COMS 6998: Algebraic Techniques in TCS (Fall'21) Oct 19, 2021

Lecture 6: HW1 Review, Fine-Grained Complexity

Instructor: *Josh Alman* Scribe notes by: *Tsung-Ju Chiang, Chengyue He*

Disclaimer: This draft may be incomplete or have errors. Consult the course webpage for the most up-to-date version.

1 Topics

- HW1 Review (omitted here)
- Fine-Grained Complexity

2 Fine-Grained Complexity

2.1 SAT, ETH and SETH

In the history of complexity theory, one of the most common hardness assumption is $P \neq NP$. We need this assumption for arguing there is no polynomial time algorithm for various problems. However, people usually need stronger assumptions to show the hardness for problems even in P. We will show the connections between several famous such assumptions using reductions.

First, we give the definition of the CNF-SAT problem (clausal normal form satisfiability), which is the first one proven to be NP-complete.

A propositional logic formula is built by variables $x_1, x_2, \ldots, x_n$, operations AND ($\wedge$), OR ($\vee$), NOT ($\neg$) and parentheses. A formula is said to be satisfiable if it can be made TRUE by assigning appropriate logical values (i.e. TRUE, FALSE) to its variables. The Boolean satisfiability problem (SAT) is, given a formula, to check whether it is satisfiable. A literal is either a variable x_i, or its negation $\neg x_i$. A clause is a disjunction ($\vee$) of literals (or a single literal). A formula is in clausal normal form (**CNF**) if it is a conjunction ($\wedge$) of clauses (or a single clause). For example,

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_4) \wedge (\neg x_3)$$

is a CNF formula, and it is satisfiable by the input $(x_1, x_2, x_3, x_4) = (T, F, F, T)$.

If every clause has exactly k literals, we call the corresponding SAT problem "k-SAT". Denote the number of variables by n, and the number of clauses by m. We assume $m = \text{poly}(n)$. In the literature, we have the following results:

Theorem 1. *2-SAT$\in P$, but k-SAT is NP-hard for every $k \geq 3$.*

By this result, pick $k = 3$, then there is no polynomial time deterministic algorithm for solving 3-SAT assuming $P \neq NP$. The best known upper bound for 3-SAT is due to Hansen et al. [HKZZ19], which achieves $O(1.307^n)$. For general k-SAT, the best known upper bound is $2^{(1-\frac{O(1)}{k})\cdot n}$, by [PPSZ05].

On the opposite side, two hardness assumptions are proposed:

Conjecture 2 (Exponential Time Hypothesis)**.** *There exists $s > 0$ such that 3-SAT can not be solved in $O(2^{sn})$ time.*

Conjecture 3 (Strong Exponential Time Hypothesis)**.** *$\forall \epsilon > 0$, there exists $k \in \mathbb{Z}$, such that k-SAT can not be solved in time $O(2^{(1-\epsilon)n})$.*

Showing SETH implies ETH is not trivial, we will give a road map about this reduction. A typical trick to transform k-SAT to 3-SAT is that for each clause:

$$(x_1 \vee x_2 \vee \cdots \vee x_k), \tag{1}$$

we introduce auxiliary variables $y_1, y_2, \ldots, y_{k-3}$ and write a new formula:

$$(x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee y_3) \wedge \cdots \wedge (\neg y_{k-4} \vee x_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee x_{k-1} \vee x_k). \tag{2}$$

(1) and (2) are equisatisfiable, i.e., one is satisfiable if and only if the other is. However, for an n variables k-SAT, by the above transform, we create a 3-SAT with $N = n + m(k-3)$ variables, which does not give the correct reduction (SETH$\Rightarrow$ETH) when $m = \omega(n)$.

The following sparsification lemma overcomes the above issue:

Lemma 4 ([IP01])**.** *$\forall \delta > 0$ and $\forall k$, there is an algorithm that runs in time $O(2^{\delta n})$, which takes a k-CNF f as input, and outputs k-CNF formulas $f_1, f_2, \ldots, f_{2^{\delta n}}$ such that*

$$f \text{ is satisfiable} \iff \text{at least one of } f_i \text{ is satisfiable},$$

and each f_i has at most cn clases, where $c = (\frac{k}{\delta})^{O(k)}$.

Proposition 5. *SETH$\Rightarrow$ETH.*

Proof. Suppose we can solve 3-SAT in time $O(2^{sn})$ for arbitrary small $s > 0$.

For any k, and any k-CNF f, we can transform f to $f_1, f_2, \ldots, f_{2^{\delta n}}$ by Lemma 4 such that each k-CNF has at most cn clauses. Then, we further transform each f_i to g_i by the trick (2). Notice that each g_i has $N \leq n + cn(k-3)$ variables, we can check if g_i is satisfiable in time $O(2^{sN})$, thus we can check if there exists a satisfiable f_i in time $O(2^{\delta n + s(ck+1)n})$. Pick $\delta = \frac{1}{2}$ and then pick $s < \frac{1}{2(ck+1)}$, then we can check if f is satisfiable in time $2^{(1-\epsilon)n}$ for some $\epsilon > 0$, which let SETH fail. □

2.2 SETH and OVC

We now give some problems in P that we can derive hardness results using SETH. The first such problem, which has been introduced before, is Orthogonal Vectors (OV). Here is the hardness result we have:

