CSE 531: Computational Complexity I **Winter 2016**

Lecture 17: The Strong Exponential Time Hypothesis

Mar 4, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

1 The precise complexity of SAT

For SAT and $CIRCUIT\text{-}SAT$ there is an obvious brute force algorithm that requires $2^n m^{O(1)}$ time where n is the number of variables and m is a measure of the overall input size. How much better we can seems to depend on the input structure. In the case of $kSAT$ there are some improvements.

Theorem 1.1 (Paturi-Pudlak,Schöning,Paturi-Pudlak-Saks-Zane,Hertli)**.** *For constants $1 \leq c \leq 2$, there are (randomized) algorithms for n variable k-SAT that run in time $2^{n(1-c/k)}$.*

Proof Sketch. The Paturi-Pudlak algorithm is of the following simple format:
Repeat:
Randomly guess an assignment variable-by-variable in random order; in this process, set any variable that is forced because it is the last one remaining in some clause.

Intuitively, a variable is likely to be forced every k-th step so the number of variables that need to be guessed is only $n(1-1/k)$ and hence the probability of success is at least $2^{-n(1-1/k)}$. (Strictly speaking this argument is only true for $USAT$, but the analysis can be extended to the general case.

Schöning's algorithm is similarly simple:

Repeat:
    Randomly guess a total assignment.
    Repeat for $O(n)$ steps:
        Choose some violated clause and flip a random variable to satisfy the clause.

This yields a biased random walk that still has a significantly better than 2^{-n} probability of finding a satisfying assignment.

The PPSZ algorithm is an extension of the Paturi-Pudlak algorithm using initial resolution steps up to a certain constant size. They only showed how to get this to work for $USAT$. Hertli showed how to extend their analysis to the general case. □

These are the best upper bounds known for k-SAT. The fact that the improvements for $kSAT$ and other NP-complete problems have been at best in the constants in the exponent led Impagliazzo and Paturi to make the following conjecture.

Exponential Time Hypothesis (ETH) There are constants $s_k > 0$ such that the time complexity of n variable $kSAT$ is at least $2^{s_k n}$.

By standard reductions it is equivalent to state the ETH solely for $k = 3$. Moreover, Impagliazzo, Paturi and Zane showed that ETH implies that a vast number of standard NP-complete require $2^{\Omega(n)}$ time. For example,

Theorem 1.2. *ETH implies that $3COLOR$ requires time $2^{\Omega(n)}$ time.*

The reason that this is not obvious is that the standard reduction from $3SAT$ to $3COLOR$ converts a formula with n variables and m clauses to a graph of size $O(n + m)$. A lower bound of $2^{\Omega(n)}$ for $3SAT$ will not be $2^{\Omega(n+m)}$ if m is super-linear and all we know *a priori* is that m is $O(n^3)$. However, one can show that the ETH also holds for k-CNF formulas in which m is only $O(n)$ which immediately implies the above theorem using the standard reduction. The following is a later strengthening of a similar lemma that is due to Capalbo, Impagliazzo, and Paturi.

Lemma 1.3 (Sparsification Lemma)**.** *For any k-CNF formula φ with n-variables and m clauses and any $0 < \varepsilon < 1$ there is a $2^{\varepsilon n} m^{O(1)}$ time algorithm that can find k-CNF formulas $\varphi_1, \ldots, \varphi_{2^{\varepsilon n}}$ such that $\varphi = \bigvee_i \varphi_i$ and each φ_i has $m = O((k/\varepsilon)^k n)$.*

For sufficiently small $\varepsilon > 0$ a general $kSAT$ algorithm can simply produce the various φ_i and test the satisfiability of each of them. This means that the exponent in the general case is at most εn larger than in the case of linear size formulas so the ETH applies to formulas for which m is $O(n)$.

The proof of the Sparsification Lemma involves branching based on subsets of variables that occur significantly more frequently than expected. The proof is a subtle potential argument.

The fact that the best algorithms known for $kSAT$ have exponents that approach n as k increases led Impagliazzo and Paturi to make the following stronger conjecture.

The Strong Exponential Time Hypothesis (SETH) For every $\varepsilon > 0$ there is a k such that $kSAT$ requires time larger than $2^{(1-\varepsilon)n}$. That is, $\lim_k \to \infty s_k = 1$.

