i = 1 \end{cases}$$

A circuit representation of f (shown in figure 3) has a root $\vee$ gate with 2^n inputs. But our

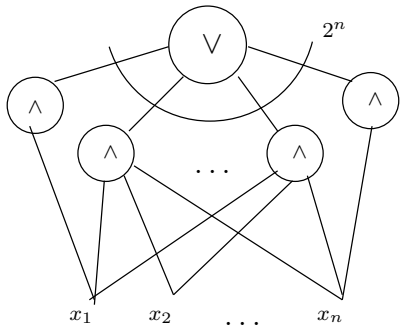

Figure 3: Circuit representation of f

basis has $\vee$ gates of arity 2. Reduction to arity 2 can be achieved by composing $\vee$ as shown. in figure 4 and replacing the root $\vee$ gate in figure 3 by this circuit. Again the $\wedge$ gates are

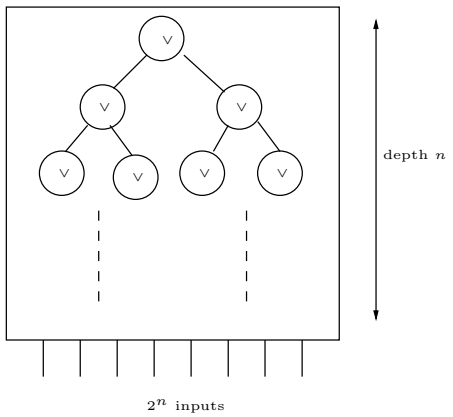

Figure 4: Arity reduction of root $\vee$ gate

taking n inputs: hence we replace the $\wedge$ gate by composing $\wedge$ s as shown in figure 5 and plug it in place of $\wedge$ in figure 3.

Now size of the circuit

$$
\begin{aligned}
&=(2^{n+1}-1) && \text{[due to } \vee \text{ gates]}\\
&\quad + 2^n \times (2n-1) && \text{[due to } 2^n \wedge \text{ gates each of size } (2n-1)]\\
&=O(n2^n)
\end{aligned}
$$

A takeaway from the whole exercise is that any function f on n input can be represented

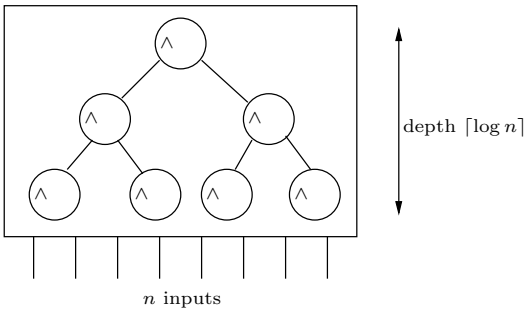

Figure 5: Arity reduction of $\wedge$ gate

using a circuit of size $O(n2^n)$. A natural question is : can there be circuits of even smaller size ?

We shall now see a lower bound due to Shannon(1942) and an upper bound due to Lupanov(1952) on the size of a boolean circuit computing a function f.

4 Shannon's Lower bound

Theorem 12. *(Shannon, 1942) For "most" of the circuits on n inputs computing a function f, the size of the circuit is larger than $\frac{2^n}{n}$ (asymptotically)*

Proof. Proof is by a counting argument on the number of boolean circuits on n inputs and of size s computing f. We argue that if we restrict the size of the circuit $s < \frac{2^n}{n}$, then the fraction of functions that can be computed using such circuits is very small.

Let $H(n, s)$ be the number of distinct circuits possible on n inputs and of size s. Observe that $s \geq n$. Since the circuit is characterised by a (unique) root, $s - 1$ gates and n inputs, we count how many ways can the s gates be *selected*. The count can be split as,

- No. of ways of fixing the root $\rightarrow s$ ways.
- Each of the internal gates (gates other than the root)
  - can compute 2^{2^2} functions[3]
  - there are $(n + s)$ possible inputs for each function and hence $(n + s)^2$ possible choices.
- Hence total ways is $s[16(n + s)^2]^{s-1}$.

[3] $\neg$ can be assumed to be of two input where it negates one input and ignores the other.

But each permutation of the gates is going to give us the same circuit. Hence total number of distinct circuits is bounded by

$$H(n,s) \leq \frac{s[16(n+s)^2]^{s-1}}{s!}$$

(Note that we are only looking locally at what is the requirement of each gate. Circuits are DAGs but since the choices made at each gate is local, we can have cycles. Hence we are counting the non-DAG circuits also. But still this count remains an upper bound)

Now,

$$\log H(n,s) \leq \log s + (s-1)\log[16(n+s)^2] - \log s!$$

By Stirling approximation, $s! \geq \left(\frac{s}{e}\right)^s$, we have $\log s! \geq s\log s - s\log e$

$$\begin{aligned}\log H(n,s) &\leq \log s + (s-1)\log[16(n+s)^2] - (s\log s - s\log e)\\ &= \log s + 2(s-1)\log(n+s) + (s-1)\log 16 - (s\log s - s\log s)\end{aligned}$$

