1 Why ETH and SETH are defined as they are?

Intuitively, ETH means that 3-SAT cannot be solved in $2^{o(n)}$ time, and SETH that SAT cannot be solved in $\mathcal{O}((2-\varepsilon)^n)$ time. However, the actual hypotheses are stronger assumptions than these layman variants.

Indeed, while $s_3 > 0$ implies that there is no $2^{o(n)}$ time algorithm for 3-SAT, it may well happen that for every $\varepsilon > 0$ there is an $\mathcal{O}(\varepsilon^n)$ time algorithm (hence ETH is false) but there is no single $2^{o(n)}$ time algorithm. Similarly for SETH, it may happen that for every k there is an $\mathcal{O}(1.99^n)$ time algorithm for k-SAT, but there is no $\mathcal{O}(1.99^n)$ time algorithm for SAT with the clause size allowed to grow with the input size.

Do we need these stronger less-likely hypotheses with complicated definitions? Often we do. ETH-based lower bounds usually require bounding the number of clauses and we do not have a variant of Sparsification Lemma that avoids the $2^{\epsilon n}$ overhead. For SETH-based lower bounds, when we need to bound the number of clauses, we need to work with k-SAT, because we do not have a variant of Sparsification Lemma for general SAT.

2 Lower bound for LCS

Recall the Longest Common Subsequence (LCS) problem: Given two sequences A, B of length n find a longest sequence that is a subsequence (i.e., elements need not be consecutive) of A and a subsequence of B.

Exercise 1. What is LCS(educated, coauthor)?

LCS can be solved in $\mathcal{O}(n^2)$ with classical Wagner-Fischer dynamic programming algorithm. Let $DP[i][j] = |LCS(A[1..i], B[1..j])|$, where $X[1..i]$ denotes the prefix of X of length i.

Exercise 2. Give a recursive formula for $DP[i][j]$.

The only improvement over this algorithm to date is due to Masek and Paterson, their algorithm runs in $\mathcal{O}(n^2/\log^2 n)$. Can we solve LCS in *truly subquadratic* time, e.g. $\mathcal{O}(n^{1.99})$? In 2015 Abboud, Backurs and Vassilevska Williams, and independently Bringmann and Künnemann proved that we cannot, unless SETH fails. Below there is a reduction from OV to LCS taken from Karl Bringmann's slides available at `https://conferences.mpi-inf.mpg.de/adfocs-18/karl/1-OV.pdf`.

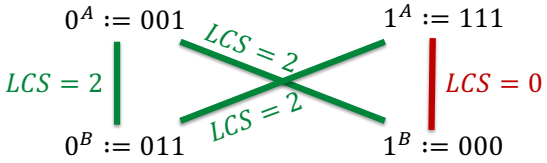
The reduction constructs a *vector gadget* for each vector from the input OV instance. Vector gadgets are composed of smaller *coordinate gadgets*, concatenated together with a suitable padding between them. The length of LCS of two vector gadgets is large when they orthogonal, and small otherwise. Finally, vector gadgets for vectors from each of the two input sets are combined into one long string using an *OR-gadget*.

An OV instance with two sets of n vectors of dimension d is reduced to an LCS instance with two strings of length $\mathcal{O}(d^2 \cdot n)$ over an alphabet of size 5. Hence, an $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for LCS would give an $\mathcal{O}(n^{2-\varepsilon}\text{poly}(d))$ time algorithm for OV, falsifying OV Hypothesis and SETH.

Proof: Coordinate Gadgets

OV: Given $A, B \subseteq \{0,1\}^d$ of size n each
Are there $a \in A, b \in B$ such that $\forall i$: $a_i \cdot b_i = 0$

we want to simulate the **coordinates** $\{0,1\}$ and the behavior of $a_i \cdot b_i$



replace a_i by a_i^A and b_i by b_i^B

$LCS(a_i^A, b_i^B)$ can be written as $f(a_i \cdot b_i)$, with $f(0) > f(1)$

Proof: Vector Gadgets

OV: Given $A, B \subseteq \{0,1\}^d$ of size n each
Are there $a \in A, b \in B$ such that $\forall i$: $a_i \cdot b_i = 0$

we want to simulate **orthogonality** of $a \in A, b \in B$

concatenate ${a_1}^A, \dots, {a_d}^A$, padded with a new symbol 2

length $4d$

$$VG(a) := {a_1}^A\ 2 \dots 2\ {a_2}^A\ 2 \dots 2\ {a_3}^A\ 2 \dots 2\ {a_4}^A$$
$$VG(b) := {b_1}^B\ 2 \dots 2\ {b_2}^B\ 2 \dots 2\ {b_3}^B\ 2 \dots 2\ {b_4}^B$$

