CS 221: Computational Complexity **Prof. Salil Vadhan**

Lecture Notes 4

February 3, 2010 **Scribe: Jonathan Pines**

1 Agenda

- **P**-/**NP**- Completeness
- **NP**-intermediate problems
- **NP** vs. **co-NP**
- **L**, **NL**

2 Recap

Last time, we ended with Turing Machine simulation by boolean circuits (with fan-in 2 using and, or, not gates) and saw the following theorem:

Theorem 1 *Let M be a time $t(n)$ TM, such that $\lceil \log t(n) \rceil$ is space-constructible. Then there exists a sequence $C_1, C_2, ...$ of boolean circuits such that the following properties hold:*

1. *Each circuit agrees with M on inputs of length equal to its index, i.e. $\forall n, \forall x \in \{0,1\}^n, C_n(x) = M(x)$.*
2. *The circuits are polynomially bounded in size (number of gates), i.e. $|C_n| \leq \text{poly}(t(n))$.*
3. *(log-space-uniformity) There exists a log-space (space $O(\log t(n))$) TM M' constructing the circuits, i.e. such that $\forall n, M'(1^n) = \lfloor C_n \rfloor$.*

3 P-/NP- Completeness with Circuits

Now, we examine **P** and **NP** in the framework of circuits.

3.1 P as Circuits

Corollary 2 *The set of languages decidable by log-space-uniform families of polynomially-sized boolean circuits is equal to* **P**.

Proof:
($\supseteq$) Follows from Theorem 1.
($\subseteq$) Given a family of circuits and an input, we can use log-space-uniformity to construct the appropriate circuit, and evaluate it gate by gate, both in poly-time. ■

So, one approach to the **P** vs **NP** problem is to prove that some **NP** problem requires super-polynomial-sized circuits to decide it, which by Corollary 2 would show $\mathbf{P} \neq \mathbf{NP}$. Intuitively, circuits seem easier to reason about than Turing Machines, and historically there was a lot of excitement around this approach. Unfortunately, proving lower bounds on the number of gates needed in a circuit is hard, and the best known bound for an **NP** problem is something like $5n$. Any super-linear lower bound would be a major breakthrough. It is currently believed that even without the assumption of uniformity, **NP**-complete problems that require super-polynomial circuits.

Theorem 3 *The following language is **P**-complete with respect to $\leq_l$:*
CIRCUIT VALUE $= \{\langle C, x\rangle : C(x) = 1\}$.
Recall that $\leq_l$ refers to log-space reduction; note that using poly-time reductions, anything in **P** *is* **P***-complete.*

Proof:
($\in$ **P**) The circuit can be evaluated in the input in time polynomial in the circuit size.
(**P**-hard) Given a language $L \in \mathbf{P}$, let M be a poly-time TM deciding L. Apply Theorem 1 to get a family $\{C_n\}$ of circuits, and then use log-space-uniformity to get a log-space reduction to CIRCUIT VALUE since $x \in L \Leftrightarrow M(x) = 1 \Leftrightarrow C_{|x|}(x) = 1 \Leftrightarrow \langle C_{|x|}, x\rangle \in$ CIRCUIT VALUE. ■

3.2 Other P-complete problems

- Linear Programming (see Problem Set 1).
- Max-flow.

As a corollary to Theorem 3, one can show that if any of these problems is in **L**, then $\mathbf{P} = \mathbf{L}$. The same can be said of other complexity classes within **P**. If any of these can be efficiently parallelized, then so can all of **P**.

3.3 Circuit Satisfiability

Theorem 4 *The following language is* **NP***-complete with respect to $\leq_l$:*
CIRCUIT SATISFIABILITY $= \{C : \exists x : C(x) = 1\}$.

