CSE533: Information Theory in Computer Science October 13, 2010

Lecture 5

Lecturer: Anup Rao *Scribe: Widad Machmouchi*

In the previous lecture, we defined the entropy of a graph and derived some of its properties.
In particular, we showed:

Claim 1. *For two graphs G_1, G_2, $H(G_1 \cup G_2) \leq H(G_1) + H(G_2)$.*

and

Claim 2. *If $G_1, \dots, G_k$ are the connected components of G, with G_i containing a ρ_i fraction of the vertices of G, then $H(G) = \sum_{i=1}^{k} \rho_i \cdot H(G_i)$.*

We start with a quick puzzle:

Question: What is fewest number of bipartite graphs you need to cover the complete graph on n vertices?

Answer: $\lceil \log n \rceil$. The graph entropy of a bipartite graph is at most 1, and the graph entropy of the complete graph is $\log n$, which proves the lower bound. To see the upper bound, write every vertex in its binary representation using $\lceil \log n \rceil$ bits. Then the i'th bipartite graph is the complete bipartite graph between vertices whose i'th bit is 1 and vertices whose i'th bit is 0.

Today, we show how to derive a lower bound for monotone formula size using graph entropy.

1 Monotone boolean formulae and functions

Given a set of input bits $x_1, \dots, x_n$, a *boolean formula* on the inputs is a rooted tree, where every node in the tree corresponds to a boolean function of the inputs. The nodes are usually called gates. Each of the leaf gates computes the function x_i for some input bit i, or a constant function. There are three types of non-leaf gates:

- An OR gate computes the OR $(f \vee g)$ of the functions computed by its two children (f, g).
- An AND gate computes the AND $(f \wedge g)$ of the functions computed by its two children (f, g).
- A NOT gate computes the NOT $(\neg f)$ of the function computed by its lone child (f).

The entire formula computes the function computed by the gate at the root. Given a formula F, the *size* of the formula is the number of gates in F. For example, $F = (x_1 \wedge (\neg x_2)) \vee ((\neg x_1) \wedge x_2)$ is a formula computing the parity of the bits x_1, x_2. Its size is 9.

Given a boolean function $f : \{0,1\}^n \to \{0,1\}$, we can always view the function as a boolean function $f : 2^{[n]} \to \{0,1\}$ on subsets of $[n]$, by interpreting every element of $\{0,1\}^n$ as a subset in the natural way. We say that f is *monotone* if for all sets $S \subset T \subseteq [n]$, $f(S) \leq f(T)$. In other words, increasing the size of the set can only increase the value of f, or alternatively, replacing a 0 with a 1 in the input can only increase f's value. Examples of monotone functions include OR, AND and the majority function.

Say that a formula is *monotone* if it does not have any NOT gates. It is easy to check that any monotone formula must compute a monotone function. Given a monotone function f, we write $\mathsf{size_m}(f)$ for the size of the smallest monotone formula computing f. A monotone f can always be computed by the depth 2 formula

$$\bigvee_{f(S)=1} \left(\bigwedge_{i \in S} x_i \right),$$

proving that $\mathsf{size_m}(f)$ is always $O(n \cdot 2^n)$.

2 Threshold functions

For every $k \in \{0, 1, \ldots, n\}$, define the threshold function $\mathsf{Th}_k^n : 2^{[n]} \to \{0, 1\}$:

$$\mathsf{Th}_k^n(S) = \begin{cases} 1 & \text{if } |S| \geq k, \\ 0 & \text{else.} \end{cases}$$

Th_1^n is the OR function, Th_n^n is the AND function, and $\mathsf{Th}_{n/2}^n$ is the majority function. It is easy to check that $\mathsf{size_m}(\mathsf{Th}_n^n) = \mathsf{size_m}(\mathsf{Th}_1^n) = 2n - 1$, which is the size of the natural formulae for these functions. $2n - 1$ gates are necessary since both OR and AND depend on all of the inputs, so the formula must have $n - 1$ internal gates just to touch all the input gates.

What about $\mathsf{size_m}(\mathsf{Th}_k^n)$ and $\mathsf{size}(\mathsf{Th}_k^n)$ for other values of k? Valiant [3] gave a clever probabilistic construction that shows that $\mathsf{size_m}(\mathsf{Th}_k^n) = O(n^{5.3})$, but it is not clear what the right value is.

