CSC 2429 – Approaches to the P versus NP Question

Lecture #1: 15 January 2014

Lecturer: Toniann Pitassi

Scribe Notes by: David Liu

1 Introduction

In this course, our primary object of study will be families of *boolean functions* $\{f_n \mid n \geq 1\}$, where each $f_n : \{0,1\}^n \to \{0,1\}$. In particular, we'll be interested in how the *time* and *space* requirements of computing these functions grow with n.

1.1 Complexity Classes

We group boolean functions into *complexity classes* by these computational requirements. This course assumes basic familiarity with famous complexity classes: the *uniform* classes of **P**, **NP**, **L**, and **NL**, etc. derived from the standard Turing Machine model.

There are also *non-uniform* (circuit) complexity classes. Here we consider circuits made from the three boolean gates AND ($\wedge$), OR ($\vee$), and NOT ($\neg$); we restrict the AND and OR gates to be fan-in 2. The class **P/poly** is the class of (families of) boolean functions $\{f_n\}$ for which there exists a constant c such that f_n can be computed by a size n^c circuit. We can additionally analyse the *depth* of these circuits, i.e., the length of the longest path from an input literal to the output gate. The classes $\mathbf{NC}^i$ capture those functions which can be computed by poly-size circuits of depth $O(\log^i n)$, and the class **NC** is the union of all $\mathbf{NC}^i$.

By allowing the AND and OR gates to have arbitrary fan-in (rather than fan-in 2), we define the analogous complexity classes $\mathbf{AC}^i$: poly-size circuits with unbounded fan-in of depth $O(\log^i n)$. Note that unlike $\mathbf{NC}^0$, which is an extremely trivial class because it cannot hope to read all bits of an input, the class $\mathbf{AC}^0$ can, making it more interesting. We can augment this class further by allowing *MOD p* gates (returns 1 iff the sum of the inputs is 1 mod p); such classes are denoted $\mathbf{AC}^0[p]$. Finally, we define the class $\mathbf{ACC}^0$ as the union of the $\mathbf{AC}^0[p]$ over all p. The following chain of inclusions is known:

$$\mathbf{AC}^0 \subseteq \mathbf{AC}^0[2] \subseteq \mathbf{ACC}^0 \subseteq \mathbf{NC}^1 \subseteq \mathbf{NC} \subseteq \mathbf{P} \subseteq \mathbf{P/poly}$$

Finally, we remind the reader of the difference between *uniform* and *non-uniform* models of computation. The familiar Turing Machine model is a *uniform* model: the same Turing machine is expected to compute every boolean function in the family $\{f_n\}$ – put another way, it works on inputs of any length. On the other hand, a single circuit can only accept inputs of a fixed size. Thus, circuit classes are *non-uniform* because they require only that a family of circuits $\{C_n\}$ *exist* which compute $\{f_n\}$. But we can also consider uniform circuit classes: for example, we can define *uniform* $\boldsymbol{NC}$ to be the set of boolean functions for which there exists an algorithm that takes as input a number n, and outputs a circuit $C_n \in \mathbf{NC}$ computing f_n.

1.2 Goals and Progress

We've all heard of the famous question that gives this course its title: does $\mathbf{P} = \mathbf{NP}$? One of the central goals of complexity theory is to show that some boolean function in **NP** isn't in **P**. Even better, we would like to show that some boolean function in **NP** isn't in **P/poly**; this would imply that $\mathbf{P} \neq \mathbf{NP}$. But we'll also look at other, related problems in this term, including, but not limited to, the following:

- $\mathbf{NC}^1 \subsetneq \mathbf{P/poly}$?
- $\mathbf{ACC} \subsetneq \mathbf{P/poly}$?
- $\mathbf{EXP}, \mathbf{NEXP} \subsetneq \mathbf{P/poly}$?
- Algebraic lower bounds

As motivation (so that we don't give up hope!), here are some results, showing the (big) progress computer scientists have made over the years:

1. Almost every boolean function requires exponential size boolean circuits. (This is a simple counting argument.) The main problem is we haven't found an *explicit* function (informally, a function we care about) which requires exponential size circuits.

2. On the other hand, our best known general circuit size lower bounds are linear: $4n - 4$ for fan-in 2 AND and OR gates, with negation; $7n - 7$ using just AND gates and negation; and $5n - o(n)$ using all fan-in 2 gates except for parity.