In particular, SETH implies that $k(n)$-SAT for any function $k(n)$ that is $\omega(1)$ requires time $2^{n-o(n)}$ and this lower bound also obviously applies directly to $CNFSAT$.

Unlike the ETH, SETH is not known to imply similar bounds for other NP-complete problems.

We will also consider less aggressive versions of SETH that apply to forms of $CIRCUIT\text{-}SAT$. This is even more believable given how small the improvements are for even slightly more general classes of circuits. For example, the best known satisfiability algorithms for AC^0 circuits is the following bound due to Impagliazzo, Matthews and Paturi:

Theorem 1.4. *Satisfiability for depth unbounded fan-in depth d AC^0 circuits can be solved in time time $2^{n-O(n/\log^{d-1} n)}$.*

Note that in the case $d = 2$ corresponds to $CNFSAT$, and the savings is $O(1/\log n)$. This comes from the fact that we can essentially assume that clauses have length $O(\log n)$ since longer clauses will be very easy to satisfy.

For $\mathsf{ACC}^0 = \bigcup_m \mathsf{AC}^0[m]$ circuits, the best savings known, which is due to Williams, is even smaller.

Theorem 1.5. *Satisfiability for ACC^0 circuits can be solved in time $2^n/n^c$ for some constant $c > 0$.*

This satisfiability algorithm is the basis for Williams' theorem that $\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$. Indeed, he has shown that any such improvement for a circuit class immediately implies a similar consequence for that class of circuits.

2 SETH and Polynomial Time Problems

We will see that SETH has implications for the exact complexity for polynomial-time problems. The most basic such implication was shown by Williams in 2004.

Orthogonal Vectors The *Orthogonal Vectors* problem ($OV_{N,m}$), is the following:
Given $y_1, \ldots, y_N, z_1, \ldots, z_n \in \{0,1\}^m$, is there a pair (i, j) such that $y_i \cdot z_j = 0$ over the integers?
If we view each element of $\{0,1\}^m$ as a subset of $\{1, \ldots, n\}$, then orthogonal vectors is an all-pairs set-disjointness problem that asks whether there is some pair y_i and z_j that y_i and z_j are disjoint.

There is an obvious $N^2 m$ time algorithm for $OV_{N,m}$. The following theorem shows that SETH implies that this cannot be significantly improved.

Theorem 2.1. *SETH implies that for every $\delta > 0$, there is no $O(N^{2-\delta})$ algorithm for $OV_{N,m}$ for $m = \omega(\log N)$.*

Note that this is not far from the upper bound which is less than $N^2 \log^2 n$ for some of the values of m to which it applies.

Proof. Given a CNF formula φ with m clauses and n variables, let $N = 2^{n/2}$. Let $C_1, \ldots, C_m$ be the clauses of φ. Let $Y = \{x_1, \ldots, x_{n/2}\}$ and $Z = \{x_{n/2+1}, \ldots, x_n\}$. For each assignment $a \in \{0,1\}^Y$ define the i-th coordinate of y_a by

$$y_a(i) = \begin{cases} 0 & \text{if } a \text{ satisfies } C_i \\ 1 & \text{if not.} \end{cases}$$

Similarly each assignment $b \in \{0,1\}^Z$ define the j-th coordinate of z_b by

$$z_b(j) = \begin{cases} 0 & \text{if } b \text{ satisfies } C_j \\ 1 & \text{if not.} \end{cases}$$

Therefore

$$\begin{aligned} \exists (a,b) \text{ satisfying } \varphi &\Leftrightarrow \exists (a,b)\ \forall i,\ (a,b) \text{ satisfies } C_i \\ &\Leftrightarrow \exists (a,b)\ \forall i,\ a \text{ satisfies } C_i \text{ or } b \text{ satisfies } C_i \\ &\Leftrightarrow \exists (a,b)\ \forall i,\ y_a(i) = 0 \text{ or } z_b = 0 \\ &\Leftrightarrow \exists (a,b)\ y_a \cdot z_b = 0 \end{aligned}$$

Therefore $y_1, \ldots, y_N, z_1, \ldots, z_N$ are orthogonal vectors iff φ is satisfiable.