Proof:
($\in$ **NP**) The satisfying assignment serves as witness.
(**NP**-hard) Given a language $L \in \mathbf{NP}$, let M be a poly-time verifier for L. Apply Theorem 3 to the verifier to get a family $\{C_n\}$ of circuits such that $C_{n+p(n)}(x, u) = M(x, u)$, where $n = |x|$ (and maybe off by a couple for divider between the pair). Then, given an input x, we can use log-space-uniformity to compute $C_{n+p(n)}$, fix x in the circuit to get a new circuit $C_x(\cdot) = C_{n+p(n)}(x, \cdot)$, and output C_x. Noting that $x \in L \Leftrightarrow M$ accepts $\Leftrightarrow C_x$ is satisfiable, we get a log-space reduction of L to CIRCUIT SATISFIABILITY as desired. ■

Theorem 5 3SAT *is* **NP***-complete.*

Proof:

($\in$ **NP**) A satisfying assignment serves as witness.

(**NP**-hard) We reduce Circuit Satisfiability to 3SAT. Given a circuit C with inputs $x_1, ..., x_n$ and m gates, we will construct a 3-CNF formula ϕ that is satisfiable iff the circuit is. To do this, we introduce variables $g_1, ..., g_m$ for the m gates of C. Then for each gate with inputs (x, y) and output z (where the inputs can be x's or g's), we add a term for each assignment to (x, y, z) that is not consistent with the gate operation. For example, if the gate is an OR gate, then the term $x \vee y \vee \neg z$ ensures that $(x, y, z) = (0, 0, 1)$ will not satisfy ϕ. The general fact at play here is that any function on k variables can be written as a k-CNF formula using one clause for each unsatisfying assignment. Finally, we take the conjunction of the output value g_m with all of these terms so that the formula is satisfiable if and only if there is a valid execution of the circuit resulting in the output $g_m = 1$, as desired. ■

Note that we have not claimed 3SAT is as powerful as a Turing Machine (false), but rather that a 3SAT instance can *verify* the computations of a boolean circuit or Turing Machine. The intuition is that the freedom introduced by the new variables g when finding a witness makes it simpler to verify solutions.

4 NP-intermediate Problems

Under the assumption that $\mathbf{P} \neq \mathbf{NP}$, one might question whether there are languages in **NP** that are neither in **P** nor **NP**-complete.

4.1 Ladner's Theorem

Theorem 6 *(Ladner's Theorem) If* $\mathbf{P} \neq \mathbf{NP}$*, then there exists a language* $L \in \mathbf{NP}$ *such that* $L \notin \mathbf{P}$ *and* L *is not* **NP***-complete. We call such languages* **NP***-intermediate languages.*

Proof: We omit the proof here; see Arora-Barak. The basic idea is to start with $L_0 \in \mathbf{NP} \setminus P$, and then obtain L by blowing large "holes" in L_0 (i.e. throwing out large intervals of strings). Each hole serves to rule out one polynomial-time reduction from SAT to L_0, and each gap between the holes serves to rule out one polynomial-time algorithm for L. This is possible by taking the holes and the gaps to be sufficiently large. To ensure that L remains in **NP**, lazy diagonalization is used. See also Problem Set 1 & section for other evidence for the existence of **NP**-intermediate problems. ■

4.2 Natural candidates for NP-intermediateness

- Factoring integers (see PS1).
- Finding Nash equilibria, e.g. in bimatrix games.
- Anything in $(\mathbf{NP} \cap \mathbf{co\text{-}NP}) \setminus \mathbf{P}$.
- **TFNP** - total search problems; subclass **PPAD** of fixed-point problems (including finding Nash equilibria).
- **BQP**.

- **SZK**.
- Lots more.

