Polynomial lower bounds

Kent Quanrud

March 22, 2021

At the moment, we do not have any running time for SAT faster than $O\big(2^n \operatorname{poly}(m,n)\big)$. The *strong exponential time hypothesis* conjectures that it is impossible to solve SAT in $O\big(2^{(1-\epsilon)n} \operatorname{poly}(m)\big)$ time for any fixed $\epsilon > 0$. Of course we do not know if this conjecture is true. But giving it a name makes a landmark in the theoretical landscape around which other ideas can organize and new connections are established, as we will see.

1 Disjoint sets

Let $\mathcal{A} = \{x_1, \dots, x_n\} \subset \{0,1\}^m$ and $\mathcal{B} = \{y_1, \dots, y_n\} \subset \{0,1\}^m$ be two families of bit strings of length m. We can interpret each x_i and y_j as a subset of m, depending on which bits are set to 1. Alternatively we can thinking of them as vectors in $\mathbb{R}^m$. In this section we are interested in the following simple problem:

> Is there a disjoint pair of sets $x_i \in \mathcal{A}$ and $y_j \in \mathcal{B}$?

Equivalently, in terms of vectors:

> Is there an orthogonal pair of vectors $x_i \in \mathcal{A}$ and $y_j \in \mathcal{B}$?

We remind the reader that the two vectors x, y are **orthogonal** if their dot product, denoted $\langle x, y \rangle = \sum_{i=1}^m x_i y_i$, is 0. This problem is called the **orthogonal vectors problem** in the literature though the reader might find it easier to think in terms of disjoint sets.

The obvious algorithm is compare every pair x_i and y_j in $O(m)$ time per comparison, giving a total running time of $O(n^2 m)$. Can one do better? One might doubt that the problem has enough structure to take advantage of and do better. But we have already seen some algorithmic surprises under similar circumstances and it is hard to rule out the possibility of a clever idea. That said, we can give the following *conditional* lower bound, which states that a substantially better dependency on n would have dramatic implications for SAT as well.

Theorem 1. *There exists a universal constant $c > 0$ such that, for all $\epsilon > 0$, a $O\big(n^{2-\epsilon} \operatorname{poly}(m)\big)$ time algorithm for the orthogonal vectors problem implies a $O\big(2^{(1-c\epsilon)n} \operatorname{poly}(m)\big)$-time algorithm for SAT.*

Proof. Let f be a CNF with n variables and m clauses. We assume[1] that n is even and split the variables into two equal halves. For ease of notation, let us name the variables

$$f\big(x_1, \ldots, x_{n/2}, y_1, \ldots, y_{n/2}\big).$$

Alternatively we write $f(x, y)$ where $x \in \{0,1\}^{n/2}$ and $y \in \{0,1\}^{n/2}$.

Fix $x, y \in \{0,1\}^{n/2}$. Then the combined assignment (x, y) satisfies a clause C_i iff some literal among the x_j's, or some literal among the y_k's, makes it true. We say that x satisfies C_i in the former case, and that y satisfies C_i in the latter. (Of course, both can occur simultaneously.) For example, the clause

$$\big(x_j \vee \bar{y}_k\big)$$

is satisfied by any $x \in \{0,1\}^{n/2}$ such that $x_j = \texttt{true}$, or by any $y \in \{0,1\}^{n/2}$ such that $y_k = \texttt{false}$. We can now rephrase the satisfiability question as finding $x, y \in \{0,1\}^{n/2}$ such that every clause C_i is satisfied by either x or y.

We create two families of subsets of the clauses, $\mathcal{A}$ and $\mathcal{B}$, as follows. For each $x \in \{0,1\}^{n/2}$, we define a set

$$A_x = \big\{\text{clauses } C_i \text{ not satisfied by } x\big\} \subseteq \{C_1, \ldots, C_m\}.$$

Similarly, for each $y \in \{0,1\}^{n/2}$, we define a set

$$B_x = \big\{\text{clauses } C_i \text{ not satisfied by } y\big\} \subseteq \{C_1, \ldots, C_m\}.$$

We define

$$\mathcal{A} = \Big\{A_x : x \in \{0,1\}^{n/2}\Big\} \text{ and } \mathcal{B} = \Big\{B_y : y \in \{0,1\}^{n/2}\Big\}.$$

Two sets $A_x \in \mathcal{A}$ and $B_y \in \mathcal{B}$ are disjoint iff every clause is satisfied by either x or y; that is, iff $f(x, y) = \texttt{true}$.

