Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)

Seminar on Advanced Algorithms and Data Structures

Simon Weber – 15-920-655

7th of April 2018

1 Introduction

In this presentation we will explore a proof from [BI15] showing that the Levenshtein distance between two strings cannot be computed in strongly subquadratic time given that the Strong Exponential Time Hypothesis [IP01] is true.

The *Levenshtein (or edit) distance* counts the minimum number of operations needed to transform one input string into the other. The allowed operations are deletion and insertion of a character, or substitution of one character for another. From now on we will denote the Levenshtein distance between two strings a and b as $EDIT(a, b)$.

As an example, the words "cat" and "mane" have an edit distance of three – the c is changed into an m, the t is changed into an n and an e is added at the end.

There exist many algorithms for solving $EDIT$ in quadratic time in the length of the input strings, for example the dynamic programming algorithm that one usually encounters in basic Algorithmics classes. There are more efficient algorithms that achieve a runtime of $O(n^2/log^2 n)$, but so far nobody has discovered a "strongly subquadratic" algorithm – that is an algorithm in $O(n^{2-\epsilon})$, with $\epsilon > 0$. We will show that this is indeed a lower bound, under the condition that the Strong Exponential Time Hypothesis is true.

In general, it is very difficult to prove lower bounds for computational problems. There are two main ways to prove such lower bounds:

- Restricting the computational model: For example, sorting can be shown to have a lower bound of $\Omega(nlogn)$, when only considering comparison-based algorithms. In practice, other algorithms like *counting sort* outperform this bound when the items to be sorted are integers.

- Reducing onto another problem: By showing that a problem is at least as hard as another problem, we can "inherit" the lower bounds proven or conjectured for that other problem. Even if no definitive proof for a lower bound is found, knowing that a large set of problems reduce to each other can reinforce the trust in that we have indeed reached the theoretical limit.

In this case, we use the *Strong Exponential Time Hypothesis (SETH)* as a root hypothesis for our lower bound. The *SETH* states that *SAT*, the satisfiability problem of Conjunctive Normal Form formulae with no limit on the amount of literals in a clause, cannot be solved in strongly subexponential time, that is in $O(2^{n(1-\epsilon)})$, $\epsilon > 0$, where n is the number of variables. It remains unknown whether *SETH* is true. *SETH* is a much stronger statement than $P \neq NP$, so it is unlikely to be proven correct in the near future, but it could be disproven simply by finding a strongly subexponential algorithm for *SAT*. We will explore whether it is reasonable to assume *SETH* later in 5.1.

2 Proof Outline

We will show that the existence of an $O(n^{2-\epsilon})$ $(\epsilon > 0)$ algorithm for *EDIT* implies the existence of an $O(poly(m)2^{(1-\delta)n})$ $(\delta > 0)$ algorithm for *SAT*, where m is the number of CNF clauses. Assuming $m = \omega(logn)$, this is in $O(2^{(1-\delta)n})$ and this contradicts *SETH*. Thus assuming *SETH*, there exists no strongly subquadratic algorithm for *EDIT*.

The reduction will be done in two main steps:

1. Reduce *SAT* to the *Orthogonal Vectors Problem*. Something very similar has been proven in a previous paper ([Wil05]) and we will show the adapted proof for *OVP*.

2. Reduce the *Orthogonal Vectors Problem* to *EDIT*.

3 The Orthogonal Vector Problem

The *Orthogonal Vectors Problem* is the problem of deciding for two sets A and B ($|A| = |B| =: n$) of binary vectors of dimension d whether there are orthogonal vectors $a \in A$ and $b \in B$, in other words $a \cdot b = 0$.

OVP can be solved easily in $O(n^2 d)$, for example by naively computing $a \cdot b$ for each pair of vectors. Similar to *EDIT*, given *SETH*, there is no strongly subquadratic (in terms of n) algorithm (given $d = \omega(logn)$). This has been called the *Orthogonal Vectors Conjecture* [Wil05]. Not only is it implied by *SETH*, it is also considered more likely to be true [Wil18].

In [Wil05] a very similar problem has been proven not to have strongly subquadratic algorithms, given *SETH* – the problem of deciding whether a vector from A is a subset of a vector from B. The proof can be easily adapted to work for the *OVP*:

Given a CNF formula, the variables are split into two equally sized sets and for both sets, all possible assignments are generated. For each of those partial assignments we compute the following vector v:

$$v_i = \begin{cases} 0 & \text{if the assignment satisfies the i-th clause} \\ 1 & \text{otherwise} \end{cases}$$

For the formula to be true under a pair of two partial assignments, at least one of the partial assignments needs to satisfy each clause. Therefore the CNF formula is satisfied if and only if there are two orthogonal vectors, because for every clause (coordinate) there is at least one partial assignment (vector) satisfying the clause (the vector being 0 at this coordinate).