- no LCS matches symbols in ${a_i}^A$ with symbols in ${b_j}^B$ where $i \neq j$
  lose one block of 2's
  cannot make up for it with symbols 0/1 since there are to few of them

$$VG(a) := {a_1}^A\ 2 \dots 2\ {a_2}^A\ 2 \dots 2\ {a_3}^A\ 2 \dots 2\ {a_4}^A$$
$$VG(b) := {b_1}^B\ 2 \dots 2\ {b_2}^B\ 2 \dots 2\ {b_3}^B\ 2 \dots 2\ {b_4}^B$$

- $LCS\big(VG(a), VG(b)\big) = (d-1)4d + \sum_{i=1}^{d} LCS({a_i}^A, {b_i}^B)$ $= f(a_i \cdot b_i)$

#2's

$LCS\big(VG(a), VG(b)\big) = C + 2$ if $a \perp b$
$LCS\big(VG(a), VG(b)\big) \leq C$ otherwise

for some constant C

Proof: Normalized Vectors Gadgets

OV: Given $A, B \subseteq \{0,1\}^d$ of size n each
Are there $a \in A, b \in B$ such that $\forall i\colon\ a_i \cdot b_i = 0$

new vector gadgets:

length C

$VG'(a)$: $VG(a)$ 3 … 3 $VG(a)$ 3 … 3

$VG'(b)$: 3 … 3 $VG(b)$ 3 … 3 $VG(b)$

$$LCS(VG'(a), VG'(b)) = \max\{LCS(VG(a), VG(b)), C\}$$

$$LCS(VG'(a), VG'(b)) = \begin{cases} C+2 & \text{if } a \perp b \\ C & \text{otherwise} \end{cases}$$

write VG for VG'

Proof: OR-Gadget

OV: Given $A, B \subseteq \{0,1\}^d$ of size n each
Are there $a \in A, b \in B$ such that $\forall i\colon\ a_i \cdot b_i = 0$

fresh symbol 4, want to construct: in the picture: $n = 3$

$VG(A[1])$ 4 … 4 $VG(A[2])$ 4 … 4 $VG(A[3])$ 4 … 4 $VG(A[1])$ 4 … 4 $VG(A[2])$ 4 … 4 $VG(A[3])$

4 … … … … … 4 $VG(B[1])$ 4 … 4 $VG(B[2])$ 4 … 4 $VG(B[3])$ 4 … … … … … 4

length $100d^2$ length $100d^2 \cdot 2n$

$VG(A[1])$ 4 … 4 $VG(A[2])$ 4 … 4 $VG(A[3])$ 4 … 4 $VG(A[1])$ 4 … 4 $VG(A[2])$ 4 … 4 $VG(A[3])$

4 … … … … … 4 $VG(B[1])$ 4 … 4 $VG(B[2])$ 4 … 4 $VG(B[3])$ 4 … … … … … 4

can align $VG(B[j])$ with $VG(A[\Delta + j \bmod n])$ for any offset Δ

$$LCS \geq \underbrace{(2n-1)100d^2}_{\#4\text{'s in upper string}} + \max_{\Delta} \sum_{j=1}^{n} LCS(VG(A[\Delta + j \bmod n]), VG(B[j]))$$

maximize over offset

need normalization!

If there is an orthogonal pair, some offset Δ aligns this pair, and we get

$$LCS \geq (2n-1)100d^2 + nC + 2$$

Claim: if no orthogonal pair exists: $LCS \leq (2n-1)100d^2 + nC$

$VG(A[1])$ 4 … 4 $VG(A[2])$ 4 … 4 $VG(A[3])$ 4 … 4 $VG(A[1])$ 4 … 4 $VG(A[2])$ 4 … 4 $VG(A[3])$

4 … … … … … 4 $VG(B[1])$ 4 … 4 $VG(B[2])$ 4 … 4 $VG(B[3])$ 4 … … … … … 4

non-orthogonal

$$LCS \leq (2n-1)100d^2 + \sum_{j=1}^{n} \begin{cases} 0 & \text{if } VG(B[j]) \text{ is not matched} \\ C & \text{if } VG(B[j]) \text{ is matched to one} \\ |VG(B[j])| - |4 \ldots 4| & \text{if } VG(B[j]) \text{ is matched to } > 1 \end{cases}$$

#4's in upper string

≤ 0

could match VG completely, but loose many 4's

3 Convolutions

Fix two binary operations $\oplus$ and $\otimes$. The $(\oplus, \otimes)$-convolution problem is, given two vectors A, B of length n, compute the vector C such that

$$C_k = \bigoplus_{i+j=k} A_i \otimes B_j = (A_0 \otimes B_k) \oplus (A_1 \otimes B_{k-1}) \oplus \cdots \oplus (A_k \otimes B_0).$$

Assuming the binary operations run in constant time, any convolution can be computed in $\mathcal{O}(n^2)$ time. Some convolutions can be computed much faster.