3. We do know that $\mathbf{AC}^0 \subsetneq \mathbf{NC}^1$. Ajtai (1983), and independently Furst, Saxe, and Sipser (1984), proved that PARITY is not in AC^0. Better size bounds were later found by Hastad (1987), using random restrictions (which we'll talk about today), and his famous Switching Lemma, a structural result for functions computed by $\mathbf{AC}^0$. Impagliazzo, Matthews, and Paturi (2012) later used these results to obtain nontrivial *upper* bounds for satisfiability on $\mathbf{AC}^0$ circuits.

4. Ryan Williams (2011) showed that **NEXP** is not in **ACC**. His general technique – showing that relatively good upper bounds on satisfiability on circuit classes lead to the non-containment of **NEXP** within those classes – has generated much interest in finding good satisfiability algorithms.

5. Razborov (1985) showed that **monotone NP** (functions in **NP** where flipping a bit from 1 to 0 never changes output from 1 to 0) is not contained in **monotone P/poly** (circuits with no variable negations).

6. If we restrict our attention to *formula size* lower bounds (analogous to circuits with fan-out 1), there are known $n^{3-o(1)}$ bounds using AND, OR, and NOT gates (Hastad, 1998). There is also the Neciporuk (1966) lower bound $n^2/\log n$ for general fan-in 2 formulas.

2 Resolution Lower Bounds via Restrictions

For the rest of this lecture, we'll show how to use restrictions to obtain *proof size* lower bounds. One main motivation for studying proof complexity is that good lower bounds for specific proof systems can be used to rule out large classes of potential algorithms solving SAT.

We consider the *Resolution* proof system. Consider an unsatisfiable CNF $f = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. A *resolution refutation* of f is a sequence of clauses $D_1, D_2, \ldots, D_l$ where each clause is one of the C_i or is obtained from two previous D_i's by the resolution inference rule

$$\frac{(x \vee D)(\bar{x} \vee C)}{D \vee C},$$

and where the final clause D_l is the empty clause. The *size* of a resolution refutation of f is the number of clauses in this sequence.

A resolution refutation may be viewed as a DAG with fan-in 2, where the sources are the initial C_i's, and each resolved clause has as its two parents the clauses which were resolved to obtain it. The sink of this DAG is the empty clause.

We note that if $\mathbf{NP} \neq \mathbf{coNP}$, then there exist a family of unsatisfiable CNF formulas $\{f_n\}$ such that the size of the shortest resolution refutation of f_n is $\geq poly(|f_n|)$.

Now, we're going to show that the CNF encoding of the Pigeonhole Principle (n pigeons can't be put into $n-1$ holes without two pigeons sharing a hole) requires exponential size resolution refutations. Let PHP^n_{n-1} be the conjunction of the following clauses over the variables P_{ij}, $1 \leq i \leq n$, $1 \leq j \leq n-1$:

1. (Pigeon clauses) $P_{i1} \vee \cdots \vee P_{i,n-1}$, for every $i \leq n$
2. (Hole clauses) $\neg P_{ik} \neg \vee P_{jk}$ for all $i \neq j$ and $k \leq n-1$

Intuitively, the variable P_{ij} denotes the statement that pigeon i is put into hole j. The first set of clauses assert that each pigeon gets put into a hole, and the second set of clauses assert that no two pigeons get put into the same hole. Clearly, PHP^n_{n-1} is unsatisfiable.

Theorem 1 (Haken (1985)) *Any resolution refutation of PHP^n_{n-1} requires at least $2^{n/20}$ clauses (when n is sufficiently large).*

Before we begin the proof, here's some intuition. Our proof will be based on that of Razborov (and refined by Beame & Pitassi). A truth assignment to the P_{ij} is *i-critical* if it maps every pigeon except pigeon i to a unique hole, and doesn't map pigeon i to any hole. Each i-critical truth assignment (i-cta) satisfies every clause in PHP^n_{n-1} except pigeon clause i. Then for any i-cta α and resolution refutation DAG, there is a unique "false path" between the empty clause and pigeon clause i; that is, a unique path where each clause on the path is falsified by α.