Since m is $\omega(\log N)$, m is $\omega(n)$. Let $\ell = \ell(n) = m/n$ which is $\omega(1)$. We can then choose $k = k(n)$ that is $\omega(1)$ such that $\ell(n) = m/n$ is larger than the ratio $c(k/\varepsilon)^k$ given by the Sparsification Lemma.

Now if there is an algorithm for $OV_{N,m}$ that has running time $O(N^{2-\delta})$ then, by the above reduction, we would obtain an algorithm we would obtain an algorithm for $k(n)$-SAT with running time $O(N^{2-\delta}) = O((2^{n/2})^{2-\delta}) = O(2^{n(1-\delta/2)})$ which contradicts SETH. □

Now Orthogonal Vectors did not seem such a critical problem and the connections in this area did not receive much attention until a 2010 paper by Patrascu and Williams. Since then, and especially in the last couple of years a number of natural problems have been shown to have the property that the best known algorithms cannot be improved without violating SETH.

At STOC 2015, Backurs and Indyk showed that one such problem is Edit Distance. Later at FOCS 2015, Abboud, Backurs, and Vassilevska-Williams, and Bringman and Kunneman extended this to finding the length of the *Longest Common Subsequence (LCS)*, with the latter showing that this hardness extends to the case of the binary alphabet. LCS is equivalent to a special case of Edit Distance in which the cost of insertion or deletion is 1 and the cost of substitution is 2. More formally, given strings $A, B \in \Sigma^*$ define $LCS(A,B)$ to be the maximum k such that there are sequences $i_1 < \ldots < i_k$ and $j_1 < \ldots < j_k$ for which $A_{i_1} = B_{i_1}, \ldots, A_{i_k} = B_{j_k}$.

There are simple natural dynamic programming algorithm for Edit Distance (and hence LCS) that run in time $O(n^2)$ where $|A| = |B|$ and the best algorithms known only shave off a $\log n$ factor.

Theorem 2.2. *SETH implies that LCS over binary strings does not have an $O(n^{2-\delta})$ algorithmn for any $\delta > 0$.*

The proof of this begins by looking at a problem closer to OV. Define $LCS\text{-}PAIR_{N,m}$ to be the problem: Given sequences $a_1, \ldots, a_N, b_1, \ldots, b_N \in \Sigma^m$, find the $\max_{i,j} LCS(a_i, b_j)$. We describe the reduction from $OV_{N,m}$ to LCS-Pair along the lines given by Bringman and Kunneman.

Lemma 2.3. *$OV_{N,m}$ reduces in linear time to $LCS\text{-}PAIR_{N,cm}$ for some constant c.*

Proof. The general idea is to produce a local substitution of each character of y_i and z_j according to different substitutions.

Define the strings $0_y = 10011$, $1_y = 11100$, $0_z = 11001$, and $1_z = 00111$. Observe that $LCS(0_y, 0_z) = LCS(0_y, 1_z) = LCS(1_y, 0_z) = 4$ but $LCS(1_y, 1_z) = 3$.

We want to ensure that any LCS for a_i and b_j involves a character-by-character match of y_i and z_j in $\{0,1\}^m$. To do this we define $code_y(y_i)$ to be the string where we replace each 0 or 1 of y_i by the corresponding 0_y or 1_y and we separate each pair by, say, a string of three 2's; do the same for $code_z(z_j)$ except we use 0_z and 1_z instead. Define $a_i = code_y(y_i)$ and $b_j = code_z(z_j)$. In particular, if $y_i = 001$ and $z_j = 011$ then

$$a_i = code_y(001) = 100111222100112221100$$
$$b_j = code_z(011) = 110012222001112220011$$

Therefore the total string length m' is $8m - 3$. It is clear that every LCS of a_i and b_j and must match all the 2's, which means that corresponding coordinates much be matched. If y_i and z_j are orthogonal then $LCS(a_i, b_j) = 4m + 3(m-1) = 7m - 3$, which we denote by S. On the other hand, if y_i and z_j are not orthogonal then the contribution is only 3 instead of 4 in all the coordinates with common 1's and hence $LCS(a_i, b_j) \leq S - 1$. The reduction simply compares the length of the LCS to S. □

To obtain a lower bound for LCS, we will need to concatenate these strings to a single pair of strings A and B so that the length of the LCS of the whole string will be larger iff there is *some* pair of orthogonal y_i and z_j. To do this we need to control things so that we know the contribution of each the failed matches also. With the above construction, the more overlapping 1's, the worse the value. To fix this we use a slight modification of the above construction that always guarantees a match of precisely 1 less than the maximum possible.