We have $|\mathcal{A}| = |\mathcal{B}| = 2^{n/2}$. Thus a $O\big(n^{2-\epsilon}\,\mathrm{poly}(d)\big)$ algorithm for orthogonal vectors (with n sets in dimension d) implies a

$$O\left(\left(2^{n/2}\right)^{2-\epsilon}\mathrm{poly}(m)\right) = O\left(2^{((1-\epsilon/2)n)}\,\mathrm{poly}(m)\right)$$

time algorithm for SAT. ∎

2 Furthest pair and diameter

In the **furthest pair** problem, you are given a graph $G = (V, E)$ and two sets $S, T \subset V$. The goal is to find $s \in S$ and $t \in T$ maximizing the distance from s and t. One can consider the

[1] for simplicty; otherwise, add a dummy variable.

different combinations of unweighted and weighted graphs, and undirected and directed graphs.

For simplicity we assume that $|S| = |T| = k$ for some $k \in \mathbb{N}$, as it will be clear in hindsight how to generalize to uneven cardinalities. A simple algorithm single source shortest paths from every $s \in S$. This gives a $O\big(k(m + n \log n)\big)$ running time, where we use Dijkstra's shortest path algorithm. In unweighted graphs, this simplifies to $O(km)$.

Theorem 2. *Suppose there is an algorithm for (S, T)-furthest pair in unweighted, undirected graphs that runs in $O\big(k^{1-\epsilon}m\big)$ time, for $\epsilon > 0$. Then the orthogonal vectors problem, with two sets of n vectors in d dimensions, can be solved in $O\big(n^{2-\epsilon}d\big)$ time.*

Proof. Let $A, B \subset \{0,1\}^d$ be two sets of n bit strings of length d. The goal is to find $a \in A$ and $b \in B$ such that $\langle a, b\rangle = 0$. We will reduce the problem to (S, T)-furthest pair as follows. We make a layered graph with three layers.

1. The first layer has one vertex v_a for each vector $a \in A$.
2. The third layer has one vertex v_b for each vector $b \in B$.
3. In the middle layer has one vertex v_i for each index $i \in [d]$.

For each vector $a \in A$, and each index $i \in [d]$ such that $a_i = 1$, we add an edge from v_a to v_i. Similarly, for each vector $b \in B$, and each index $i \in [d]$ such that $a_i = 1$, we add an edge from v_i to v_b. For $a \in A$ and $b \in B$, if $\langle a, b\rangle \neq 0$, then there is a path of length 2 from v_a to v_b via any common index in their support. If $\langle a, b\rangle = 0$, then any path from v_a to v_b must have length at least 4. Thus, for $S = \{v_a : a \in A\}$ and $T = \{v_b : b \in B\}$, the farthest (S, T) pair gives an orthogonal pair of vectors, if one exists. ■

The **diameter** of a graph $G = (V, E)$ is the maximum distance $d(s, t)$ over all pairs $s, t \in V$. It is a special case of furthest pair where $S = T = V$. One can also show obtain lower bounds for computing the diameter of a graph, which we leave as an exercise.

3 Longest common subsequence

Recall the longest common substring problem. We are given two strings x and y from some finite alphabet. A **common substring** is a string z that appears (non-consecutively) as a substring of both x and y. We denote the length of the longest common subsequence of x and y by $\mathtt{LCS}(x, y)$. The longest common substring problem, which is very similar to the edit distance problem, can be solved with dynamic programming in $O(mn)$ time if m is the length of x and n is the length of y. Can one do better then the straight forward dynamic program?

