CS 354: Unfulfilled Algorithmic Fantasies May 29, 2019

Lecture 17: The Strong Exponential Time Hypothesis

Lecturer: Joshua Brakensiek *Scribe: Weiyun Ma*

1 Introduction

We start with a recap of where we are. One of the biggest hardness assumption in history is P $\neq$ NP, which came around in the 1970s. This assumption is useful for a number of hardness results, but often people need to make stronger assumptions. For example, in the last few weeks of this course, we have used average-case hardness assumptions (e.g. hardness of random 3-SAT or planted clique) to obtain distributional results.

Another kind of results have implications on hardness of approximation. For instance, the PCP Theorem is known to be equivalent to solving some approximate 3-SAT is hard unless P=NP. We have also seen conjectures in this category that are still open, even assuming P $\neq$ NP, such as the Unique Games Conjecture and the 2-to-2 Conjecture.

Today, we are going to focus on hardness assumptions on time complexity. For example, for 3-SAT, P $\neq$ NP only implies that it cannot be solved in polynomial time. On the other hand, the best known algorithms run in exponential time in n (the number of variables), which motivates the hardness assumption that better algorithms do not exist. This leads to conjectures known as the **Exponential Time Hypothesis (ETH)** and a stronger version called the **Strong Exponential Time Hypothesis (SETH)**.

Informally, ETH says that 3-SAT cannot be solved in $2^{o(n)}$ time, and SETH says that k-SAT needs roughly 2^n time for large k. We will discuss the connections between ETH and SETH and derive a few hardness results assuming SETH.

2 ETH, SETH, and the Sparsification Lemma

Throughout the rest of the lecture, we use the following notation:

- k is the width of a SAT formula. We assume k is constant.
- n is the number of variables.
- m is the number of clauses. We assume $m \sim \text{poly}(n)$.

We define

$$s_k = \inf\{\delta : k\text{-SAT can be solved in } 2^{\delta n}\text{poly}(m) \text{ time}\}.$$

Observe that $s_2 = 0$ since 2-SAT is in P. For $k = 3$, the following algorithm shows that $s_3 < 1$: We start with a clause in the formula, say involving variables x_1, x_2, x_3, and branch on the 7 possible assignments of x_1, x_2, x_3 that make the clause true. We substitute the partial assignment to the rest of the formula, find the next clause with three unassigned variables, and branch again on the 7 possible assignments of the three variables there. Repeating this process, we are left with an instance of 2-SAT in the end, which can be solved in polynomial time. Note that in the original

formula, there are at most $\frac{n}{3}$ clauses such that no two of them involve a common variable. Thus the running time of the algorithm is at most $7^{n/3}\text{poly}(m) = 2^{(\log_2 7)n/3}\text{poly}(m)$. This means that $s_3 \leq \frac{\log_2 7}{3} < 1$.

Formally, ETH is the following hardness assumption, first proposed by Impagliazzo and Paturi in 2001 [1]:

Assumption 2.1 (ETH [1])**.** *There exists k such that $s_k > 0$.*

That is, solving k-SAT needs exponential time for some k. Impagliazzo and Paturi [1] were able to show the following result:

Proposition 2.2 ([1])**.** *ETH (2.1) is equivalent to $s_3 > 0$.*

The implication ($\Leftarrow$) is obvious. Here is an attempt to prove the implication ($\Rightarrow$): We can turn any k-SAT formula into a 3-SAT formula by introducing new variables. For example, the satisfiability of a clause $(x_1 \vee \cdots \vee x_k)$ is equivalent to the satisfiability of

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

Starting with a k-SAT formula with n variables and m clauses, we then obtain a 3-SAT formula with $N = n + m(k-3)$ variables. Suppose $s_3 = 0$ and we can solve the 3-SAT formula in $2^{o(N)}$ time. This implies that we can solve the original k-SAT formula in $2^{o(n+m(k-3))}$ time. However, since m can be $\omega(n)$, this is not enough for us to conclude that the k-SAT formula is solvable in $2^{o(n)}$ and derive a contradiction to $s_k > 0$.

The above argument would work if we have $N = O(n)$. This is where the following lemma kicks in:

Lemma 2.3 (Sparsification Lemma [1])**.** *Let f be a k-SAT formula f. For any $\epsilon > 0$, there exists an algorithm that outputs formulae $f_1, \ldots, f_{2^{\epsilon n}}$ in $2^{\epsilon n}$ time such that:*

1. *Each f_i is a k-SAT formula on the same variables as f.*
2. *There exists a constant $c = c(\epsilon, k)$ such that any variable x_j appears at most c times in any formula f_i.*
3. *f is satisfiable if and only if at least one f_i is satisfiable.*

Let us use Lemma 2.3 to prove the implication ($\Rightarrow$) of Proposition 2.2. For a k-SAT formula f, we sparsify it into the f_i's using Lemma 2.3. We can then turn each f_i into a 3-SAT formula g_i using the above method. Observe that since each variable appears at most a constant number of times in f_i, the number N_i of variables in g_i is at most linear in n. Now suppose $s_3 = 0$ and we can solve each g_i in $2^{o(N_i)} = 2^{o(n)}$ time. Then, the total time for solving f is $2^{\epsilon n}2^{o(n)}$. Since ϵ can be taken to be arbitrarily small, it follows that $s_k = 0$, which is a contradiction.