Proposition 7 *If there exists some language $L \in \mathbf{NP} \cap \mathbf{co\text{-}NP}$ that is $\mathbf{NP}$-hard with respect to Cook reduction (i.e. poly-time given an L-oracle), then $\mathbf{NP} = \mathbf{co\text{-}NP}$. (Common belief is that $\mathbf{NP} \neq \mathbf{co\text{-}NP}$, so there is probably no $\mathbf{NP}$-hard language in $\mathbf{NP} \cap \mathbf{co\text{-}NP}$.)*

5 NP vs. co-NP

Claim 8 $\mathbf{NP} = \mathbf{co\text{-}NP}$ *iff all boolean propositions have poly-sized proofs of validity, with respect to some poly-time checkable proof system.*

Proof Sketch: Tautologies is in **co-NP**, and **NP** languages allow for poly-time verifiable proofs of membership. □

The study of *Proof Complexity* approaches **NP** vs. **co-NP** by attempting to find tautologies that have a provably super-polynomial lower bound for proof length in natural proof systems; this has been successful for relatively weak proof systems (like Resolution) but elusive for stronger systems (like Frege, or Zermelo-Frankel Set Theory). The intuition is that we can't have poly-sized certificates for exponentially many inputs. Proof Complexity is a very active area of research that we unfortunately will not have time to say more about.

6 Space Complexity: L=Log-space

6.1 Review

- We only count the number of cells used on work tapes. Input/output are read/write only, respectively
- $\mathbf{L} = \mathbf{SPACE}(\log n)$
- $\mathbf{L} \in \mathbf{P}$, since the number of configurations of an **L**-space machine is at most polynomial.

6.2 Why study L?

- **L** is a natural point at which nontrivial algorithms exist, such as $+$, $\cdot$, $\div$, and undirected connectivity (UPATH). The latter two are nontrivial and were discovered in the last 5 years.
- Massive datasets can't be held in memory. For this, people use even more restricted models, like log-space streaming and k-pass algorithms.
- **L** is closely related to parallel computation.
- But, $\mathbf{NP} = \mathbf{L}$ is not ruled out.

6.3 Addition in L

In log-space, we can store pointers to input/output by maintaining the index of a given digit. Addition can be computed one digit at a time by maintaining pointers to the current input/output as well as a carry bit at each step. We print the digits from left to right, starting with the least significant bit.

What if we wanted to print the output of addition in reverse order? We can't just reverse the output, because the length can be n or $n+1$, where n is the length of the summands. To get around this, we run the entire addition process without outputting anything except the last (most significant) bit, and then repeat the addition process for each other bit (stopping each addition when we get the bit needed at that step). This algorithm suggests the general result that follows:

6.4 Closure of L under composition

Proposition 9 *If f_1, f_2 are computable in log-space, then so is $f_2 \circ f_1$.*

Proof: Note that we can't simply write the output of f_1 to a work tape, because the output may be polynomial while the tape is constrained to logarithmic size. The intuition is that as for reversed addition, we compute each output bit of f_1 as f_2 actually needs to read it.

Slightly more formally, we maintain an input pointer for f_2, and each time it wants to read a value, we simulate f_1 (without printing) until its output head matches f_2's input head, at which point we print that bit and continue executing f_2. This can be thought of as a "virtual tape" for the input to f_2, where each input bit is printed as it is needed for access.

Using this model, we can effectively run f_2 on the output of f_1 while printing only one bit at a time of f_1's output onto the work tape.

■

Corollary 10 *Log-space reductions are transitive.*

Corollary 11 $A \leq_l B, B \in \mathbf{L} \Rightarrow A \in \mathbf{L}$.

7 NL

Next, we look at nondeterministic log-space.

Theorem 12 *The following language is* **NL***-complete:*
PATH $= \{\langle G, s, t\rangle : G$ *is a directed graph with an* $s \to t$ *path*$\}$.

Proof:

Note that the undirected version of this language is in **L**.

($\in$ **NL**) Let $v = s$, repeatedly (up to n times) replace v with a nondeterministically chosen one of its neighbors, and halt/accept if $v = t$. Space required is $O(\log n)$ to remember v and a counter for the loop.

(**NL**-hard) Next time - think about it! ■

8 Next Time

- Finish proof of Theorem 12.
- Consequences: proving things about **NL** by reasoning about PATH.
- **NL** = **co-NL**.
- **NPSPACE** = **PSPACE**.