By considering the false paths of *all* critical truth assignments, there must be many bottleneck nodes – nodes that not a lot of these paths go through. These bottlenecks are "wide" clauses containing a lot of literals. A bit more concretely, we'll prove the following two contradictory claims:

(1) Wide Clause Lemma: Any resolution refutation of PHP^n_{n-1} has a "wide" clause.

(2) Given a "small" resolution refutation of PHP^n_{n-1}, we can apply a restriction to the truth assignments to obtain a resolution refutation of $PHP^{n'}_{n'-1}$ with no "wide" clauses, contradicting (1).

Now let's begin the proof.

Proof For simplicity, we'll consider a monotone version of PHP^n_{n-1} where each negated literal $\neg P_{ij}$ is replaced by $\vee_{k\neq j} P_{ik}$. This preserves validity with respect to all critical truth assignments.

First we prove the Wide Clause Lemma: every resolution refutation of PHP^n_{n-1} contains a clause of width at least $\frac{2n^2}{9}$. Fix some resolution refutation P of PHP^n_{n-1}. For each clause C in P, let $Pigeons(C) = \{i \mid \text{some } i\text{-cta falsifies } C\}$. Note that $Pigeons(\text{pigeon clause } i) = \{i\}$, and $Pigeons(\emptyset) = \{1, \dots, n\}$. By soundness, we know that if we can infer a clause C_3 from C_1 and C_2, then $Pigeons(C_3) \subseteq Pigeons(C_1) \cup Pigeons(C_2)$, and hence $|Pigeons(C_3)| \leq |Pigeons(C_1)| + |Pigeons(C_2)|$; that is, the *Pigeons* function is sub-additive.

Then there exists some clause C^* in P such that $\frac{n}{3} \leq Pigeons(C^*) \leq \frac{2n}{3}$. Let $m = |Pigeons(C^*)|$. We claim that C^* has at least $m(n-m) \geq \frac{2n^2}{9}$ literals in it. Fix $i \in Pigeons(C^*)$ and let α be an i-cta falsifying C^*. Let $j \notin Pigeons(C^*)$, and let $k \leq n-1$ be the hole that α assigns pigeon j to; i.e., the unique k such that α sets $P_{jk} = 1$. Consider the j-cta α_j obtained from α by assigning pigeon i to hole k rather than assigning pigeon j to it. Then α_j must now *satisfy* C^*, because $j \notin Pigeons(C^*)$. Note that setting P_{jk} to 0 can't satisfy C^* because it is monotone; therefore setting P_{ik} to 1 must satisfy C^*, i.e., C^* contains the literal P_{ik}.

We can repeat this argument for any $j \notin Pigeons(C^*)$ to show that C^* contains $n-m$ literals of the form P_{ik}; note that for each j, the k obtained is unique, because critical truth assignments map each pigeon to a unique hole. Finally, repeating this argument for each $i \in Pigeons(C^*)$ shows that C^* contains at least $m(n-m)$ literals, and this concludes the proof of the Wide Clause Lemma.

Now we prove claim (2), using this lemma. Fix a resolution refutation P of monotone PHP^n_{n-1}, and suppose the size of P is $S < 2^{n/20}$. Consider the set $\mathcal{S}$ of clauses containing at least $\frac{n}{10}$ of the literals. On average, setting some variable $P_{ij} = 1$ will satisfy $\frac{1}{10}$ of the clauses in $\mathcal{S}$. So pick some P_{ij} which achieves this average and set it equal to 1 and all of the other variables $P_{ij'}$, $j' \neq j$ and $P_{i'j}$, $i' \neq i$ to 0. This restricts the truth assignments to those which map pigeon i to hole j. The resulting resolution refutation (after removing the set variables and the satisfied clauses) is a resolution refutation of PHP^{n-1}_{n-2} with at most $\frac{9}{10}S$ clauses of width $\geq \frac{n}{10}$ (since we started with at most S of them).

We repeat this restriction operation $\log_{10/9} S$ times to obtain a resolution refutation of $PHP^{n'}_{n'-1}$ with $n' = n - \log_{10/9} S = n - (\log_{10/9} 2)\frac{n}{20} \geq 0.67n$. On the other hand, this resolution refutation has no clauses of width $\frac{n}{10}$. This contradicts the Wide Clause Lemma, which says that this resolution refutation must have a clause of width $\frac{2n'^2}{9} > \frac{n}{10}$.