3 A small formula for Th_2^n

In this lecture, we focus on the function Th_2^n, where things already start to become interesting. It is easy to see that $\mathsf{size_m}(\mathsf{Th}_2^n) = O(n^2)$: in fact this can be done with the depth two formula:

$$\bigvee_{|S|=2} \left(\bigwedge_{i \in S} x_i \right).$$

A more careful divide and conquer based construction gives a smaller formula. Let x be the first $\lceil n/2 \rceil$ bits of the input, and y be the remaining $\lfloor n/2 \rfloor$ bits. Let $S = S_x \cup S_y$ be the two parts of the inputs viewed as sets. Then $|S| \geq 2$ if and only if $|S_x| \geq 2$, or $|S_y| \geq 2$, or both S_x, S_y have at least one element. More formally,

$$\mathsf{Th}_2^n(x, y) = (\mathsf{Th}_2^{\lceil n/2 \rceil}(x) \vee \mathsf{Th}_2^{\lfloor n/2 \rfloor}(y)) \vee (\mathsf{Th}_1^{\lceil n/2 \rceil}(x) \wedge \mathsf{Th}_1^{\lfloor n/2 \rfloor}(y)).$$

Since $\mathsf{size_m}(\mathsf{Th}_1^n) = 2n - 1$, we can recursively construct a formula for Th_2^n of size S_n, with S_n satisfying the recurrence $S_n \leq 2S_{n-1} + O(n)$. The solution of the recurrence is $S_n = O(n \log n)$. Thus, $\mathsf{size_m}(\mathsf{Th}_2^n) = O(n \log n)$. A more careful calculation shows that $S_n \leq (2n + \lceil \log n \rceil + 1)\lceil \log n \rceil$.

One can do even better than this:

Exercise 1. *Show that there is a monotone formula computing Th_2^n of size $2n\lceil \log n \rceil - 1$. (Hint: Try to make the proof for Claim 6 tight.)*

4 A lower bound for $\mathsf{size_m}(\mathsf{Th}_2^n)$ using graph entropy

We give a lower bound that almost exactly matches the upper bound from Exercise 1. We shall prove

Theorem 3. *$\mathsf{size_m}(\mathsf{Th}_2^n) \geq 2\lceil n \log n \rceil - 1$.*

Note that the lower bound and upper bound are exactly the same when n is a power of 2. The theorem (upto constant factors) was proved by Krichevskii [1]. The graph entropy based proof that we discuss here was discovered by Newman, Ragde, and Wigderson [2].

Given any monotone $f : 2^{[n]} \to \{0, 1\}$, define:

$$(f)_i = \{S \subseteq [n], |S| = i \text{ s.t. } f(S) = 1 \text{ and } \forall T \subset S, f(T) = 0\}.$$

The sets in $\cup_i (f)_i$ are the witnesses (usually called min-terms) for f being 1 on any set. Namely $f(W) = 1$ exactly when W contains some set of $\cup_i (f)_i$. Consider $(\mathsf{Th}_k^n)_j$ for all j, k. When $j < k$, this is just the empty set, since Th_k^n evaluates to 0 on all sets of size less than k. When $j = k$, this consists of all sets of size k.

When $j > k$, it is again the empty set, since although $\mathsf{Th}_k^n(S) = 1$ for sets S of size j, such an S always contains a strict subset of size k.

Let F be the smallest monotone formula computing Th_2^n. We shall prove that F must have at least $2\lceil n \log n \rceil - 1$ gates. Observe that $(\mathsf{Th}_2^n)_2$, contains all sets of size 2, yet if $f = x_i$ is a function computed at a leaf, $(f)_2$ is the empty set. For each gate in the formula computing the function f, let G_f denote the graph on the vertex set $[n]$ whose edges are $(f)_2$. Then at the root of the formula, $G_{(\mathsf{Th}_2^n)}$ is the complete graph, and at any leaf, G_{x_i} is the empty graph. Thus, as we move up the formula and consider the graphs G_f associated with the internal gates, the graphs must eventually turn into the complete graph. It is natural to try to understand how many gates it takes to build the complete graph in this process, in order to prove a lower bound.

Let us consider each of the two possible internal gates in the formula in turn.

4.1 OR gates

We claim that the graph of an OR gate is contained in the union of the graphs for its inputs:

Claim 4. *If $f = g \vee h$, then $G_f \subseteq G_g \cup G_h$.*

Indeed, if G_f contains the edge e, then $g(e) \vee h(e) = 1$. Without loss of generality assume $g(e) = 1$. For every strict one element subset $T \subset e$, we must have that $g(T) = 0$, or else $f(T) = 1$ also, contradicting the fact that $e \in G_f$. This proves that $e \in G_g$.

4.2 AND gates

AND gates are a little more tricky. Suppose $f = g \wedge h$. Life would be great if $G_f \subseteq G_g \cup G_h$, but this is not the case, for example if $e = \{1, 2\}$, we have that $e \in G_{x_1 \wedge x_2}$, but e is not in G_{x_1} nor G_{x_2}.

