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

Lecture Notes 9

March 1, 2010 **Scribe: Önder Eker**

Correction (May 2016): Ryan Williams has informed me that the proof that NTIME(n) (and hence SAT) is not in TISP(n^{1+o(1)},n^{1-epsilon}) does *not* relativize in the usual model of relativized space-bounded computation, where the oracle machine can ask oracle queries (written on a write-only tape) that are longer than its space bound. This causes the simulation of the TISP class by Sigma_2TIME to fail (since describing a configuration can now require much more length than the space bound). In fact, the equivalence of AP and PSPACE, which we used to prove PSPACE-completeness of TQBF also runs into the same problem. There are alternate models of relativized space-bounded computation, but then there are other proofs that don't relativize (like APSPACE=EXP). See Fortnow's survey "The role of relativization in complexity theory".

1 Agenda

- Relativization
- Circuit Complexity
- **P/poly**
- Karp-Lipton Theorem

2 Recap

Lower bounds so far:

$$\text{TQBF} \notin \mathbf{SPACE}(n^{\varepsilon}), \quad \text{SAT} \notin \mathbf{TISP}(n^{1+o(1)}, n^{1-\varepsilon}), \quad \text{ALL}_{\text{REX}\uparrow} \notin \mathbf{SPACE}(o(2^n))$$

All of these proofs were based on diagonalization, sometimes combined with nontrivial simulations of one resource by others (such as the simulation of **TISP** by $\mathbf{\Sigma_2 TIME}$). Finally, we use completeness to deduce a lower bound for a natural problem, but in all cases this was done, or could be done, as a final step in the proof.

In the next section, we ask whether such an approach can resolve **P** vs. **NP**.

3 Relativization

The key observation is that all of the diagonalization and simulation arguments we have done so far hold in the presence of an arbitrary oracle. (Sometimes we have presented these arguments using specific complete problems, eg the way we used PATH to Savitch's Theorem, but they can actually be done directly, e.g. using the configuration graph.)

The next theorem states that using techniques that hold relative to an oracle can't resolve **P** vs. **NP**.

Theorem 1 *There exist oracles A, $B : \{0,1\}^* \to \{0,1\}$ such that*

1. $\mathbf{P}^A = \mathbf{NP}^B$
2. $\mathbf{P}^A \neq \mathbf{NP}^B$

This phenomenon was new in complexity theory compared to recursion/computability theory, where almost all results relativize — hold relative to an arbitrary oracle. The initial expectation was that recursion-theoretic methods, such as sophisticated diagonalization arguments, could resolve **P** vs. **NP**. The above theorem provides a dramatic explanation of why it wouldn't work.

Proof:

1. The proof is based on the fact that, for space-bounded computation, there is not much gap between deterministic and nondeterministic computation. We take $A = \text{TQBF}$.

$$\mathbf{NP}^{\text{TQBF}} \subseteq \mathbf{NPSPACE} = \mathbf{PSPACE} \subseteq \mathbf{P}^{\text{TQBF}} \subseteq \mathbf{NP}^{\text{TQBF}}$$

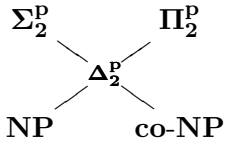
   This holds because,

   $\mathbf{NP}^{\text{TQBF}} \subseteq \mathbf{NPSPACE}$: running **NP** with oracles calls takes poly space.

   $\mathbf{NPSPACE} = \mathbf{PSPACE}$: Savitch's theorem.

   $\mathbf{PSPACE} \subseteq \mathbf{P}^{\text{TQBF}}$: TQBF is **PSPACE**-complete. Any **PSPACE** problem can be reduced in poly space to one oracle call to TQBF.

   Side Remark: (response to question) Taking $A = \text{SAT}$ doesn't work. With this choice, $\mathbf{NP}^A = \mathbf{\Sigma_2^p}$ and $\mathbf{P}^A = \mathbf{\Delta_2^p}$. If they are equal, **PH** collapses.