The *OVP* problem instance has size $O(2^{n/2})$ for n *SAT* variables. The number of clauses m directly translates into the dimension d of the vectors. Therefore a strongly subquadratic algorithm for *OVP* would induce a strongly subexponential algorithm for *SAT*, as $(2^{n/2})^{2-\epsilon} = 2^{n(1-\epsilon/2)}$.

4 Reducing EDIT

For this second part of the proof, we will build a string S_1 out of the vector set A and a string S_2 out of B, such that the edit distance takes on a fixed value if there are no two orthogonal vectors and a lower value if there are.

Let us first define our goal in detail: We want to generate two strings S_1 and S_2, such that we can decide from the value of $EDIT(S_1, S_2)$ whether there are orthogonal vectors. The length of the strings has to be linear in the number of vectors n in A and B and at most polynomial in their dimension d. For a more streamlined proof we allow us to use an alphabet of size 4 for the strings, $\Sigma = \{0, 1, 2, 3\}$. In [BK15], a proof for a binary alphabet is shown, but it complicates the process too much for only a small improvement in the final conclusion.

Luckily the proof is possible with an incremental construction. We will encode every coordinate of the vectors into a so called *coordinate gadget* string. Using the coordinate gadgets, we will build an encoding for the full vectors (*vector gadgets*). The vector gadgets are then used to build the final strings for our instance of *EDIT*.

We will go through the proof backwards, such that it is more clear why each step is constructed the way it is.

$\circ$ will denote the concatenation of two strings, $\bigcirc$ the concatenation of a set of strings.

4.1 From vector gadgets to the final strings

We assume that we have vector encodings with the following properties:

- A vector $a \in A$ is translated to the vector gadget $VG_1(a)$, a vector $b \in B$ is translated to $VG_2(b)$.
- $EDIT(VG_1(a), VG_2(b)) \leq E_s$ if a and b are orthogonal, and $= E_u$ otherwise. $E_u > E_s$ and both are constants (only depending on the dimension of the vectors).
- Vector Gadgets only contain 0s and 1s.

Now we concatenate these vector gadgets in such a way that if there are no orthogonal vectors, the edit distance will be a predicted value, and any lower value otherwise.

For our first string (from vector set A) we concatenate all vector gadgets, separated with long runs of 2s. Additionally, we encapsulate the whole string in huge runs of 3s.

$$S_1 = 3^{|S_2|} \bigcirc_{a \in A} (2^T VG_1(a) 2^T) 3^{|S_2|}$$

where T is just a "large enough" number (polynomial in d), that prevents interference between neighboring vector gadgets.

For our second string (from vector set B) we also concatenate all vector gadgets, separating them the same way. We now insert additional vector gadgets on both beginning and end. These additional vector gadgets are all identical and are the gadgets of the full-one-vector f.

$$S_2 = \left(\bigcirc_{i=1}^{|A|-1} 2^T VG_2(f) 2^T\right) \left(\bigcirc_{b \in B} 2^T VG_2(b) 2^T\right) \left(\bigcirc_{i=1}^{|A|-1} 2^T VG_2(f) 2^T\right)$$

As the full-one-vector f obviously can't be orthogonal to any other vector, adding it's gadget into S_2 gives us the ability to shift the two sets relative to each other, without introducing any new orthogonal vector pair. Thanks to this shifting we can make the gadgets of orthogonal vectors *align*.

Notice that the lengths of S_1 and S_2 are in $O(n \cdot poly(d) \cdot max(|VG_1|, |VG_2|)$, which is what we required.

If there are no two orthogonal vectors, going from S_1 to S_2, one obviously has to remove/transform the huge runs of 3s as there are no 3s in S_2. Additionally, the $|A|$ vector gadgets have to be transformed as well, each one giving E_u edit operations. This gives a fixed cost of

$$EDIT(S_1, S_2) = 2 \cdot |S_2| + |A| \cdot E_u$$

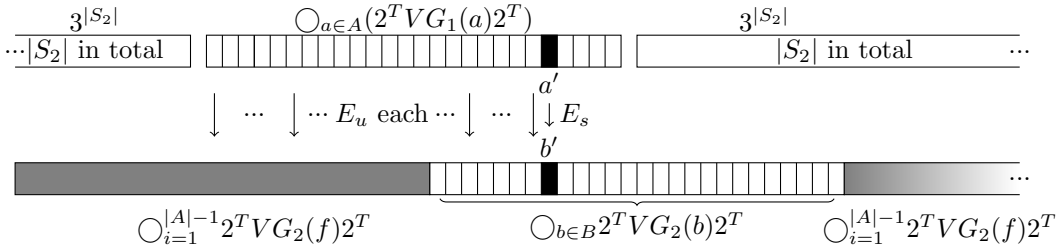
We assumed that it is optimal to transform each vector gadget independently. This might be intuitive (especially when considering the structure of the vector and coordinate gadgets in the next chapters) but it isn't trivial to

prove. The full proof involves an extensive case distinction on the location of similar substrings in the two strings, but we omit it here.

