Subquadratic Edit Distance Approximations

Ashley Lin, James Lin, David Wu
ashlin@mit.edu, jameslin@mit.edu, dxwu@mit.edu

May 26, 2021

Abstract

Edit distance is a common measure for the similarity of two strings, counting the minimum number of insertions, deletions, and substitutions required to transform one string into the other. A straightforward dynamic program can compute edit distance in quadratic time, but strong hardness bounds have been proven against the existence of a subquadratic time solution. In lieu of this, in this paper we focus on approximations of edit distances in subquadratic time. we first describe the breakthrough work of Chakraborty et al. [6], which develops the first truly subquadratic time algorithm approximating edit distance to a constant factor, and then the related work of Boroujeni et. al [5], which improves the approximation factor to $1 + o(1)$ in a smoothed setting.

1 Introduction

A common problem in computer science is measuring the "similarity" between two long strings, for some definition of similarity. For instance, in computational biology, DNA and protein sequences are compared to identify similar regions, giving insight into evolutionary and structural relationships. Further applications include speech recognition, information extraction, and machine translation.

In this paper, we focus on *edit distance* (also *Levenhstein distance*), written $\text{ed}(s, s')$, between two strings s and s', which measures the least number of insertions, deletions, or substitutions required to modify a string s to s'. This is also related to the well-known *Longest Common Subsequence* (LCS) problem, which measures the length of the longest (not necessarily contiguous) subsequence $\text{LCS}(s, s')$ between two strings. In fact, computing the LCS is equivalent to computing the modified edit distance $\text{ed}'(s, s')$ with **no substitutions**, due to the fact that $\text{ed}'(s, s') = |s| + |s'| - \text{LCS}(s, s')$ [1].

Assuming the two strings both have length n, then the edit distance can be computed deterministically in quadratic time with dynamic programming, as seen in Figure 1.

For $0 \le i \le n$:

$$T[0][i] = T[i][0] = i$$

For $i, j \ge 1$:

$$T[i][j] = \min \begin{cases} T[i-1][j] + 1 \\ T[i][j-1] + 1 \\ T[i-1][j-1] + \mathbb{1}_{s[i] \neq s'[j]} \end{cases}$$

The edit distance is the value of $T[n][n]$.

*	8	8	7	6	6	6	6	6	5
*	7	7	6	5	5	5	5	5	4
R	6	6	5	4	4	4	4	4	3
E	5	5	4	3	3	3	4	3	4
T	4	4	3	2	2	3	3	4	5
S	3	3	2	1	2	3	4	5	6
I	2	2	1	2	3	4	5	6	7
M	1	1	2	3	4	5	6	7	8
@	0	1	2	3	4	5	6	7	8
	@	D	I	S	A	S	T	E	R

Figure 1: On the left is the recursive definition of the dynamic programming solution, where $s[i]$ denotes the ith character of s (one-indexed). A visualization is on the right for the words *DISASTER* and *MISTER*, after adding a new symbol '*' to ensure both words have the same length of 8.

Landau et al. [7] present a deterministic algorithm which improves on the $O(n^2)$ runtime for small solution sizes — specifically, their algorithm runs in time $O(n + \mathrm{ed}(s, s')^2)$. This algorithm is used as a subroutine for both of the algorithms we survey in the paper, but since the algorithm is deterministic and quite complex, we will black box it.

Backurs and Indyk [3] proved that a *truly* sub-quadratic algorithm (meaning a runtime of $(O(n^{2-\delta})$ for some $\delta > 0$, and not just subquadratic by polylog factors) would imply a $2^{(1-\gamma)n}$ runtime algorithm for k-SAT for some $\gamma > 0$, contradicting the Strong Exponential Time Hypothesis (SETH). Although SETH is believed to be true, it will certainly not be proven anytime soon – it is a stronger version of the Exponential Time Hypothesis that k-SAT runs in exponential time, which is itself stronger than the still unproven $P \neq NP$ that says k-SAT does not run in polynomial time. In fact, Abboud et al. [2] proved that even the easier task of shaving arbitrarily many polylog factors from the $\theta(n^2)$ runtime would imply the unproven result that $NEXP$ does not have uniform NC^1 circuits.