For simplicity we will consider longest common substring for bit strings $x, y \in \{0,1\}^m$ of the same length m.

Theorem 3. *Suppose that the longest common subsequence between two strings of length n can be solved in $O\big(n^{2-\epsilon}\big)$ time for some fixed $\epsilon > 0$. Then the orthogonal vectors problem, with two sets of n vectors in m dimensions, can be solved in $O\big(\mathrm{poly}(m)n^{2-\epsilon}\big)$ time.*

We break down the proof into two main components. Recall that in the orthogonal vectors problem input we are given two sets of vectors $A = \{x_1, \ldots, x_n\}$ and $B = \{y_1, \ldots, y_n\}$. Our goal is to identify a single pair $x_i \in A$ and $y_j \in B$ such that $\langle x_i, y_j \rangle = 0$. The first step is to somehow simulate the inner product with an LCS computation. This is addressed by our first main lemma which describes a LCS-based gadget for verifying orthogonality.

Lemma 3.1. *Let $m \in \mathbb{N}$ be fixed. There are functions $h_1, h_2 : \{1,2\}^m \to \{1,2,3,4\}^M$ and an integer $T \in \mathbb{N}$ with the following properties.*

1. *Both $M, T \leq \text{poly}(m)$.*
2. *h_1 and h_2 both take $\text{poly}(m)$ time to evaluate.*
3. *For any two vectors $x, y \in \{0,1\}^m$,*

$$\texttt{LCS}(h_1(x), h_2(y)) = \begin{cases} T+1 & \text{if } \langle x, y \rangle = 0 \\ T & \text{if } \langle x, y \rangle > 0 \end{cases}$$

We prove this lemma below in Section 3.1.

While we can now implicitly identify an orthogonal pair of vectors via LCS, the real emphasis of orthogonal vectors is identifying a pair of vectors faster than an all-to-all comparison. The next lemma describes how to simulate an all-to-all-comparison with a single edit distance function.

Lemma 3.2. *Let $A_1, \ldots, A_n \in \{1,2,3,4\}^M$ and $B_1, \ldots, B_n \in \{1,2,3,4\}^M$ be two sets of n bit strings of length M, and T an integer, such that for all A_i and all B_j,*

$$T \leq \texttt{LCS}(A_i, B_j) \leq T + 1.$$

Then in $O(Mn)$ time, one can construct strings $\bar{A}, \bar{B} \in \{1,2,3,4,5\}^{M'}$ of length $M' = O(Mn)$, and select an integer U, such that

$$\texttt{LCS}(\bar{A}, \bar{B}) > U \text{ iff } \texttt{LCS}(A_i, B_j) = T + 1 \text{ for some } A_i \text{ and } B_j.$$

We prove this lemma below in Section 3.2.

If we assume the two lemmas above to be true, then we have the ingredient to prove Theorem 3. Let us do so now, and address each lemma separately afterwards.

Proof of Theorem 3. Given two sets A and B of n bit strings in m dimensions, by combining the above two lemmas, two strings $\bar{A}$ and $\bar{B}$ of length $O(n\,\text{poly}(m))$ such that $\texttt{LCS}(\bar{A}, \bar{B})$ indicates whether there is an orthogonal pair. A subquadratic algorithm for LCS, applied to $\bar{A}$ and $\bar{B}$, would imply a subquadratic (in n) algorithm for orthogonal vectors. ■

3.1 The LCS Orthogonality Gadget

We build up the gadget described in Lemma 3.1 in three stages.

1. Simulate the product xy for single bits $x, y \in \{0,1\}$ via LCS.
2. Simulate the dot product $\langle x, y \rangle$ for single bits $x, y \in \{0,1\}$ via LCS.
3. Normalize step 2 so that it assigns the same value to all non-orthogonal pairs.

3.1.1 Encoding bits

Lemma 3.3. *There are function $f_1, f_2 : \{0,1\} \to \{1,2\}^2$ such that for any two bits $x_1, x_2 \in \{0,1\}$,*

$$\mathtt{LCS}(f_1(x_1), f_2(x_2)) = 1 - x_1 x_2.$$

Proof. Consider the following table computing the LCS between particular two bit strings.