Now, since $n \leq s$ we have $n+s \leq 2s$.

$$\begin{aligned}\log H(n,s) &\leq \log s + 2(s-1)\log 2s + (s-1)\log 16 - (s\log s - s\log e)\\ &< \log s + 2s\log(2s) + s\log 16 + s\log e\\ &= (1+s)\log s + 6s + s\log e\end{aligned}$$

Substituting $s = \frac{2^n}{n}$, we get,

$$\begin{aligned}\log H(n,s) &< \left(\frac{2^n}{n}+1\right)(n-\log n) + (6+\log e)\frac{2^n}{n}\\ &= 2^n - \frac{2^n}{n}[\log n - (6+\log e)] + n - \log n\\ &= 2^n - \frac{2^n}{n}\left[\log n - (6+\log e) - \frac{n}{2^n}(n-\log n)\right]\\ &= 2^n - \frac{2^n}{n}\log n\left[1 - \frac{(6+\log e)}{\log n} - \frac{n(n-\log n)}{2^n \log n}\right]\\ &= 2^n - \frac{2^n}{n}\log n[1-o(1)]\end{aligned}$$

Hence asymptotically,

$$H\left(n, \frac{2^n}{n}\right) \leq \frac{2^{2^n}}{2^{\frac{2^n}{n}\log n(1-o(1))}}$$

Since 2^{2^n} is the total number of functions on n inputs, we have,

$$H\left(n, \frac{2^n}{n}\right) = 2^{-\frac{2^n}{n}\log n(1-o(1))}(\text{Total no. of functions on input } n)$$

Hence the fraction of functions that can be computed with small size is very less since the number of distinct circuits is an exponentially small fraction of total functions on n inputs. Hence we conclude that large fraction of functions require size $> \frac{2^n}{n}$. □

5 Lupanov's Upper bound

Shannon's lower bound says that there is (asymptotically) large fraction of functions that remains uncomputable with circuits of size $< \frac{2^n}{n}$. What Lupanov's bound says is, if we allow circuit size to be larger by a small fraction of $\frac{2^n}{n}$, then we can compute all functions in n inputs.

Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function. We can express $f(x_1, x_2, \ldots, x_n)$ as

$$(x_1 \wedge f(1, x_2, \ldots, x_n)) \vee (\overline{x_1} \wedge f(0, x_2, \ldots, x_n))$$

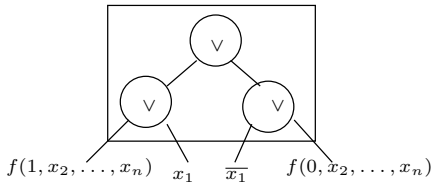

Figure 6: Circuit computing f

Hence $f(x_1, x_2, \ldots, x_n)$ can be expressed in the form of a recursive structure as, where each

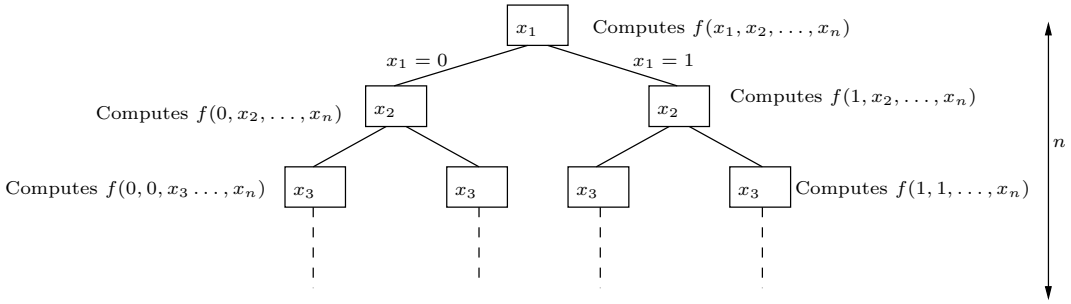

Figure 7: Complete circuit computing f

box represents the two $\wedge$ and one $\vee$ of figure 6. Now, we can find the size of the circuit follows the recurrence

$$S(n) = 2S(n-1) + 6$$

$$S(2) = 1$$

where $6 = (1 - \wedge, 2 - \vee, \text{one } \neg \text{ of } \overline{x_1}, \text{ and } 0, 1 \text{ inputs})$. Hence $S(n) = \frac{7}{4}2^n - 6 = O(2^n)$. Note that this is a better bound than the bound from the trivial circuit. So the question is how much more can we improve ? How to make the circuit more compact ?

Theorem 13. *(Lupanov, 1952) An function on input defined on the complete basis* $\Omega = \{\neg, \vee, \wedge\}$ *of two input gates can be computed by a circuit of size* $[1 + o(1)]\frac{2^n}{n}$.