To do this we add an extra dummy coordinate to the encoding. Define $code'_y(y_i) = code(y_i, 0)$ and $code'_z(z_j) = code(z_j, 1)$ as well as an extra "easy string" $E = code(0^m, 1) =$. Observe that $(y_i, 0)$ and $(z_j, 1)$ are orthogonal iff y_i and z_j are. Also every vector $(z_j, 1)$ has precisely one coordinate

with overlapping 1's with $(0^m, 1)$. Let $m'' = 8m + 5$ be the length of $code'_y(y_i)$. Now define

$$a_i = code'_y(y_i)3^{m''}E$$
$$b_j = 3^{m''}code'_z(z_j)3^{m''}$$

Observe that any LCS for the two strings must match one of the two groups of 3's in b_j in its entirely to the middle group of 3's in a_i and then include an LCS between $code'_z(z_j)$ and either $code'_y(y_i)$ of E. If y_i and z_j are orthogonal then we get a total contribution of $S' = m" + S + 7$ where S is the value from the above lemma since the extra coordinate gives a total contribution of 7. In the latter case there is precisely 1 segment where the contribution is 3 instead of 4 so we get $m" + S + 6 = S' - 1$ (there is a contribution of 3 for the matching 2's and another 3 from the LCS between 1_y and 1_z).

We now are in a position to describe the construction of the strings A and B for the proof of the theorem. The idea will be that the contribution will be to allow an arbitrary rotated alignment of the combined string of encodings $b_1, \ldots, b_N$, suitably separated, with the the string of encodings $a_1, \ldots, a_N$. This will necessitate two copies of one of the strings to allow for the orthogonal pair (y_i, z_j) to have $i > j$ as well as $i \leq j$.

Define the strings

$$A = a_1 4^\ell a_2 4^\ell \ldots 4^\ell a_N 4^\ell a_1 4^\ell a_2 4^\ell \ldots 4^\ell a_N$$
$$B = 4^{N\ell} b_1 4^\ell b_2 4^\ell \ldots 4^\ell b_N 4^{N\ell}$$

where ℓ is a suitable constant.

The total length of A and B is at most $n = O(N \log^2 N)$ (for $m = \log_2^2 N$ say). Now any LCS can match at most the $(2N-1)\ell$ 4's in A. It is not hard to see that any optimal LCS will align $b_1 4^\ell \ldots 4^\ell b_N$ with some $a_i 4^\ell \ldots 4^\ell a_{i-1}$ (where we write $a_0 = a_N$). If the alignment does not include an orthogonal pairing then the LCS is at most $(2N-1)\ell + N \cdot (S' - 1)$. If the alignment does include an orthogonal pairing them the total LCS is at least $S'' = (2N-1)\ell + N \cdot (S' - 1) + 1$.

Therefore, $OV_{N,\log_2^2 n}$ is reducible in time $O(n) = O(N^2 \log^2 n)$ to LCS on length n strings over $\{0, 1, 2, 3, 4\}$. This proves the theorem except for the reduction of the alphabet size to binary. That reduction requires a more subtle way to put the various strings in order to replace the symbols $2, 3, 4$ by binary strings.

3 LCS is hard even given a very weak SETH

Abboud, Hansen, Vassilevska-Williams, and Williams have a paper at STOC 2016 which shows that LCS is hard even if very high complexity $CIRCUIT\text{-}SAT$ is hard. We sketch a much simpler proof of their theorem.

Theorem 3.1. *If there is no $2^{n-o(n)}$ algorithm to compute satisfiability for*

- *depth $o(n)$ circuits,*
- *size $2^{o(n)}$ Boolean formulas, or*
- *verifiers given by space $o(\sqrt{n})$ nondeterministic Turing machines,*

then binary LCS does not have an $O(n^{2-\delta})$ algorithm for any $\delta > 0$.

We note that by the formula balancing construction we sketched on Monday, the first two classes are identical. The third follows from the first by the fact that $\mathsf{NSPACE}(\mathsf{S}(\mathsf{n})) \subseteq \mathsf{DEPTH}(\mathsf{S}^2(\mathsf{n}))$, which you proved on the midterm.

