18.405J/6.841J Advanced Complexity Theory March 31st, 1998

Lecture 13

Lecturer: Daniel A. Spielman *Scribe: Zulfikar A. Ramzan*

In today's lecture, we will discuss circuit lower bounds as well as the tools needed to prove a theorem about lower bounds for sizes of monotone circuits which compute CLIQUE. This result is called Razborov's theorem.

1 Circuit Lower Bounds

The aim of *circuit lower bounds* research is to study the difficulty of deciding a class of functions by circuits. Usually one tries to find lower bounds on the sizes of these circuits via combinatorial analysis. The hope is to give us a better understanding of why certain problems are complicated.

For general circuits, i.e. a circuit with all three different gates (AND, OR, and NOT), it may be difficult to perform some type of clean combinatorial analysis and get a nice result. Actually, the best known lower bounds for deciding NP funtions under general circuits is only 3n. (unfortunately, linear lower bounds for circuits are not so useful in complexity theory). So to make our task of proving a lower bound easier, we restrict our study to certain classes of circuits. In this lecture we study the circuit lower bounds for deciding the **CLIQUE** function under the class of **monotone circuits**.

2 Monotone Functions and Circuits

First, let's start with some definitions.

Definition 1 *A boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ is* **monotone** *if for all x $f(x) = 1 \Longrightarrow f(x') = 1$ where x' is derived from x by flipping any* 0 *in x to* 1.

Here is another equivalent way to think about monotone functions:

Definition 2 *A function f is* **monotone** *if $f(\bar{x}) = 1 \Rightarrow f(\bar{y}) = 1$ for all $\bar{y}$'s such that all bits that are 1 in $\bar{x}$ are also 1 in $\bar{y}$. (So we can see that*

$f(\bar{y}) = 0 \Rightarrow f(\bar{x}) = 0$ *for all* $\bar{x}$*'s such that all bits that are 0 in* $\bar{y}$ *are also 0 in* $\bar{x}$)

Yet another equivalent formulation of monotone circuits is: $f(x) = 1 \Longrightarrow f(y) = 1$ whenever y is greater than x under the partial order defined by the directed edges of the n-dimensional hypercube.

Here are some examples of monotone functions:

1. The all 0 function; i.e. for all x, $f(x) = 0$.

2. The all 1 function; i.e. for all x, $f(x) = 1$.

3. Threshold functions; i.e. $f(x) = 1$ if there are more than k 1's in x.

We can also think about defining monotone functions via a *dual* property: if $f(x) = 0$ and you flip a bit in x from a 1 to a 0 to get x' then $f(x') = 0$. It turns out that we can characterize any monotone function f by its set of minterms, where minterms are defined as follows:

Definition 3 *The* **minterms** *of a monotone function* f *are the inputs on which* f *evaluates to 1 but if you flip any bit in those inputs from a 1 to a 0, then the evaluation of* f *on those new inputs is 0. More formally, the set of minterms is:* $\{x \mid f(x) = 1$, *but* $f(x') = 0$ *where* x' *is derived from* x *by flipping any* 1 *in* x *to* $0\}$.

Similarly, you can characterize f by its **maxterms**:

Definition 4 *The* **maxterms** *of a monotone function* f *are the inputs on which* f *evaluates to 0 but if you change any bit in those inputs from a 0 to a 1, then the evaluation of* f *on those new inputs is 1. More formally, the set of maxterms is:* $\{x \mid f(x) = 0$, *but* $f(x') = 1$ *where* x' *is derived from* x *by changing any* 0 *in* x *to a* $1\}$.

Here are some examples:

1. For threshold functions, the minterms are those x with exactly $k+1$ 1's. And the maxterms are those x with exactly k 1's.

2. Consider the CLIQUE function defined as follows: $\text{CLIQUE}_k(x) = 1$ if and only if the input x describes a graph that contains a clique of size greater than or equal to k. Here we describe a graph by a list of 0's and 1's (one bit for each possible edge). For example, the input x could be the bits in the upper triangular portion of the adjacency matrix of the graph listed in column major order. From this definition it is clear that CLIQUE_k is monotone, since flipping a bit from a 0 to

1 represents adding an edge to the graph – and if a graph already has a k-clique, and you add an edge to it, then the resulting graph will still have a k-clique. In this case, the minterms are the set of graphs for which k of the vertices form a k-clique, and any remaining vertices are all isolated (i.e. have degree 0). The maxterms are the set of complete $k-1$ partite graphs – i.e. graphs for which it is possible to partition the vertices into $k-1$ sets such that there must be an edge between any pair of vertices in distinct sets, but no edge between any pair of vertices in the same set.

Given this formulation of monotone functions in terms of their minterms, it's not hard to see that any monotone function computing CLIQUE can be expressed as a big OR of ANDs where each AND checks, in some sense, whether a particular minterm is in the graph. Of course there could be a simpler way to construct a circuit which computes CLIQUE.