LCS	12	22
21	1	1
11	1	0

Consequently we define

$$f_1(0) = \mathtt{01}, \qquad f_1(1) = \mathtt{11} \qquad f_2(0) = \mathtt{10} \qquad f_2(1) = \mathtt{00},$$

which satisfies the claim. ∎

3.1.2 Bit strings

Lemma 3.4. *There are functions $g_1, g_2 : \{1,2\}^m \to \{1,2,3\}^M$, for $M = O(m^2)$, with the following properties.*

1. *g_1 and g_2 both take $O(M)$ to compute.*
2. *For all $x_1, x_2 \in \{0,1\}^m$,*

$$\mathtt{LCS}(g_1(x_1), g_2(x_2)) = 2m^2 - \langle x_1, x_2 \rangle.$$

Proof. By Lemma 3.3, we already have a good solution for the simplest case of a single bit, $m = 1$. The claim is that we can obtain something similar for longer strings of length m.

Let $m' \in \mathbb{N}$ be a parameter TBD. Let $C = \mathtt{3}^{m'}$; that is, C is the string

$$C = \mathtt{3}^{m'} = \mathtt{333\cdots3}$$

where 3 is repeated m' times. We use C as a sort of divider between coordinates. We define g_1 and g_2 by

$$g_1(x_1) = C \cdot f_1(x_1) \cdot C \cdot f_1(x_2) \cdot C \cdots C \cdot f_1(x_1) \cdot C$$
$$g_2(x_2) = C \cdot f_2(y_1) \cdot C \cdot f_2(y_2) \cdot C \cdots C \cdot f_2(y_m) \cdot C,$$

where $\cdot$ denote concatenation. These strings have length

$$(m+1)|C| + 2m = (m+1)m' + 2m.$$

Claim. *The length of the longest common subsequence is at least*

$$\texttt{LCS}(y_1, y_2) \geq (m+1)m' + \sum_{i=1^m} \texttt{LCS}(f_1(x_i), f_2(y_i)).$$

Indeed, the claimed length is obtained by matching up all the C's, and taking the longest common subsequence of $f_1(x_i)$ and $f_2(y_i)$ for each index i. □

Claim. *Any common subsequence between $g_1(x_1)$ and $g_2(x_2)$ matches up a character from $f_1(x_i)$ with a character from $f_2(y_j)$, for different indices $i \neq j$, has length at most $(m'+2)m$.*

Indeed, suppose an edit sequence matched some characters between A_i and B_j for $i \neq j$. Suppose $i < j$. Then there is at least one more substring C to the right of A_i then B_j, which implies that at least m' of the 2's on the right of A_i will not be matched. Similarly if $i > j$, then at least m' of the 2's on the left of A_i will not be matched. Otherwise each of the A_i's can contributed at most 2. □

Let $m' = 2m$. Then we have $(m+1)m' \geq (m'+2)m$. $(m+1)m'$ is attainable simply by matching up all the C's. Thus the longest common subsequence can always be attained without matching up a character from $f_1(x_i)$ to $f_2(y_j)$ for $i \neq j$. This forces the solution to be bounded above by

$$\texttt{LCS}(g_1(x), g_2(y)) \leq (m+1)m' + \sum_{i=1}^{m} \texttt{LCS}(f_1(x_i), f_2(y_i)).$$

Combined with the lower bound in the first claim, we obtain the equality

$$\texttt{LCS}(g_1(x), g_2(y)) = (m+1)m' + \sum_{i=1}^{m} \texttt{LCS}(f_1(x_i), f_2(y_i)).$$

Moreover, we have $\texttt{LCS}(f_1(x_i), f_2(y_i)) = 1 - x_i y_i$ for all i by Lemma 3.3. Substituting into the above completes the proof of the lemma. ■

3.1.3 Normalization

Lemma 3.1. *Let $m \in \mathbb{N}$ be fixed. There are functions $h_1, h_2 : \{1,2\}^m \to \{1,2,3,4\}^M$ and an integer $T \in \mathbb{N}$ with the following properties.*