The most common convolution, $(+, \cdot)$-convolution is equivalent to polynomial multiplication (can you see why?) and is frequently used for multiplying large integer (can you see how?). It can be solved in $\mathcal{O}(n \log n)$ time using Fast Fourier Transform (FFT). FFT is beyond the scope of this course (but it is not very difficult, check it out if you are interested), but we will see another subquadratic algorithm for polynomial multiplication, due to Karatsuba.

Let $P(x)$ and $Q(x)$ be two degree n polynomials we want to multiply. Let us write them down as $P(x) = A(x)x^{n/2} + B(x)$ and $Q(x) = C(x)x^{n/2} + D(x)$, where A, B, C, D are of degree $n/2$. Note that

$$PQ = ACx^n + (AD + BC)x^{n/2} + BD,$$

and $AD + BC$ can be written as $(A + B)(C + D) - AC - BD$. Hence, it is sufficient to perform (recursively) three degree $n/2$ multiplications, namely AC, BD, $(A+B)(C+D)$, and several linear-time additions.

The running time of Karatsuba algorithm is described by the recursive equation

$$T(n) = 3T(n/2) + \mathcal{O}(n),$$

and by Master Theorem $T(n) = \mathcal{O}(n^{\log_2 3}) \leqslant \mathcal{O}(n^{1.585})$.

4 Pattern matching with mismatches

In the Pattern Matching With Mismatches problem (sometimes called Approximate Pattern Matching) we are given a text of length n, a pattern of length m, and an integer k, and we want to know if there is a continuous fragment of the text which equals to the pattern everywhere apart from at most k positions (in other words, their Hamming distance is at most k).

By an easy observation, any (n, m, k)-instance of the problem can be reduced to $\lceil n/m \rceil$ independent $(2m, m, k)$-instances. In the running time dependence on n was linear, this reduction does not change the asymptotic running time, and if the dependence was superlinear, the reduction improves the overall running time. Thus, we can restrict our attention to instances with $n = \Theta(m)$.

Exercise 3. Solve Pattern Matching With Mismatches in $\widetilde{\mathcal{O}}(n^{3/2})$ time[1]. Hint: compute for each *offset* i the number of positions at which the pattern matches the fragment of the

[1] $\widetilde{\mathcal{O}}(f(n)) = \mathcal{O}(f(n)\text{polylog}(n))$

text starting at position i. To this end, count naively characters that appear less than $\sqrt{n}$ times in both strings, and for each of the remaining characters run FFT on 0-1 vectors indicating where the character appears.

5 Matrix multiplication

Recall that the product of two $n \times n$ matrices A, B is the $n \times n$ matrix C such that

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}.$$

That can be naively computed in $\mathcal{O}(n^3)$. Volker Strassen was the first one to give a truly subcubic algorithm for the problem. His algorithm resembles Karatsuba algorithm for integer/polynomial multiplication, but the formulas are more obscure. It splits each of the two input matrices into four $n/2 \times n/2$ matrices, and performs seven (instead of naive eight) $n/2 \times n/2$ recursive matrix multiplications and a number of matrix additions. This gives the running time $\mathcal{O}(n^{\log_2 7}) \leqslant \mathcal{O}(n^{2.81})$.

After Strassen's breakthrough people came up with smarter and smarter ways to set up the recursion leading to better and better running times. See also `https://www.smbc-comics.com/comic/mathematicians` and Table 1 (borrowed from slides by Lech Duraj). The best possible matrix multiplication exponent is denoted by

$$\omega = \inf\{\delta \mid \text{matrix multiplication can be solved in time } \mathcal{O}(n^\delta)\}.$$

It is a big open problem whether $\omega = 2$.