Of course, in the absence of more efficient exact algorithms, we are also interested in fast approximation algorithms of edit distance. For example, the aforementioned algorithm due to Landau achieves a $\sqrt{n}$-approximation in linear time, but fails to achieve a constant approximation in subquadratic time. In Section 3, we describe the first subquadratic time algorithm in 2019 due to Chakraborty et al. [6] that is able to approximate edit distance to within a constant factor, which has been improved to a factor $3 + \epsilon$ [4]. Finally, in Section 4, we describe how this is improved to a $1 + o(1)$ approximation algorithm by Boroujeni et al. [5] assuming a relatively mild extra condition on the initial strings s and s'.

2 High-Level Overview

We begin with a high-level overview covering both the algorithms presented in [6] and [5], as they rely on a similar framework. For simplicity, we assume that the input strings s and s' have the same length n, for n a power of two, by padding a new symbol at the end of the strings as necessary, though both algorithms can be generalized to remove this assumption. In both cases we preprocess by using Landau's deterministic algorithm [7] to find asymptotically small solutions (of size $n^{4/5}$ in [2] and $n^{37/39}$ in [5]) if they exist.

We can view the dynamic programming solution from Figure 1 as a path finding algorithm where the cost of horizontal and vertical edges is always 1, and the cost of diagonal edges depends on whether the characters at the corresponding indices in the strings match. From our preprocessing, we know that not too many diagonal edges have cost 0. The primary intuition is that in an approximation algorithm, we can find "approximate" diagonal edges across larger regions of the graph which upper bound the true cost of traversing that region.

More formally, we break up each string into substrings, which we call *windows*. Further, we consider the intersection of a window w in s (along the horizontal axis) and a window w' in s' (along the vertical axis) to define a *box*, and attempt to approximate the solution to the subproblem $\mathrm{ed}(w, w')$ for each box we consider. Since we only want an approximate solution for each box, we can discretize our search space – in particular, for an approximation factor of $1 + \epsilon$, it is sufficient to find all boxes for which the edit distance is at most the threshold $(1 + \epsilon)^k$ for increasing integers k. To do so efficiently, we use sparsification; essentially, for each value of k we make a distinction between windows w from s which correspond to many boxes within the threshold edit distance, and windows in s which are not. In [6] this distinction is called *dense* versus *sparse*, while in [5] this distinction is called *high degree* versus *low degree*. In either case, we improve our runtime by doing less work for each window which is dense, and considering "larger" windows for windows which are sparse. Together, these optimizations allow us to solve all the needed subproblems in subquadratic time.

Then the number of boxes we approximate is subquadratic, so we can run a modified dynamic program over these edges to obtain a subquadratic algorithm overall. In fact, this runtime can be improved (with a tradeoff in the approximation factor) by recursively running the same algorithm as a subprocess when solving the subproblem of finding the edit distance between windows.

3 Approximating Edit Distance to a Constant Factor

The main result of [6] is a randomized algorithm to compute the edit distance $\mathrm{ed}(s, s')$ up to a factor of 1680 in $\tilde{O}(n^{12/7})$ time when $\mathrm{ed}(s, s') \geq n^{4/5}$. More specifically, this is stated as the following theorem:

Theorem 3.1. *For every $\theta \geq n^{-1/5}$, there is a randomized algorithm with runtime $\tilde{O}(n^{2-2/7}\theta^{4/7})$ that takes two strings s and s' for which $\mathrm{ed}(s, s') \leq \theta n$, and outputs a number u such that $u \leq 840\theta n$ with probability at least $1 - n^{-7}$.*

By taking $\theta = 2^{-i}$ with $i \in \{0, 1, \ldots, \frac{\log n}{5}\}$, this implies that if the edit distance satisfies $\mathrm{ed}(s, s') \geq n^{4/5}$, then we have a randomized subquadratic algorithm that can determine the edit distance within a factor of $2 \cdot 840 = 1680$. This provides a subquadratic runtime when combined with Landau's algorithm (which we recall runs in $O(n + \mathrm{ed}(s, s')^2)$ time) in the case when $\mathrm{ed}(s, s') \leq n^{4/5}$.

We adapt the grid setting for the problem from the dynamic programming solution in Section 1 – namely, we place s on the horizontal axis and s' on the vertical axis. We can then consider the graph as seen on the left side of Figure 2, where gray edges have weight 1 and red edges have weight 0, and we wish to find the *minimal source-sink path* τ from the bottom left corner with vertex $(0, 0)$ to the top right corner at vertex (n, n).