Having defined monotone functions, we will now move into the realm of circuits:

Definition 5 *A circuit C is* **monotone** *if it has $\vee$ and $\wedge$ gates, but no $\neg$ gates.*

Proposition 6 *Every monotone circuit computes a monotone function and every monotone function can be computed by a monotone circuit.*

One can use an inductive argument to prove this proposition. Specifically, notice that $\wedge$ and $\vee$ are monotone boolean functions on two single-bit inputs. And, it's not hard to see that if f and g are monotone functions, then so are $f \vee g$ and $f \wedge g$.

3 Razborov's Theorem

Now, we present the main theorem of today's lecture: *Razborov's Theorem.*

Theorem 7 (Razborov) *Any monotone circuit which computes $CLIQUE_{\log n}$ has at least $n^{\Omega(\log n)}$ gates.*

Using similar techniques, and a slightly more intricate analysis, Alon and Boppana were able to get a stronger result:

Theorem 8 (Alon-Boppana) *Any monotone circuit which computes $CLIQUE_{n^{2/3}}$ has at least $2^{n^{1/3}/\log n}$ gates.*

The rest of today's lecture will discuss the tools needed to prove Razborov's theorem. We start with some definitions:

Definition 9 *A* **Clique detector** *is a function that is 1 if the input graph contains some particular clique, and 0 otherwise. The size of a clique detector is the number of vertices in the clique that it looks for.*

In other words, a clique detector is an AND that examines all edges between a particular subset of the vertices, and returns 1 if all the edges are present. It's easy to build a clique detector: take l vertices, and compute the $\wedge$ for all the edges between them to see whether these l vertices are part of a clique.

Definition 10 *A* **Clean function** *is an OR of at most n clique indicators of size at most k (where n is the input size).*

Here is the idea behind the proof of Razborov's theorem: Every small monotone circuit can be approximated by a clean function.

We do a recursive approximation from the inputs up. Observe that the inputs to the circuit can themselves be thought of as clique indicators of size 2; i.e. if the input is set to 1, then there is an edge present at a particular location, and this edge itself represents a clique of size 2. We show that the AND and OR of two functions that are approximated by clean functions can also be approximated by a clean functions.

But there is one problem, even if f and g are both clean functions, there is no guarantee that $f \wedge g$ or $f \vee g$ is clean. We get around this by replacing each OR with an approximate-OR and each AND with an approximate-AND.

Let's start with the OR function. Note that if f and g are clean, then $f \vee g$ is the OR of at most $2n$ clique detectors. So to make $f \vee g$, we must somehow reduce the number of detectors. We can reduce the number of detectors, by using a combinatorial result due to Erdos and Rado called the *The Sunflower Theorem.*

Definition 11 *A* **sunflower** *is a collection of sets $S_1, S_2, ..., S_m$ such that*

$$\forall i < j, S_i \cap S_j = \bigcap_i S_i$$

We call $\bigcap_i S_i$ the "core" of the sunflower, and each S_j is called a "petal".

Theorem 12 (Sunflower theorem) *Let C be a collection of sets of size less than or equal to l. If $|C| > (p-1)^l l!$, then C contains at least one sunflower with p petals.*

Now, in our case, we set our parameters as follows: $k = l = \log\log n$ and $p = (\log n)^3$ (thus, $n \geq (p-1)^l l!$). Given this setting of the parameters, it follows that if we look at the sets of vertices corresponding to the clique detectors in our circuit, then a sunflower exists. Now, we define the following operations on our circuits:

pluck(f) : let f be an OR of at least n clique detectors, pluck(f) is obtained by finding a sunflower among the clique detectors, discarding those indicators, and adding their core as a new detector.

weed(f) : keep applying pluck on f until there are at most n clique detectors left.

We can now define our approximate-OR as: approx-OR := weed($f \vee g$). We analyze how this approximation affects the circuit during the next lecture.

Now we consider the case of approximate-AND. Observe that if f and g are clean functions, then you can compute $f \wedge g$ as the OR of at most n^2 ANDs (using the distributive law) – where each AND deals with edges from at most $2k$ vertices. However, this is not necessarily a clean function. In order to come up with an approximation, we need to consider some other operations on monotone circuits:

cover : Replace each AND with a minimal clique detector that incorporates all edges examined by that AND. More formally: replace $indicator(S_i) \wedge indicator(T_j)$ with indicator$(S_i \cup T_j)$.

trim : Remove all clique detectors of size more than k.

Now, we can define our approximate-AND as follows: approx-AND(f, g) := weed(trim(cover($f \wedge g$)))

Here is some intuition: recall that $f \wedge g$ can be expressed as the OR of at most n^2 ANDs. By first applying cover, we decrease the number of clique detectors. By then applying trim, we insure that the size of each clique detector is at most k. Finally, by applying weed, we insure that there are no more than n clique detectors – thus the resulting circuit is clean. We will analyze how this approximation affects the circuit during the next lecture.

This concludes the main tools needed for Razborov's theorem. In the next lecture we will analyze these operations, and present most of the proof of Razborov's theorem.