Table 1: Matrix multiplication algorithms

$O(n^3)$		naïve
$O(n^{2.81})$	1969	Strassen
$O(n^{2.79})$	1979	Pan
$O(n^{2.78})$	1979	Bini et al.
$O(n^{2.55})$	1981	Schoenhage
$O(n^{2.53})$	1981	Pan
$O(n^{2.52})$	1982	Romani
$O(n^{2.50})$	1982	Coppersmith, Winograd
$O(n^{2.48})$	1986	Strassen
$O(n^{2.376})$	1987	Coppersmith, Winograd
$O(n^{2.373})$	2010	Stothers
$O(n^{2.3729})$	2012	Vassilevska Williams
$O(n^{2.3728639})$	2014	Le Gall
$O(n^{2.3728596})$	2021	Alman, Vassilevska Williams

All these truly subcubic algorithms are considered impractical, because constants hidden in the $\mathcal{O}$ notation make them faster than the naive cubic algorithm only for matrices

so large they cannot be conceivably stored in a computer, let alone multiplied.[2] Some recent experimental studies[3] challenge that folk wisdom, showing that Strassen algorithm can be beneficial already for reasonably sized (millions of entries) matrices. Nevertheless, asymptotically faster algorithms remain unpractical.

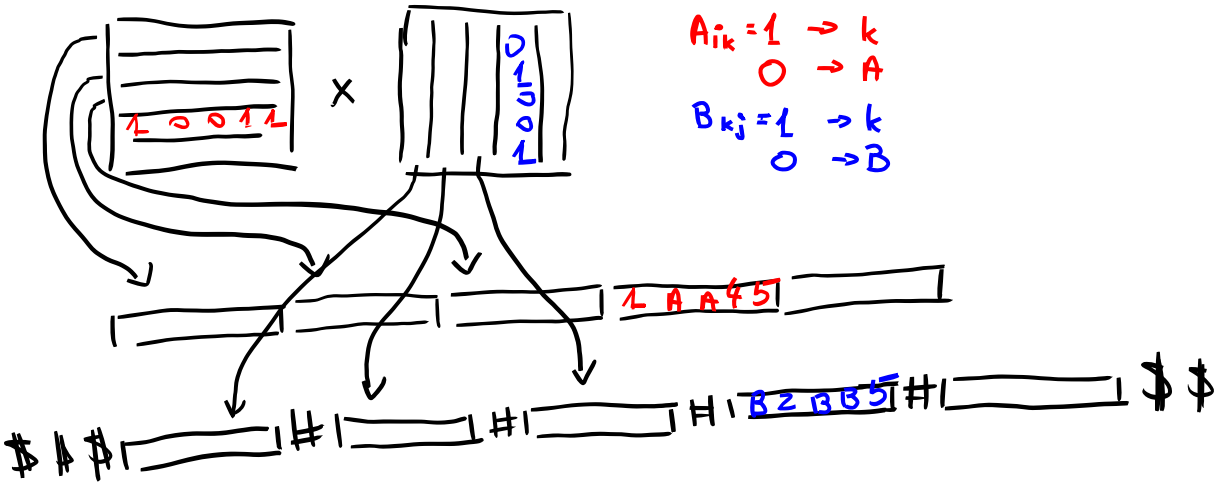
The fastest algorithm that does not use Strassen-like tricks, even for Boolean matrix multiplication, is due to Huacheng Yu and runs in $\mathcal{O}(n^3\text{polyloglog}n/\log^4 n)$ time.

This motivates the following "hypothesis" (in quotes because it is not well-defined).

Hypothesis 1 (BMM "Hypothesis")**.** *There is no* combinatorial[4] *algorithm running in* $\mathcal{O}(n^{3-\varepsilon})$ *time, for any* $\varepsilon > 0$*, that given two* 0-1 *matrices* A *and* B*, both of size* $n \times n$*, computes their Boolean product, that is the* $n \times n$ *matrix* C *such that*

$$C[i][j] = (A[i][1] \wedge B[1][j]) \vee (A[i][2] \wedge B[2][j]) \vee \cdots \vee (A[i][n] \wedge B[n][j]).$$

6 BMM-based lower bound for Pattern Matching



The reduction sketched above shows that computing the Hamming distances between a pattern and all fixed-length fragments of a text, both of size $\mathcal{O}(n)$, can compute the Boolean product of two $\sqrt{n} \times \sqrt{n}$ matrices. Hence, a combinatorial algorithm solving that problem (which is harder than Pattern Matching With Mismatches, but is something that the algorithm form Exercise 3 does anyway) in $\mathcal{O}(n^{3/2-\varepsilon})$ would falsify BMM Hypothesis. A conclusion is that if we want to improve over the $\widetilde{\mathcal{O}}(n^{3/2})$ running time, we should probably look for a way to use a fast matrix multiplication algorithm as a subprocedure.

[2]See this blog post on *galactic algorithms*: `https://rjlipton.wpcomstaging.com/2010/10/23/galactic-algorithms/`.

[3]E.g. `https://dl.acm.org/doi/10.5555/3014904.3014983`

[4]The term *combinatorial*, which is not well-defined, is supposed to rule out Strassen-like algorithms.