1. *Both $M, T \leq \text{poly}(m)$.*
2. *h_1 and h_2 both take $\text{poly}(m)$ time to evaluate.*
3. *For any two vectors $x, y \in \{0,1\}^m$,*

$$\texttt{LCS}(h_1(x), h_2(y)) = \begin{cases} T+1 & \text{if } \langle x, y \rangle = 0 \\ T & \text{if } \langle x, y \rangle > 0 \end{cases}$$

Proof. Let $g_1, g_2 : \{1,2\}^m \to \{1,2,3\}^M$ be given by Lemma 3.4, where M is a polynomial in m. Recall that we have

$$\texttt{LCS}(g_1(x), g_2(y)) = 2m^2 - \langle x, y \rangle$$

for any two $x, y \in \{0,1\}^m$.

Let $T = 2m^2 - 1$. Consider the function $h_1, h_2 : \{0,1\}^m \to \{0,1\}^{M+T}$ defined by

$$h_1(x) = 4^T g_1(x)$$
$$h_2(x) = g_2(x)4^T,$$

where 4^T denotes the string of T consecutive 4's. Observe that the LCS of $h_1(x)$ and $h_2(y)$ can only have matches between the 4's, or between $g_1(x)$ and $g_2(x)$, but not both. As such, we have

$$\texttt{LCS}(h_1(x), h_2(y)) = \max\{T, \texttt{LCS}(g_1(x), g_2(y))\} = \begin{cases} 2m^2 = T+1 & \text{if } \langle x, y \rangle = 0 \\ T & \text{otherwise,} \end{cases}$$

as desired. ∎

3.2 The LCS ∨-gadget

Our goal in this section is to prove the second main lemma, which we first restate for convenience.

Lemma 3.2. *Let $A_1, \ldots, A_n \in \{1,2,3,4\}^M$ and $B_1, \ldots, B_n \in \{1,2,3,4\}^M$ be two sets of n bit strings of length M, and T an integer, such that for all A_i and all B_j,*

$$T \leq \texttt{LCS}\big(A_i, B_j\big) \leq T + 1.$$

Then in $O(Mn)$ time, one can construct strings $\bar{A}, \bar{B} \in \{1,2,3,4,5\}^{M'}$ of length $M' = O(Mn)$, and select an integer U, such that

$$\texttt{LCS}\big(\bar{A}, \bar{B}\big) > U \text{ iff } \texttt{LCS}\big(A_i, B_j\big) = T + 1 \text{ for some } A_i \text{ and } B_j.$$

Given A_i's and B_j's as described above, the goal, motivated by the reduction from orthogonal vectors, is to decide if there is a single pair of indices i, j such that $\texttt{LCS}\big(A_i, B_j\big) = T + 1$. We are particularly interested in the regime where the parameters M and T are lower-order terms and the dominant term is n. In particular, in the reduction from orthogonal vectors, a $O\big(n^{1.99}\operatorname{poly}(M,T)\big)$-time running time identify such a pair A_i and B_j gives a $O\big(n^{1.999}\operatorname{poly}(M,T)\big)$ time algorithm for SAT.

We will combine the A_i's into (some carefully constructed) string $\bar{A}$, and the B_j's into a string $\bar{B}$, each of same size $O\big(\operatorname{poly}(M,T)n\big)$. We will design $\bar{A}$ and $\bar{B}$ so that $\texttt{LCS}\big(\bar{A}, \bar{B}\big)$ (somehow) reveals whether or not we have $\texttt{LCS}\big(A_i, B_j\big) = T + 1$ for any pair of indices i and j.