2. We will find a language B such that

$$U_B = \{1^n \ : \ \exists x \in \{0,1\}^n,\, B(x) = 1\} \in \mathbf{NP}^B \backslash \mathbf{P}^B$$

   Note that for every oracle B, $U_B \in \mathbf{NP}^B$ because the witness for membership is just the x for which the oracle says yes. We design the oracle such that no polynomial time algorithm can decide the language. This should not be too difficult since there are exponentially many choices and the machine can only look at the oracle in polynomially number of places. There is no resource constraint for the oracle B. We construct B by diagonalization. Let M_1, M_2, M_3, ... be an enumeration of all oracle TMs running up to time $2^n/2$. The iteration proceeds by showing that on some inputs M_1 with oracle B doesn't decide U_B correctly. Once we have shown for M_i, we can pick some more inputs for M_{i+1} and so on. Pseudocode on the next page ensures that $1^{n_i} \notin M_i^B$.

■

Remark 2 Completeness of a particular problem is not a statement that holds relative to an oracle. However, as mentioned above, so far we have not used completeness in a fundamental way for lower bounds - it can always be pushed to the end of the argument.

for $i = 1, 2, 3, \ldots$ **do**
  idea: Fix B on finitely many inputs to ensure $L(M_i^B) \neq U_B$
  Pick n_i s.t. we haven't fixed B on any input of length n_i yet.
  for all query q in the simulation of $M_i^B(1^{n_i})$ **do**
    if $B(q)$ already determined **then**
      Answer $B(q)$
    else
      Fix $B(q) = 0$, and answer 0.
    end if
  end for
  There must be some input of length n_i that $M_i^B(1^{n_i})$ hasn't queried (since $M_i^B(1^{n_i})$ runs in time less than 2^{n_i}).
  for all q that $M_i^B(1^{n_i})$ hasn't queried **do**
    Fix $B(q) = \neg M_i^B(1^{n_i})$
  end for
  $\Rightarrow M_i^B(1^{n_i}) \neq U_B(1^{n_i})$
end for

Remark 3 Most open questions about complexity classes are known to *not* have relativizing answers. In addition to **P** vs. **NP**, these include **P** vs. **PSPACE** and whether **PH** collapses. But we *do* have some nonrelativizing results such as **IP** = **PSPACE** (interactive proof system), **PCP** theorem (probabilistically checkable proof), $\mathbf{MA}_{\mathrm{EXP}} \not\subseteq \mathbf{P/poly}$ ($\mathbf{MA}_{\mathrm{EXP}}$ is a randomized analogue of **NEXP**).

Remark 4 We shouldn't be overly pessimistic and shouldn't interpret nonrelativization as limitations of current techniques but see it as a guide for what might or might not work.

4 Circuit Complexity

One approach to get around the relativization barrier for proving $\mathbf{P} \neq \mathbf{NP}$ is to use circuit complexity.

4.1 Definitions

Recall:

Definition 5 *A* boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ *has*

- n *sources (*input nodes*),* m *sinks (*output nodes*)*
- *some computation nodes (gates) labeled by functions* $\{0,1\}^{\mathrm{indegree}(v)} \to \{0,1\}$
- size = *number of nodes*
- fan-in k = *max indegree. Usually* $k = 2$.

Definition 6 *For a function $f = \{0,1\}^n \to \{0,1\}^m$, $size_B(f)$ is the minimum size boolean circuit computing f with gates taken from basis B (a set of operations). Basis B is universal if all boolean functions can be computed with gates from B.*

Example 7 Some universal bases:

- $B_k = \{\text{all boolean functions on } k \text{ variables}\}$, $k \geq 2$, $|B_k| = 2^{2^k}$
- $S = \{\wedge, \vee, \neg\}$
- $\{\text{NAND}\}$
- $\{\wedge, \otimes\}$ = addition and multiplication mod 2. Provides a connection between boolean circuit complexity and arithmetic circuit complexity.

Gates in one universal basis can be expressed by a constant number of gates in another universal basis:

Fact 8 *If B, B' are universal bases then $\forall f$ $size_{B'}(f) \leq c_{B,B'} \cdot size_B(f)$.*

Recall: Every language L in **P** has polynomial sized circuits C_1, C_2, ... s.t.