One key observation is that the previous construction did not make use of the full range of parameters possible in constructing A and B. All we needed for the conclusion is that each a_i for the $LCS\text{-}PAIR$ problem has size $N^{o(1)}$ rather than restricting it to $O(\log^2 N)$.

Proof Sketch. We use the same framework to convert from $LCS\text{-}PAIR$ to LCS so we just give the description for the a_i and b_j strings for $LCS\text{-}PAIR$ and we don't discussion the conversion to binary strings. We use the same separation of the input variables into Y and Z and $N = 2^{n/2}$ assignments to each player as before. (We will index assignments by α and β so as not to confuse notation.) We will define a_α and b_β recursively. We will define a a^v_α and b^v_β for each gate v in the circuit. We will first produce them as weight formulas and then argue that the weights can be removed. We will assume wlog that the depth $o(n)$ circuit has been converted to a balanced formula in which all negations have been pushed to the leaves and each gate at depth k has two predecessors at depth $k-1$.

We create a^v_α and b^v_β by induction on $k = depth(v)$. We will construct these so that $LCS(a^v_\alpha, b^v_\beta)$ is maximal iff v evaluates to 1 on input (α, β).

Suppose that $k = 0$. Then u is labelled by a literal ℓ_i that is either x_i or $\neg x_i$. Set

$$a^u_\alpha = \begin{cases} * & \text{if } i > n/2 \text{ or } \ell_i(\alpha) = 1 \\ \$ & \text{otherwise,} \end{cases}$$

and

$$b^u_\beta = \begin{cases} * & \text{if } i \leq n/2 \text{ or } \ell_i(\beta) = 1 \\ \# & \text{otherwise,} \end{cases}$$

We will ensure that none of the a strings contain $\#$ and none of the b strings contain \$. Clearly both will have a $*$ iff literal ℓ_i is set to true on assignment (α, β).

Now suppose that u has children v and w at depth $k-1$.
If $u = v \wedge w$ then define

$$a^u_\alpha = P_k a^v_\alpha Q_k a^w_\alpha P_k$$
$$b^u_\beta = P_k b^v_\beta Q_k b^w_\beta P_k$$

where P_k and Q_k are new symbols of weight $W_k = 3^k$. Clearly any optimal LCS for this pair must align all of the P_k and Q_k and hence will have a maximal LCS iff both $LCS(a^v_\alpha, b^v_\beta)$ and $LCS(a^w_\alpha, b^w_\beta)$ are maximal.
If $u = v \vee w$ then define

$$a^u_\alpha = P_k a^v_\alpha Q_k a^w_\alpha P_k$$
$$b^u_\beta = Q_k b^w_\beta P_k b^v_\beta Q_k$$

In this second case, the optimal LCS must match precisely one P_k and one Q_k and include the LCS of precisely one of the two pairs (a^v_α, b^v_β) or (a^w_α, b^w_β). Hence it will be maximal iff at least one of the two matches is maximal. (Note that the target weight of the maximal match is different depending on whether the gate is an $\vee$ or an $\wedge$ gates, but this doesn't matter since we know the target size from the property of the circuit itself. We could alternatively simply use higher weight in the OR gadget to make them equal. We end up with $o(n)$ different symbols.

The weighted a_α and b_β strings are the ones for the output gate of the circuit. We can remove all the weights easily by replacing a symbol σ with weight w by w consecutive copies of σ without changing the LCS size. Clearly we can just use entire blocks corresponding the weighted LCS. The claim is that this is the best we can do. Suppose that we have an LCS that does not match things blockwise for σ and consider the leftmost *partial match* of σ between A and B. That, say matches the i-th element in the block in A to the j-th element in the block in B. This (i, j) match splits the LCS. Suppose wlog that $i \leq j$. Clearly that are at most $w - i + 1$ elements of the LCS that touch either of these two blocks since there is no match in the block to the left of the (i, j) match and matches in the LCS cannot cross each other. We can do at least well (obtaining w matched pairs in the blocks) by locally replacing all of those edges by a full matching of the two blocks.

Note that the total weight for depth k is c^k for some constant c, each resulting string has length $2^{o(n)} = N^{o(1)}$. □