Recall that the A_i's and B_j's consist of characters from the set $\{1, 2, 3\}$. Let $D = 5^S$ be the string of S consecutive 5's, for a sufficiently large parameter $S \in \mathbb{N}$ TBD. Consider the concatenated strings

$$\bar{A} = D^n A_1 D A_2 D \cdots D A_n D^n$$
$$\bar{B} = B_1 D B_2 D \cdots D B_n D B_1 D B_2 D \cdots D B_n$$

To gain some informal intuition for the above constructions, observe that $\bar{A}$ has $3n-1$ D-blocks and $\bar{B}$ has $2n-1$ D-blocks. For S, the $\texttt{LCS}(\bar{A}, \bar{B})$ will be incentivized to match up the maximum number of D-blocks, $2n-1$. Any maximum matching between the $2n-1$ D-blocks of $\bar{B}$ to the $3n-1$ D-blocks to $\bar{A}$, means that each A_i-block above can be partially matched to at most one B_j-block below. The net effect is that the D's encourages the lengths of the LCS to have the form $(2n-1)\,\texttt{LCS}(D, D) + \sum_{i=1}^{n} \texttt{LCS}(A_i, B_{\pi(i)})$ for some mapping $\pi : [n] \to [n]$ of A_i's to B_j. The sum is $> nT$ iff $\texttt{LCS}(A_i, B_{\pi(i)})$ for one of these matches i.

The formal argument consists of two parts. Let

$$U = (2n-1)S + nT$$

Observe that $\texttt{LCS}(\bar{A}, \bar{B}) \geq U$ by (say) matching up the A_i-blocks with the first n B-blocks (each contributing at least T), and then matching up all $(2n-1)$ D-blocks in between and afterwards (each contributing S). This lower bound of U is the threshold that will decide whether or not some $\texttt{LCS}(A_i, B_j) > T$ for some pair of A_i and B_j.

Lemma 3.5. *Suppose* $\texttt{LCS}(A_i, B_j) = T + 1$ *for some* i, j. *Then* $\texttt{LCS}(\bar{A}, \bar{B}) \geq U + 1$.

Proof. Suppose $i \leq j$. Consider the common subsequence where we matching up A_i to the first B_j, and then match up A_{i-k} with B_{j-k} to the left and then match up A_{i+k} with B_{j+k} to the right for each relevant value of k. In between we match up the D's. In the example below, we have $n = 5$. The strings are lined up to match up A_3 with B_4, and unmatched blocks are crossed off.

$$\cancel{D}\cancel{D}\cancel{D}\cancel{D}\cancel{D}DA_1DA_2DA_3DA_4DA_5DDDD\cancel{D}$$
$$\cancel{B_1}DB_2DB_3DB_4DB_5DB_1D\cancel{B_2}D\cancel{B_3}D\cancel{B_4}D\cancel{B_5}$$

The total cost of the matching is at least $(n-1)T + T + 1 + (2n-1)S$, since $\texttt{LCS}(A_i, B_j) = T + 1$, while the other A-blocks and B-blocks match up to contribute at least T to the LCS.

If $i > j$ then we match up A_i to the *second* B_j instead, and otherwise proceed similarly, arriving at the same conclusion as before. ■

Lemma 3.6. *Let* $S \geq M - T$. *Suppose* $\texttt{LCS}(A_i, B_j) \leq T$ *for all* i, j. *Then* $\texttt{LCS}(\bar{A}, \bar{B}) \leq U$.

Proof. We start with the assumption that all the D-blocks of B-blocks are matched, giving us an initial score of $(2n-1)S$. We then analyze each A_i's in order and consider the net contribution from A_i to the LCS. If in the analysis of a block A_i we realize that we did not match part of a D-block in $\bar{B}$ then we charge the unmatched letters from the D-block against the contribution of A_i to the LCS. We claim that each A_i has a net contribution of at most T. For $i = 1, \ldots, n$, consider A_i.

1. If A_i is partially matched to characters of (at most) one B_j, then the maximum contribution from A_i is at most $\texttt{LCS}(A_i, B_j)T$.

2. If A_i is partially matched to characters of at least two disjinct B_j's, then at least one D-block between the B_j's is forfeited. This results in an extra loss of S_i because we had initially assumed all of the D-blocks in $\bar{B}$ were matched. Thus the net contribution of A_i is at most $|A_i| - S = M - S \leq T$.