Conjecture 6 (Orthogonal Vectors Conjecture)**.** *For $\forall \epsilon > 0$, when $d = \omega(\log n)$, there is no $O(n^{2-\epsilon})$ time algorithm for OV with $2n$ vectors of dimension d.*

OVC can be shown assuming SETH:

Proposition 7. *SETH$\Rightarrow$OVC. More specifically, if OV can be solved in time $O(n^{2-\epsilon})$, then k-SAT can be solved in time $O(2^{(1-\epsilon/2)n} \cdot \text{poly}(n))$.*

Proof. The idea is to reduce k-SAT to OV and use the hardness assumption for k-SAT. Fix k, for a k-SAT instance with n variables $x_1, x_2, \dots, x_n$, we split them into

$$S_1 = \{x_1, \dots, x_{\frac{n}{2}}\}, S_2 = \{x_{\frac{n}{2}+1}, \dots, x_n\}.$$

Denote the clauses by $C_1, C_2, \dots, C_m$, then there are $2^{n/2}$ different assignments for each subset:

$$a : S_1 \to \{\text{True, False}\}^{n/2},$$

$$b : S_2 \to \{\text{True, False}\}^{n/2}.$$

For each assignment, we make vectors $v_a, u_b \in \{0,1\}^m$ such that

$$v_a[i] = \begin{cases} 0, & \text{if } a \text{ satisfies } C_i \text{ no matter how } S_2 \text{ is assigned,} \\ 1, & \text{otherwise.} \end{cases}.$$

$$u_b[i] = \begin{cases} 0, & \text{if } b \text{ satisfies } C_i \text{ no matter how } S_1 \text{ is assigned,} \\ 1, & \text{otherwise.} \end{cases}.$$

Then we obtain

$$< v_a, u_b >= 0 \iff \text{The assignment } a \to S_1 \text{ and } b \to S_2 \text{ is satisfiable.}$$

In fact, $< v_a, u_b >= 0$ if and only if for every i, at least one of $v_a[i]$ and $u_b[i]$ is 0. That is to say, for every clause C_i, at least one of a and b makes C_i satisfiable, this means the whole formula is satisfiable.

Now suppose OV can be solved in time $(2^{n/2})^{2-\epsilon} = 2^{n-(n\epsilon/2)}$, then k-SAT can be solved in time

$$O(m \cdot 2^{\frac{n}{2}} + 2^{n-(n\epsilon/2)}) = O(2^{(1-\epsilon/2)n} \cdot \text{poly}(n)),$$

which contradicts with SETH. □

In fact, the current best algorithm for SAT is achieved by reducing to OV, and its running time is $2^{(1-\frac{1}{O(\log MN)})N}$ for instances with N variables and M clauses.

2.3 Closest Pair and Nearest Neighbor Search

We introduce another two problems whose hardness are related to SETH.

Closest Pair: Given $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n \in \{0,1\}^d$, and given $k \in \mathbb{Z}_+$, find $i, j \in [n]$ such that $||x_i - y_j||_1 \le k$.

Nearest Neighbor Search: Given $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n \in \{0,1\}^d$, and given $k \in \mathbb{Z}_+$, for every $i \in [n]$, find a $j \in [n]$ such that $||x_i - y_j||_1 \le k$.

Intuitively, NNS is harder than CP since solving NNS in particular solves CP. In the last lecture, we have seen that, assuming SETH, CP can not be solved in time $O(n^{2-\epsilon})$, thus NNS can not be solved in this time either. In fact, the following proposition tells us exactly the same hardness will happen in NNS:

Proposition 8. *If one can solve CP in time $O(n^{2-\epsilon})$ for some $\epsilon > 0$, then one can also solve NNS in time $O(n^{2-\epsilon/2})$.*

Proof. Partition the inputs into $\sqrt{n}$ groups of size $\sqrt{n}$:

	$y_1, \ldots, y_{\sqrt{n}}$	$y_{\sqrt{n}+1,\ldots,y_{2\sqrt{n}}}$	$\ldots$	$y_{n-\sqrt{n}+1}, \ldots, y_n$
$x_1, \ldots, x_{\sqrt{n}}$	M_{11}	M_{12}	$\ldots$	$M_{1,\sqrt{n}}$
$x_{\sqrt{n}+1}, \ldots, x_{2\sqrt{n}}$	M_{21}	M_{22}	$\ldots$	$M_{2,\sqrt{n}}$
$\ldots$				
$x_{n-\sqrt{n}+1}, \ldots, x_n$	$M_{\sqrt{n},1}$	$M_{\sqrt{n},2}$	$\ldots$	$M_{\sqrt{n},\sqrt{n}}$

Fix k, and start from M_{11}, call the oracle of CP to find the closest pair from the first two groups. If there exists such pairs (x_i, y_j), we output the nearest neighbor (x_i, y_j) and remove x_i from the first group. After we remove all such x from the first group, we move to M_{12} and call the oracle of CP, and so on. This procedure terminates when we find the nearest neighbor to x_i for all $i \in [n]$. Notice that we use the CP oracle $2n$ times, thus if each oracle costs $\sqrt{n}^{2-\epsilon} = n^{1-\frac{\epsilon}{2}}$, we can solve NNS in time $2n \cdot n^{1-\epsilon/2} = 2n^{2-\epsilon/2}$. ☐

References

[HKZZ19] Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. Faster k-sat algorithms using biased-ppsz. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 578–589, 2019.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[PPSZ05] Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.