3.1 An Overview of the Covering Algorithm

As specified in section 2, we split s and s' into *windows* so that we can focus on smaller *boxes* as subproblems. The key is to find a sequence of boxes that "approximate" the minimal path τ (as seen in Figure 3) in subquadratic time, known as the *covering algorithm*. We then quickly compute a path that is a constant factor approximation of the edit distance for each of those boxes. Then, the *approximated path* formed by stitching together these subpaths with vertical segments (shown as the blue line in Figure 3) gives a constant approximation of the edit distance of τ. With more care, this approximate path can be improved via a more complex construction that we choose to omit.

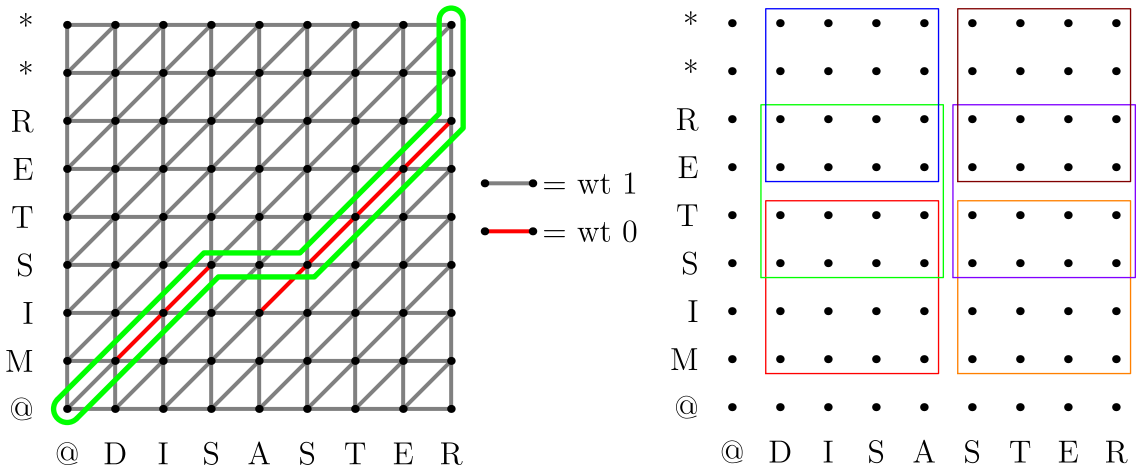

Figure 2: On the left, one minimal path τ is circled in green, giving the edit distance of 5 as seen in Figure 1. This is formed by substituting the 'D' in "DISASTER" into an 'M', deleting the substring "AS", and then inserting the new symbols "**" to the end. On the right, example *boxes* are shown in the case when $n = 8$, $w_1 = 4$, and $\theta = 1/2$.

The first step is to divide the grid into (overlapping) *boxes*. We partition s into $\ell = n/w_1$ **non-overlapping** windows $s_1, \ldots, s_\ell$ of size w_1. For some $\theta < 1$, we also divide s' into $\ell/\theta = n/(\theta w_1)$ **overlapping** windows $s'_1, \ldots, s'_{\ell/\theta}$ of size w_1 – each with bottom and top edges aligned at multiples of θw_1 (see Figure 2). A *box* is then of the form $s_i \times s'_{j_i}$. To be more explicit about what we mean by an "approximate" sequence of these boxes, we first need a few definitions:

Definition 3.2. *Let $s_i \times s'_{j_i}$ be a box with side length w_1, and let κ be an upper bound on the box's edit distance $\mathrm{ed}(s_i, s'_{j_i})$. Then $(s_i \times s'_{j_i}, \kappa)$ is a* ***certified box****. Furthermore, if $\kappa \le c\,\mathrm{ed}(s_i, s'_{j_i})$ for a constant c, then say that $s_i \times s'_{j_i}$ is* ***certified within a factor of*** *c.*

Now, we describe the "approximate" sequence consisting of these certified boxes:

Definition 3.3. *An* ***adequate approximating sequence*** *is a sequence of certified boxes $(s_1 \times s'_{j_1}, \kappa_1), (s_2 \times s'_{j_2}, \kappa_2), \ldots, (s_\ell \times s'_{j_\ell}, \kappa_\ell)$ such that*

- *$s_i \times s'_{j_i}$ is an* ***adequate cover*** *of τ_{s_i}, the minimal subpath of τ in window s_i. This means that the vertical distance from the start (resp. end) vertex of τ_{s_i} and the bottom left (resp. top right) corner of τ_{s_i} is at most $c_1(cost(\tau_{s_i}) + \theta w_1)$ for some constant c_1 (see Figure 3). Here, $cost(\tau_{s_i})$ is just the sum of the edge weights along τ_{s_i}.*
- *The sequence of boxes $s_i \times s'_{j_i}$ is* ***adequately bounded****, so that $\sum_i \kappa_i \le c_2(\mathrm{ed}(s, s') + \theta n)$ for some constant c_2.*

Note that both c_1 and c_2 are fixed over all θ and possible strings s, s'.