In either case, the net contribution each A_i is T. Overall we have

$$\texttt{LCS}(\bar{A}, \bar{B}) = (2n-1)S + \sum_{i=1}^{n} (\text{net contrib. of } A_i) \leq (2n-1) + nT,$$

as desired. ■

4 Additional notes references

The strong exponential time hypothesis was formulated by Calabro, Impagliazzo, and Paturi [4] and builds on the exponential time hypothesis formulated in [5]. For more results of this nature, see (e.g.) the recent classes by Bringmann and Künnemann [3] and Williams and Williams [8]. This chapter is based in part on lecture notes from these classes. The connection between k-SAT and orthogonal vectors is from [7]. The lower bounds for furthest path and diameter in Section 2 is by Roditty and Williams [6]. The description here is based on a recent talk by Nicole Wein at the Michigan-Purdue Theory Seminar in January, 2021. The presentation of the lower bound for LCS is specifically based on [3], and the result is from [1, 2].

5 Exercises

Exercise 1. Assuming the strongly exponential time hypothesis, give a lower bound on the running time for any exact algorithm for subset sum.

Exercise 2. 1. Prove a lower bound for computing the diameter of an unweighted and undirected graph, assuming the strong exponential time hypothesis.

2. Here and in the discussion we gave lower bounds for exact algorithms, but one can consider *approximation* algorithms as well. For $\alpha \geq 1$, let us say that an algorithm gives an α-approximation of the diameter D if it returns a value D' guaranteed to satisfy the inequality $D/\alpha \leq D' \leq D$. For example, a 2-approximation returns a value D' that is at least half the real diameter, and $\leq$ the real diameter.

   For some constant $\alpha > 1$ of your choosing[2], give a lower bound for the running time of any α-approximation for computing the diameter of an unweighted and undirected graph, assuming the strong exponential time hypothesis.

Exercise 3. k orthogonal vectors...

[2]The larger the better – I think that 3/2 is doable. Anything greater than 1 is already interesting.

References

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. "Tight Hardness Results for LCS and Other Sequence Similarity Measures". In: *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. Ed. by Venkatesan Guruswami. IEEE Computer Society, 2015, pp. 59–78.

[2] Karl Bringmann and Marvin Künnemann. "Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping". In: *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. Ed. by Venkatesan Guruswami. IEEE Computer Society, 2015, pp. 79–97. URL: https://arxiv.org/abs/1502.01063.

[3] Karl Bringmann and Marvin Künnemann. *Fine-Grained Complexity Theory*. Lecture notes for a course at the Max-Planck-Institut für Informatik. 2019. URL: https://www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/summer19/fine-complexity/.

[4] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. "The Complexity of Satisfiability of Small Depth Circuits". In: *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*. Ed. by Jianer Chen and Fedor V. Fomin. Vol. 5917. Lecture Notes in Computer Science. Springer, 2009, pp. 75–85. URL: https://cseweb.ucsd.edu/~paturi/myPapers/pubs/CalabroImpagliazzoPaturi_2009_iwpec.pdf.

[5] Russell Impagliazzo and Ramamohan Paturi. "Complexity of k-SAT". In: *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*. IEEE Computer Society, 1999, pp. 237–240. URL: http://cseweb.ucsd.edu/~paturi/myPapers/pubs/ImpagliazzoPaturi_2001_jcss.pdf.

[6] Liam Roditty and Virginia Vassilevska Williams. "Fast approximation algorithms for the diameter and radius of sparse graphs". In: *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM, 2013, pp. 515–524.

[7] Ryan Williams. "A new algorithm for optimal 2-constraint satisfaction and its implications". In: *Theor. Comput. Sci.* 348.2-3 (2005), pp. 357–365. URL: https://people.csail.mit.edu/rrw/2-csp-final.pdf. Preliminary version in ICALP, 2004.

[8] Virginia Vassilevska Williams and Ryan Williams. *Fine-Grained Algorithms and Complexity*. Course 6.S078 at MIT. 2020. URL: https://people.csail.mit.edu/virgi/6.s078/.