On the other hand, if there are orthogonal vectors, say $a' \cdot b' = 0$, showing that the edit distance is lower than a certain value can be done simply by showing the editing process: The sequence of vector gadgets in S_1 is transformed into a substring of S_2 by transforming each vector gadget independently. The vector gadgets are aligned in such a way that $VG_1(a')$ is transformed into $VG_2(b')$. Thanks to the repetitions of $VG_2(f)$, this is always possible and takes at most $(|A|-1) \cdot E_u + E_s$ steps. Now we remove or substitute the 3s, to get to S_2. Note that we will never need to add any symbols, as $|S_2|$ 3s on both sides are always enough to build S_2 from. This gives us a cost of

$$EDIT(S_1, S_2) \leq 2 \cdot |S_2| + |A| \cdot E_u + (E_s - E_u)$$

Figure 1: The editing process if there are orthogonal vectors a' and b'



4.2 From coordinate gadgets to vector gadgets

In this part we will construct vector gadgets out of coordinate gadgets with the following properties:

- A coordinate a_i from a vector $a \in A$ is translated to the coordinate gadget $CG_1(a_i)$ and a coordinate b_i from a vector $b \in B$ is translated to $CG_2(b_i)$.
- The edit distance between two coordinate gadgets is $3 \cdot l_0$ if they are both 1 and l_0 otherwise.
- There is an additional gadget g that has edit distance of $l_0 + 1$ to any coordinate gadget from B.
- Coordinate gadgets only contain 0s and 1s.

We obviously want our vector gadgets to have the properties outlined in the previous chapter.

It would be easy to construct vector gadgets for which the edit distance scales linearly with the product $a \cdot b$, for example by concatenating all coordinate gadgets. This would generate problems when looking at more than one vector, as a "strongly non-orthogonal" vector pair would increase the edit distance by a lot, and thus masking the lower edit distance coming from an orthogonal vector pair.

Luckily, the additional gadget g comes in handy. As it is always more expensive to transform a coordinate gadget into g than to transform an orthogonal coordinate gadget pair, but cheaper than non-orthogonal coordinate gadget pairs, it can be used to provide a constant-sized alternative editing path. The edit distance always searches for the minimum possible, therefore it will choose the cheaper orthogonal way if it exists, but also not grow without limit with regards to $a \cdot b$ if the vectors aren't orthogonal.

The vector gadgets we use are

$$VG_1(a) = Z_1 L V_0 R Z_2$$

and

$$VG_2(b) = V_1 D V_2$$

where $Z_1 = Z_2 = 0^{l_2}$ and $V_0 = V_1 = V_2 = 1^{l_2}$. R is the concatenation of all coordinate gadgets of a and D is the concatenation of all coordinate gadgets of b. L is the sequence of d times the extra gadget g. l_2 is once again just a "large enough" constant, defined as $(1000d)^3$ and designed to prevent interference between the different sections of the vector gadget.

The lengths of the vector gadgets VG_1 and VG_2 are in $O(poly(d) \cdot max(|CG|, g))$.

If a and b are orthogonal, we can show the upper bound

$$EDIT(VG_1(a), VG_2(b)) \leq E_s$$

by showing how to transform one into the other: Z_1 and L are deleted from $VG_1(a)$ ($l_2 + |L|$ operations), then R is transformed into D by transforming each coordinate independently ($d \cdot l_0$ operations). To finish, Z_2 is transformed into V_2 (l_2 operations). This costs $E_s := 2l_2 + |L| + dl_0$ in total.

If a and b are not orthogonal, the edit distance is exactly

$$EDIT(VG_1(a), VG_2(b)) = E_u := E_s + d.$$

We will first show that $EDIT(VG_1(a), VG_2(b)) \leq E_u$ using a concrete transformation: We delete R and Z_2 from $VG_1(a)$ ($l_2 + |R| = l_2 + |L|$ operations). Then we transform L into D, which takes at most $(l_0 + 1) \cdot d$ operations. Lastly, Z_1 is transformed into V_1 (l_2 operations). In total, this amounts to $E_u = 2l_2 + |L| + dl_0 + d = E_s + d$ operations. The proof that this is also a lower bound is once again very tedious and we will omit it here.