1. $|C_n| \leq \text{poly}(n)$
2. $\forall x \in \{0,1\}^n, \quad C_n(x) = L(x)$

The converse is false. Consider the unary language $L = \{x : |x|^{th} \ TM \text{ halts on } \varepsilon\}$. It is undecidable, but has trivial poly-sized circuits (since L is constant on each input length). In a previous lecture, we've seen that **P** consists of languages decidable by *uniform* (log-space or P-uniform) poly-sized circuits. This motivates a TM characterization of nonuniform poly-sized circuits.

Definition 9 *For a class C of languages $a : \mathbb{N} \to \mathbb{N}$ we say $L \in C/a$ if $\exists L' \in C$, and "advice strings" $\alpha_1, \alpha_2, \alpha_3, \ldots \in \{0,1\}^*, \quad |\alpha_n| = a(n), \quad \forall x \in L \iff (x, \alpha_{|x|}) \in L'$.*

Theorem 10 $\{$*languages decidable by poly-sized circuits*$\} = \mathbf{P/poly} \stackrel{\text{def}}{=} \bigcup_c \mathbf{P}/n^c$

Proof:

($\subseteq$) $\alpha_n = \lfloor C_n \rfloor$ (description of the n^{th} circuit), $\quad L' = \{(x, \lfloor C \rfloor) \ : \ C(x) = 1\}$

($\supseteq$) L' decided by poly-sized circuits C'_1, C'_2, ... Circuits for L are given by $C_n(\cdot) = C'_{n+a(n)}(\cdot, \alpha_n)$ with α_n hardwired to the circuit.

■

4.2 Motivation for studying P/poly

P $\neq$ NP : Using circuit complexity approach, hope is to prove nonrelativizing lower bounds.

Circuit design: Minimizing the amount of hardware on a chip.

Precomputation: Work hard to compute α_n, then the problem is easy on inputs of length n.

4.3 Karp-Lipton Theorem

One way to approach **P** vs. **NP** is to show that an **NP**-complete problem is not in **P/poly**. This is a harder problem. Hope is that circuits are easier to reason about than TMs. Note that **P/poly** contains even undecidable problems. Can it be that **NP**-complete problems have poly-sized circuits? Karp-Lipton Theorem says it is unlikely.

Theorem 11 (Karp-Lipton Theorem) $\mathbf{NP} \subseteq \mathbf{P/poly} \Rightarrow \mathbf{PH} = \mathbf{\Sigma_2^p}$

One way to interpret Karp-Lipton Thm is as an evidence that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. However, this is only based on our intuition that **PH** doesn't collapse. A more objective interpretation is that it establishes a connection between nonuniform and uniform complexity. If there is a collapse with nonuniform algorithms, then there is a collapse with uniform algorithms.

Proof: Suppose $\mathbf{NP} \subseteq \mathbf{P/poly}$. Then, by search vs. decision equivalence for **NP**, there exists poly-sized circuits $\{C_n\}$ s.t. $\varphi \in \text{SAT} \Rightarrow C_{|\varphi|}(\varphi)$ outputs a satisfying assignment to φ. To prove the theorem, it suffices to show ($\Pi_2\,\text{SAT} \in \mathbf{\Sigma_2^p}$) since ($\mathbf{\Pi_2^p} = \mathbf{\Sigma_2^p}$) implies ($\mathbf{PH} = \mathbf{\Sigma_2^p}$). An instance of Π_2SAT is given by

$$\forall u \underbrace{\exists v\ \varphi(u, v)}_{\text{SAT}}$$

The idea is to first guess ($\exists$) the circuit for SAT and then use it to remove the inner $\exists$ quantifier. We may guess the wrong circuit, but since the circuit is supposed find a satisfying assignment to $\varphi_u(\cdot) \stackrel{\text{def}}{=} \varphi(u, \cdot)$, we can check the result by applying φ_u. Specifically, we can solve our Π_2SAT instance as follows:

$$\forall u\ \exists v\ \varphi(u, v) \Leftrightarrow \exists C\ \forall u\ \varphi_u(C(\varphi_u)).$$

■

There are analogues of the Karp–Lipton Theorem for other classes, such **EXP** vs. **P/poly**. There too, the result is based on some kind of checkability property of **EXP**-complete computations.

Next lecture: Most functions don't have poly-sized circuits. In fact, most functions require huge circuits.