For the rest of the lecture, we will focus on SETH, which is also proposed in [1].

Assumption 2.4 (SETH [1])**.**

$$\lim_{k\to\infty} s_k = 1.$$

In fact, our current best algorithms for k-SAT give us

$$s_k \leq 1 - \frac{\Theta(1)}{k}.$$

SETH roughly conjectures that no algorithms can beat this bound other than possibly reducing the k factor to $\log k$, etc.

3 Orthogonal Vectors

We now discuss some examples that use SETH (2.4) to derive hardness results for problems that are solvable in polynomial time (!). The first such problem is **Orthogonal Vectors (OV)**: given two sets A, B of $\{0,1\}$-vectors in a d-dimensional space, do there exist $a \in A, b \in B$ such that $\langle a, b\rangle = 0$?

For simplicity, we assume that $|A| = |B| = N$, and $d = \Theta(\log^2 N)$. (In fact, it is sufficient to assume $d = \omega(\log N)$; otherwise the sets A, B would either have repetitions or be the whole space $\{0,1\}^d$ of vectors.) We have a $O(N^2 d)$ baseline algorithm that checks the inner products of all possible pairs of vectors. The following result of Williams [3] shows that, if we assume SETH, then we cannot find algorithms with much better time complexity:

Theorem 3.1 ([3])**.** *Assume SETH (2.4). Then for all $\epsilon > 0$, solving OV needs $\Omega(N^{2-\epsilon})$ time.*

Proof. The idea is to reduce k-SAT to OV and use the hardness assumption SETH on k-SAT to show that OV is also hard. We fix k and take a k-SAT formula f on n variables $x_1, \ldots, x_n$ with m clauses $C_1, \ldots, C_m$. We will define an injective map $a : \{0,1\}^{n/2} \to \{0,1\}^m$ which embeds all possible assignments of the variables $x_1, \ldots, x_{n/2}$ as a set A of vectors in $\{0,1\}^m$; similarly, we will define an injective map $b : \{0,1\}^{n/2} \to \{0,1\}^m$ which embeds all possible assignments of the variables $x_{n/2+1}, \ldots, x_n$ as a set B of vectors in $\{0,1\}^m$. Most importantly, we will make the maps a and b satisfy the following condition:

$$\langle a(\alpha), b(\beta)\rangle = 0 \Leftrightarrow \text{partial assignments } \alpha \text{ and } \beta \text{ together satisfy } f, \tag{1}$$

which reduces the k-SAT instance f to an instance of OV.

Let us first explain why condition (1) allows us to prove the theorem. Let $N = 2^{n/2} = |A| = |B|$ and $d = m \approx n \approx \log N$. (Here, we only consider the case where $m = \Omega(n)$. For the case $m = o(n)$ there are much faster algorithms for k-SAT.) For $\epsilon > 0$, suppose we have an $O(N^{2-\epsilon})$ algorithm for OV. The above reduction then implies that k-SAT can be solved in $O(2^{(2-\epsilon)n/2}) = O(2^{(1-\epsilon/2)n})$ time. Since k does not depend on ϵ, we have shown that $s_k < 1 - \frac{\epsilon}{2}$ for all k. This contradicts SETH (2.4).

It remains to define the embeddings a and b that satisfy condition (1). For all $i = 1, \ldots, m$, we set

$$a(\alpha)_i = \begin{cases} 0 & \text{if } C_i \text{ is satisfied by the partial assignment } \alpha, \\ 1 & \text{otherwise;} \end{cases}$$

$$b(\beta)_i = \begin{cases} 0 & \text{if } C_i \text{ is satisfied by the partial assignment } \beta, \\ 1 & \text{otherwise.} \end{cases}$$

Now $\langle a(\alpha), b(\beta)\rangle = 0$ if and only if every clause C_i is satisfied by either α or β, which is equivalent to that α and β together satisfy f. □

4 Nearest Neighbor

Now we turn to the geometric problem of the **Nearest Neighbor (NN)**: for a set of points A in some space and a query point b, find the point a in A that is closest to b and the distance between a and b. We might have a set B of query points to be processed. We will discuss two settings of NN. In the **offline NN**, the algorithm is given both the initial set A and the query points B at once and asked to answer all queries in B. In the **online NN**, the algorithm is given the set A at first to preprocess and then asked to answer incoming queries one at a time as fast as possible.