Figure 2: The editing process if the vectors are orthogonal

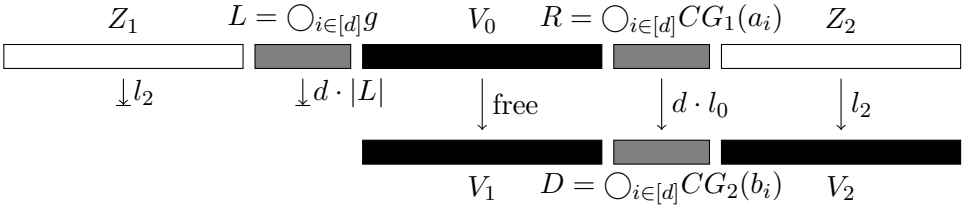
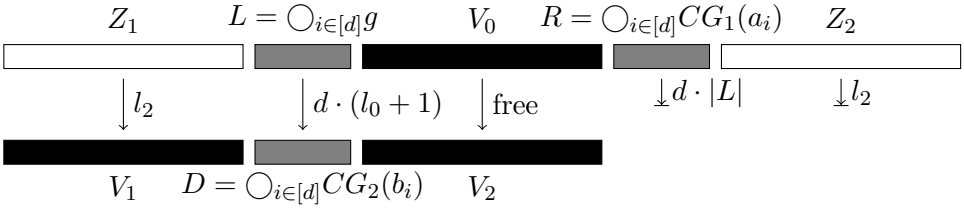


Figure 3: The editing process if the vectors are **not** orthogonal



4.3 Constructing our coordinate gadgets

Now it only remains to construct the coordinate gadgets used in the previous section. This isn't too complicated:

$$CG_1(x) := \begin{cases} 0^{l_1}0^{l_0}1^{l_0}1^{l_0}1^{l_0}0^{l_1} & \text{if } x = 0 \\ 0^{l_1}0^{l_0}0^{l_0}0^{l_0}1^{l_0}0^{l_1} & \text{if } x = 1 \end{cases}$$

$$CG_2(x) := \begin{cases} 0^{l_1}0^{l_0}0^{l_0}1^{l_0}1^{l_0}0^{l_1} & \text{if } x = 0 \\ 0^{l_1}1^{l_0}1^{l_0}1^{l_0}1^{l_0}0^{l_1} & \text{if } x = 1 \end{cases}$$

$$g = 0^{\frac{l_1}{2}-1}10^{\frac{l_1}{2}}0^{l_0}1^{l_0}1^{l_0}1^{l_0}0^{l_1}$$

where $l_0 = 1000 \cdot d$ and $l_1 = (1000 \cdot d)^2$. When writing these gadgets aligned above each other, it should be obvious that they have the desired edit distances to each other.

These coordinate gadgets and g once again have lengths in $O(poly(d))$. Piecing it all together, we thus know that our final strings have the desired lengths in $O(n \cdot poly(d) \cdot O(poly(d) \cdot O(poly(d))))$, which is $O(n \cdot poly(d))$.

5 Conclusion

We have shown that from an instance of the *Orthogonal Vectors Problem*, we can construct an instance of *EDIT*, such that the strings have lengths linear in the number of vectors and polynomial in the vector dimension. By looking at the result of *EDIT*, we can decide whether there are orthogonal vectors or

not. We can produce the strings in $O(n \cdot poly(d))$ and we can check the result in constant time. Therefore we have proven that a strongly subquadratic algorithm for *EDIT* would induce a strongly subquadratic ($O(n^{2-\epsilon} \cdot poly(d))$, to be exact) algorithm for *OVP*, which assuming $d = m = \omega(logn)$ would in turn contradict *SETH*.

5.1 SETH

It remains to be discussed whether *SETH* is a reasonable assumption.

On one hand, compared to $P \neq NP$, which is accepted by almost everyone as a valid assumption, the much stronger *SETH* is believed to be false by way more researchers. This stems partly from the large complexity gap between $poly(n)$ and 2^n. *SAT* could be anywhere in between, for example $\Omega(2^{n/2})$, which is strongly subexponential, but still not in P.

On the other hand, many problems have been reduced onto *SETH*, providing us with lower bounds that match the current state of research. These include other types of string distances, graph problems like Dominating Set and connectivity, and matrix problems. [BI15] even calls this set of problems "SETH-hard". All of this seems to support *SETH* as a valid root for complexity arguments.

References

[BI15] Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). *ACM Symposium on Theory of Computing*, 2015.

[BK15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 79–97, 2015.

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

[Wil05] Ryan Williams. A new alorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

[Wil18] Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity in computational geometry. *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1207–1215, 2018.