CMPT 407/710 - Complexity Theory: Lecture 20

Valentine Kabanets

July 27, 2017

1 Exponential-Time Hypothesis and Polytime Problems

Impagliazzo and Paturi's *Exponential-Time Hypothesis (ETH)* says that k-SAT requires time $2^{\Omega(n)}$. More formally, for every k, there exists a $\delta = \delta(k) > 0$ such that k-$\mathsf{SAT} \notin \mathsf{BPTIME}(2^{\delta n})$.

While already a strengthening of the assumption that $\mathsf{SAT} \notin \mathsf{BPP}$, ETH can be further strengthened to the following *Strong ETH (SETH)*, which says that k-SAT requires time $2^{n(1-o(1))}$. More formally, for $\delta = \delta(k)$ as above, we have that $\delta(k) \to 1$ as $k \to \infty$.

1.1 Orthogonal Vectors (OV)

Consider the following problem: Given two sets of d-dimensional vectors $A, B \subseteq \{0,1\}^d$, where $|A| = |B| = n$, decide if there is a pair of vectors $a \in A$ and $b \in B$ such that $\langle a, b \rangle = 0$ (i.e., a pair of orthogonal vectors). Think of $d = O(\log n)$.

The naive algorithm for this problem takes time $O(n^2 \cdot \mathsf{poly} \log n) = \tilde{O}(n^2)$: Simply try all n^2 pairs of vectors, and see if any pair are orthogonal.

OV Conjecture: No algorithm solves OV in time less than $n^{2-\epsilon}$, for some constant $\epsilon > 0$.

Theorem 1 (R. Williams). *SETH $\Rightarrow$ OV Conjecture.*

Proof. We prove the contrapositive: a better than n^2 algorithm for OV will imply a better than 2^n algorithm for k-SAT.

Given an instance of k-SAT: $\phi(x_1, \dots, x_n) = C_1 \wedge C_2 \wedge \dots \wedge C_m$, with n variables and m clauses (of size at most k each), construct the following two sets of m-dimensional vectors A and B:

> Enumerate all partial truth assignments to $x_1, \dots, x_{n/2}$ (assume n is even, wlog), and for each such assignment $z \in \{0,1\}^{n/2}$, place into set A a vector $V_z = (v_1, \dots, v_m)$, where $v_i = 0$ if clause C_i is satisfied by the partial assignment z to the variables $x_1, \dots, x_{n/2}$, and is 1 otherwise.

Similarly, enumerate all partial truth assignments to $x_{n/2+1}, \ldots, x_n$, and for each such assignment $z \in \{0,1\}^{n/2}$, place into set B a vector $V_z = (v_1, \ldots, v_m)$, where $v_i = 0$ if clause C_i is satisfied by the partial assignment z to the variables $x_{n/2+1}, \ldots, x_n$, and is 1 otherwise.

It is easy to see that ϕ is satisfiable iff there are vectors $a \in A$ and $b \in B$ such that $\langle a, b \rangle = 0$.

Thus to solve k-SAT, we construct an instance of OV as above, in time $O(2^{n/2})$, where $|A| = |B| = N := 2^{n/2}$. Then we apply a hypothetical $\tilde{O}(N^{2-\epsilon})$-time algorithm for OV to decide if ϕ is satisfiable. The overall time is at most $\tilde{O}(N^{2-\epsilon}) = \tilde{O}(2^{n(2-\epsilon)/2}) = \tilde{O}(2^{n(1-\epsilon')})$, for a constant $\epsilon' = \epsilon/2 > 0$. The latter runtime is true for any k-SAT, which violates SETH. $\square$

Similar results are known for many other problems in P. That is, assuming SETH, we know exact polynomial-time complexity for problems like Edit Distance, All-Pairs Shortest Paths, etc. The upshot is: even if you are working on algorithms for "simple" problems like Edit Distance, and trying to beat the well-known n^2-time Dynamic Programming algorithm, you are also trying to get a better than brute force algorithm for SAT! Maybe that's a good way to get a faster SAT algorithm! Or, maybe it's an indication that the Edit Distance algorithm can't be improved below quadratic time. Either way, you can't escape working on SAT algorithms, even if you are an Algorithms (rather than Complexity) person, interested in designing better polytime algorithms for natural problems in P!

2 Barriers

Why can't we resolve the "P vs. NP" question? One of the basic answers is: We need "new techniques"! But what are the old techniques and why are they not useful for resolving the big open questions such as the "P vs. NP"?

There are several formal answers.

1. Relativization: "black-box" (relativizing) techniques alone are not enough to resolve "P vs. NP" [Baker, Gill, Solovay'1975].

2. Algebrization: "arithmetization of propositional formulas" is not enough to resolve "P vs. NP" [Aaronson, Wigderson'08 and Impagliazzo, Kabanets, Kolokolova'09]

3. Natural proofs: "natural arguments" are not enough to prove superpolynomial circuit lower bounds for those circuit classes that can compute secure PseudoRandom Generators [Razborov, Rudich]

We present more details next.