The offline NN problem is related to the simpler problem of finding the **Bichromatic Closest Pair (BCP)**: given two sets A and B of points in some space (as above), find $a \in A$ and $b \in B$ such that $\|a - b\|$ is as small as possible. (We will mostly be working with the hypercube $\{0,1\}^d$ endowed with the ℓ_1-norm.) Note that solving offline NN in particular solves BCP. Thus any lower bound on the complexity of BCP is a lower bound on the complexity of offline NN.

Suppose $|A| = |B| = N$. The following result of Williams [3] gives a lower bound on the time complexity of BCP assuming SETH:

Theorem 4.1 ([3])**.** *Assume SETH (2.4). Then for all $\epsilon > 0$, solving BCP needs $\Omega(N^{2-\epsilon})$ time.*

As a consequence, solving offline NN also needs $\Omega(N^{2-\epsilon})$ time if we assume SETH.

Proof. The idea is to reduce OV to BCP and use the hardness of OV assuming SETH (see Theorem 3.1). Take two sets of vectors $A, B \subset \{0,1\}^d$ with equal size $N = |A| = |B|$. We assume that $d = O(\log^2 N)$. We have the following relation: for $a, b \in \{0,1\}^d$,

$$\|a - b\|_1 = \|a\|_1 + \|b\|_1 - 2\langle a, b\rangle. \tag{2}$$

For $b \in \{0,1\}^d$, denote $\bar{b} = \bar{1} - b$, where $\bar{1}$ is the vector whose entries are all 1. Using (2), we have that for $a, b \in \{0,1\}^d$,

$$\|a - \bar{b}\|_1 = \|a\|_1 + \|\bar{b}\|_1 - 2\langle a, \bar{b}\rangle = \|a\|_1 + \|\bar{b}\|_1 - 2(\|a\|_1 - \langle a, b\rangle) = \|\bar{b}\|_1 - \|a\|_1 + 2\langle a, b\rangle. \tag{3}$$

Thus, if we partition A and B into the subsets

$$A_i = \{a \in A : \|a\|_1 = i\}, \qquad B_i = \{b \in B : \|\bar{b}\|_1 = i\}$$

and set

$$\bar{B}_i = \{\bar{b} : b \in B_i\},$$

(3) then implies that for any $a \in A_i$ and $\bar{b} \in \bar{B}_j$, we have

$$\|a - \bar{b}\|_1 = j - i + 2\langle a, b\rangle \geq j - i. \tag{4}$$

Here, the inequality follows from the nonnegativity of the inner product, and equality holds if and only if $\langle a, b\rangle = 0$.

Therefore, to find $a \in A, b \in B$ with $\langle a, b\rangle = 0$, we can run BCP on the pairs of sets A_i and $\bar{B}_j$ for all $i, j \in \{0, \ldots, d\}$ and check whether equality in (4) is ever achieved among the norm-minimizing pairs of vectors. If we have an $o(N^{2-\epsilon})$ algorithm for BCP, the above gives us an $o((d+1)^2 N^{2-\epsilon}) = o(N^{2-\epsilon} \log^4 N)$ algorithm for OV, which contradicts Theorem 3.1. □

Theorem 4.1 can be used to deduce the following hardness result for online NN, due to Vassilevska Williams and Williams [4] (proof as presented in Rubinstein [2]).

Theorem 4.2 ([4])**.** *Assume SETH (2.4). Let $\delta, c > 0$. Assume an algorithm is allowed $O(N^c)$ time to preprocess A. It still needs $\Omega(N^{1-\delta})$ time to answer each online NN query $b \in B$.*

Proof. The idea is to reduce offline NN to online NN and use the hardness of offline NN assuming SETH (see the comment after Theorem 4.1). Take sets A, B as before. Pick the parameter $\gamma = \frac{1}{2c}$. We partition A into $N^{1-\gamma}$ subsets $A_1, \ldots, A_{N^{1-\gamma}}$ such that each A_i has N^γ elements. We can preprocess each A_i in $O((N^\gamma)^c) = O(N^{1/2})$ time, and the total preprocessing time is $O(N^{3/2-\gamma})$. Now for each $b \in B$, we can treat it as an online query for each A_i and take the best result.

Suppose we have an online NN algorithm that runs in $o(N^{1-\delta})$ time per query. Then the above gives us an offline NN algorithm that has runtime

$$O(N^{3/2-\gamma}) + o(N \cdot N^{1-\gamma} \cdot (N^\gamma)^{1-\delta}) = o(N^{2-\delta\gamma}),$$

which contradicts the hardness of offline NN assuming SETH. □

References

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

[2] Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 1260–1268, New York, NY, USA, 2018. ACM.

[3] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

[4] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 645–654, 2010.