Proof. Observe that at the k^{th} level (see figure 8) for $0 < k \leq n$ (where the bottom of the circuit is level 0) there are k variables appearing below it (All the $n - k$ variables above the k^{th} level would have got 0 or 1 assigned).

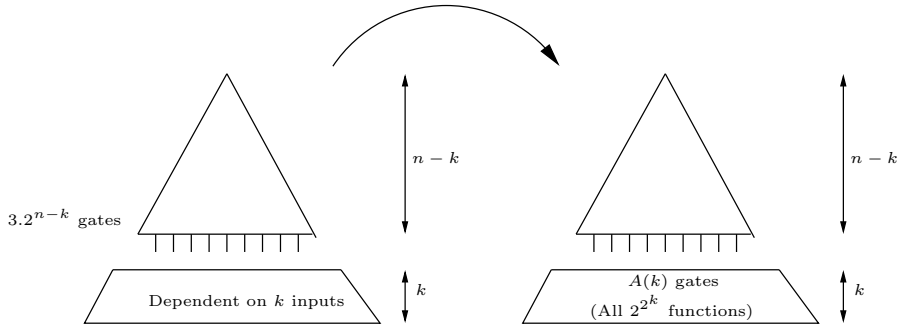

Figure 8: Compacting of circuit

Also from our construction, it is clear that there are $3(2^{n-k})$ gates at the k^{th} level. Since circuits are DAGs the input to these gates can only come from the k variables. Also there can be 2^{2^k} distinct functions computable using k inputs. Now if $A(k)$ is the number of gates required to realise these 2^{2^k} distinct functions then the $3(2^{n-k})$ gates can get input from any of the $A(k)$ gates and if we have the case where $3(2^{n-k}) > A(k)$ (after appropriately setting k), the result of computation can be reused and compaction can be achieved. So unlike the earlier case where we might have needed at least $3(2^{n-k})$ gates below the k^{th} level, it now suffices to generate all 2^{2^k} functions.

Now it remains to estimate $A(k)$. This function can be recursively characterised as

$$(x_k \wedge f_1(x_1, x_2, \ldots, x_{k-1}) \vee (\overline{x_k} \wedge f_2(x_1, x_2, \ldots, x_{k-1})$$

If we have the circuits computing all the $k - 1$ input functions, we can construct circuits computing all of the k input function using the recursive formulation.

Note that the total number of gates required to realise 2^{2^k} functions $= A(k)$

$=$ (Gates required to realise k variate functions from circuit computing functions on $k - 1$ variables)

$+$

Number of gates required to realise all the $2^{2^{k-1}}$ functions.

The first term equals

$$\begin{aligned}
&= \text{Total no. of } \wedge \text{ gates} + \text{Total no. of } \vee \text{ gates} \\
&= (2^{2^{k-1}}) \times 2 + (2^{2^{k-1}}) \times (2^{2^{k-1}}) && [\text{Two } \wedge \text{ gates per } f] \\
& && [\text{Ways of choosing } f_1,\ f_2]
\end{aligned}$$

The second term is nothing but $A(k-1)$. Hence,

$$\begin{aligned}
&A(k) = A(k-1) + 2^{2^{k-1}} \times 2 + 2^{2^k} \\
&A(0) = 2 && [\text{i.e. True or False}]
\end{aligned}$$

Note that this is nothing but the series summation

$$\begin{aligned}
&= \sum_{i=1}^{k} (2^{2^i} + 2.2^{2^{i-1}}) + A(0) \\
&= \sum_{i=1}^{k} 2^{2^i} + \sum_{i=1}^{k} 2.2^{2^{i-1}} + A(0) \\
&= \sum_{i=0}^{k-1} 2^{2^i} + \sum_{i=0}^{k-1} 2.2^{2^i} + 2 + 2^{2^k} + A(0) \\
&= 3 \sum_{i=0}^{k-1} 2^{2^i} + 4 + 2^{2^k} \\
&< 3 \sum_{i=0}^{k-1} 2^{2^{k-1}} + 2^{2^k} \\
&= 2^{2^k} + 3k.2^{2^{k-1}} = 2^{2^k} \left(1 + \frac{3k}{2^{(2^k - 2^{k-1})}} \right)
\end{aligned}$$

Hence $A(k) = 2^{2^k}(1 + o(1))$. Therefore total number of gates $= 3.2^{n-k} + 2^{2^k}(1 + o(1))$. Now to have $3.2^{n-k} > 2^{2^k}$, we choose $k = \log(n - \log n)$ (verification of k is left to the reader). Total number of gates is now

$$\begin{aligned}
&= 3. \frac{2^n}{n - \log n} + \frac{2^n}{n}(1 + o(1)) \\
&= \frac{2^n}{n} \left[\frac{3}{1 - \frac{\log n}{n}} + 1 + o(1) \right] \\
&\leq \frac{2^n}{n}(4 + o(1))
\end{aligned}$$

This proves a bound which is almost as tight as the Lupanov's bound. Reducing the constant from 4 to 1 requires more effort. □