2.1 Relativizing techniques

What are *relativizing* techniques? Recall our definition of oracle Turing machines. These are TMs with an extra tape, oracle tape, where they can write a query (a string), and in the next step of computation, they get back the answer whether that string is in the oracle or not. When we choose a particular oracle A (which, recall, is just a language $A \subseteq \{0,1\}^*$), our oracle machine will get its oracle queries answered according to that language A. This oracle TM is denoted M^A to stress that it has oracle access to the oracle A.

If you recall proofs such as $\mathsf{P} \subseteq \mathsf{NP}$ or $\mathsf{EXP} \not\subseteq \mathsf{P}$, you will convince yourself that the same proofs also show that, relative to *any* oracle A, $\mathsf{P}^A \subseteq \mathsf{NP}^A$ and $\mathsf{EXP}^A \not\subseteq \mathsf{P}^A$. In general, we say that a proof technique/argument is *relativizing* if it remains valid relative to every oracle A. That is, if we prove some statement (e.g., $\mathsf{EXP} \not\subseteq \mathsf{P}$) using relativizing arguments, then our proof can be modified to also show the same statement relative to any given oracle A; the modification is syntactic – add oracle access to A everywhere in the proof where you talk about TMs.

The two basic techniques, simulation (which means that there is a TM that can simulate any other TM with not too much overhead) and diagonalization (which means we can construct a language not accepted by any TM of certain type), are examples of relativizing techniques!

Such techniques are "black-box" in the sense that we are treating the TM as a "black-box": Given a TM M, we don't open it up, but simply run (simulate) it on a given input x. If that TM were an oracle TM with oracle A, we would still just run it on a given input x. If our proof methods are based only on the idea "just run it" for Turing machines (i.e., be completely black-box), then, as Baker, Gill, Solovay argued, we will never resolve the "P vs. NP" question!

More formally, they proved the following:

Theorem 2 (Baker, Gill, Solovay). *There are oracles A and B such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.*

As a consequence, *there cannot be a relativizing proof of $\mathsf{P} = \mathsf{NP}$, or a relativizing proof of $\mathsf{P} \neq \mathsf{NP}$.* The reason is that such a proof would remain valid relative to any oracle (by the definition of a relativizing proof), but there are oracles relative to which "P vs. NP" has opposite answers.

Proof of Theorem 2. Take $A = TQBF$, the PSPACE-complete language. We have that $\mathsf{P}^A \subseteq \mathsf{NP}^A$ (for trivial reasons), then $\mathsf{NP}^A \subseteq \mathsf{NPSPACE}$ (as we can answer A-queries ourselves in PSPACE), then by Savitch's theorem we have $\mathsf{NPSPACE} = \mathsf{PSPACE}$, and finally, by the definition of PSPACE-completeness, we get $\mathsf{PSPACE} \subseteq \mathsf{P}^A$. Thus, we got

$$\mathsf{P}^A \subseteq \mathsf{NP}^A \subseteq \mathsf{NPSPACE} = \mathsf{PSPACE} \subseteq \mathsf{P}^A.$$

This implies that all inclusions are in fact equalities, and so in particular $\mathsf{P}^A = \mathsf{NP}^A$.

Now we will construct the oracle B. First, given any oracle B (which will be determined later), define the unary language

$$U_B = \{1^n \mid \exists x \in B,\ |x| = n\}.$$

Observe that $U_B \in \mathsf{NP}^B$ (no matter what B is). Indeed, given 1^n, our oracle NTM will nondeterministically guess a string x of length n, and check if $x \in B$ (by an oracle query), accepting if the oracle query is answered positively.

Next we will define B in such a way that $U_B \notin \mathsf{P}^B$. We construct B in stages. At first, we set $B = \emptyset$. Before stage i (for $i > 0$), we will have made decisions on which strings to put in B and which to leave out for a finite number of strings. Let ℓ be the length of the longest string about which we've already made our decision. Set $n = \ell + 1$. In stage i, we will extend our set B so as to fool the ith TM M_i, where we have an enumeration of all TMs running in time at most $2^{n/10}$ on inputs of size n.

We will run $M_i^B(1^n)$ on the partially defined oracle B. When M_i asks about a string for which we've already made the decision to put it or not put it in B, we answer according to our previous decision. If M_i asks about a new string, we answer negatively, and make this decision permanent.

After $2^{n/10}$ steps, the machine M_i will halt with Yes or No as its answer. If it answers Yes, then we decide not to put into B any string of length n. If M_i answers No, we pick some n-bit string that was not asked about by M_i during its simulation (such a string exists since M_i could only ask about $2^{n/10}$ strings, whereas there are 2^n strings of length n in total) and place that string into B. In both cases, the new partially defined set B is such that $M_i^B(1^n)$ makes a mistake when deciding U_B (because we change the oracle B for strings that are never queried by $M_i^B(1^n)$, and so the answer of $M_i^B(1^n)$ remain the same).

It follows that the final set B (obtained after infinitely many stages) is such that no $2^{n/10}$-time TM M^B is correct for the language U_B, and hence, $U_B \notin \mathsf{P}^B$. $\square$