In fact, this is the *only* potential problem. Suppose $e = \{i, j\} \in G_f - (G_g \cup G_h)$. Consider the functions f, g, h restricted to the sets in $E = 2^{\{i,j\}}$, call them f_E, g_E, h_E. Then we have that $f_E = x_i \wedge x_j = g_E \wedge h_E$. If one of g_E or h_E is 1, then the other must be equal to f_E, which is not possible, since $e \in G_f - G_g \cup G_h$. The only remaining option is $g_E = x_i, h_E = x_j$ or $g_E = x_j, h_E = x_i$. Thus, $e \in ((g)_1 - (h)_1) \times ((h)_1 - (g)_1)$. Define the graph $T_{g,h}$ whose edges are exactly $((g)_1 - (h)_1) \times ((h)_1 - (g)_1)$. We have just argued that

Claim 5. *If $f = g \wedge h$, then $G_f \subseteq G_g \cup G_h \cup T_{g,h}$.*

4.3 A cost function based on graph entropy

Somehow the formula starts with empty graphs at the leaves, and is able to build the complete graph at the root by taking unions as described above. While OR gates are simple to control, we saw that AND gates contributed edges from the graph $T_{g,h}$. Notice that $T_{g,h}$ is bipartite. This suggests that we should measure the graph entropy of the graphs associated with the gates, since bipartite graphs have small graph entropy.

By Claim 1, if $f = g \vee h$, $H(G_f) \leq H(G_g) + H(G_h)$. On the other hand, if $f = g \wedge h$, then $H(G_f) \leq H(G_g) + H(G_h) + H(T_{g,h})$. Since $T_{g,h}$ is bipartite, $H(T_{g,h}) \leq 1$. Thus OR gates simply add up the entropy of the underlying graphs, while AND gates may add an additional 1 to the graph entropy. Since $H(G_{\mathsf{Th}_2^n}) = \log n$ and $H(G_{x_i}) = 0$, this proves the following claim:

Claim 6. *Any monotone formula for Th_2^n must have at least $\lceil \log n \rceil$ AND gates.*

Indeed, there is a formula of size $2n\lceil \log n \rceil - 1$ that uses exactly $\lceil \log n \rceil$ AND gates (see Exercise 1).

To get a lower bound on the formula size, we need a better bound on $H(T_{g,h})$. Since all the edges in $T_{g,h}$ are contained in the symmetric difference of $(g)_1$ and $(h)_1$, and the graph is bipartite, we can apply Claim 2 to show that $H(T_{g,h}) \leq \frac{|(g)_1 \cup (h)_1 - (g)_1 \cap (h)_1|}{n}$.

Now consider the cost function $\mu(f) = H(G_f) + \frac{(f)_1}{n}$. At a leaf, $\mu(x_i) = 1/n$, and at the root $\mu(\mathsf{Th}_2^n) = \log n$. We shall prove that this cost function does add for every gate, which will show that F (the smallest formula computing Th_2^n) has at least $\lceil n \log n \rceil$ leaf gates. Since F is the smallest formula, F cannot have any gates computing a constant function — such a gate cannot be at the root, and if any non-root gate computes the constant function, we can obtain a smaller formula computing Th_2^n by simplifying $f \wedge 1 = f$, $f \wedge 0 = 0$, $f \vee 1 = 1$, $f \vee 0 = f$. In particular, if f is computed by a gate in F, it must be the case that $f(\emptyset) = 0$, or else f being monotone would imply that $f = 1$. This gives us the following claim:

Claim 7. *If f is computed by any gate in F, $(f)_1 = \{\{i\} | f(\{i\}) = 1\}$*

If $f = g \vee h$, it is clear that $(f)_1$ is equal to $(g)_1 \cup (h)_1$ by Claim 7. Thus Claims 1 and 4 give

$$\begin{aligned}
\mu(f) &= H(G_f) + \frac{|(f)_1|}{n} \\
&\leq H(G_g) + H(G_h) + \frac{|(g)_1| + |(h)_1|}{n} \\
&\leq \mu(g) + \mu(h)
\end{aligned}$$

On the other hand, if $f = g \wedge h$, it is clear from Claim 7 that $(f)_1 = (g)_1 \cap (h)_1$. Thus Claims 1 and 5 give

$$\begin{aligned}
\mu(f) &= H(G_f) + \frac{|(f)_1|}{n} \\
&\leq H(G_g) + H(G_h) + H(T_{g,h}) + \frac{|(g)_1 \cap (h)_1|}{n} \\
&\leq H(G_g) + H(G_h) + \frac{|(g)_1 \cup (h)_1 - (g)_1 \cap (h)_1|}{n} + \frac{|(g)_1 \cap (h)_1|}{n} \\
&= H(G_g) + H(G_h) + \frac{|(g)_1| + |(h)_1|}{n} \\
&\leq \mu(g) + \mu(h)
\end{aligned}$$

These bounds imply that the number of leaf gates must be at least $\lceil n \log n \rceil$, or μ would not reach $\log n$ at the root. The total number of gates must be at least $2\lceil n \log n \rceil - 1$, since $\lceil n \log n \rceil - 1$ gates are required to connect all the leaf gates to the root.

References

[1] R. E. Krichevskii. Complexity of contact circuits realizing a function of logical algebra. *Soviet Physics, Dokladi 8*, pages 770–772, 1964.

[2] Ilan Newman, Prabhakar Ragde, and Avi Wigderson. Perfect hashing, graph entropy, and circuit complexity. In *Structure in Complexity Theory Conference*, pages 91–99, 1990.

[3] Leslie G. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5(3):363–366, 1984.