Then this will give an approximating path of length at most $(2c_1 + c_2)(\mathrm{ed}(s, s') + \theta n)$. Given inputs s and s' for which $\mathrm{ed}(s, s') \le \theta n$, then this approximation path has length at most $2(2c_1 + c_2)\theta n$. By testing $\theta \in 2^{-i}$ for $i \in \{0, 1, \ldots, \log n\}$, this gives a $4(2c_1 + c_2)$-approximation of edit distance.

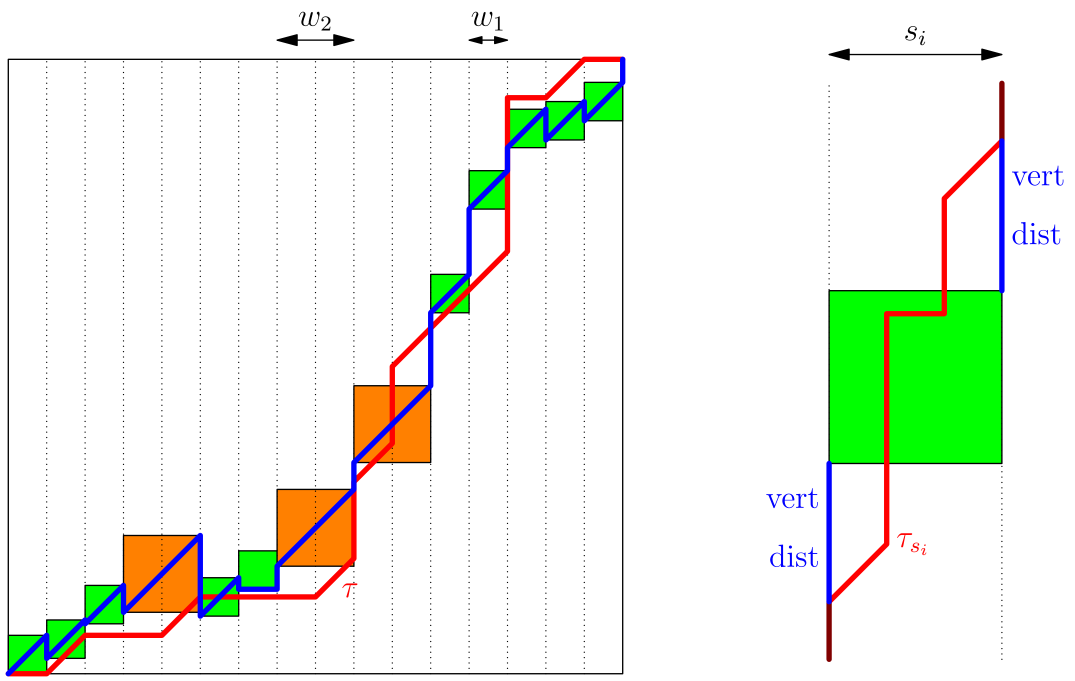

Figure 3: On the left is an adequate approximating sequence, with the approximating path formed by stitching together paths of certified boxes with vertical segments. In addition to boxes of side length w_1, there are boxes of side length w_2 which are introduced in Subsection 3.3. On the right, a certified box $s_i \times s'_j$ is in the window s_i, along with the minimal subpath τ_{s_i}. The "height" of τ_{s_i} is $O(\mathrm{cost}(\tau_{s_i}))$, hence why the vertical distances in the definition of an *adequate cover* are allowed to be $O(\mathrm{cost}(\tau_{s_i}))$.

It can be shown that for a sufficient choice of c_1, then for each horizontal window s_i there is a vertical window s'_{j_i} for which $s_i \times s'_{j_i}$ is an *adequate cover* of s_i. Naively, we could compute the edit distances of each $s_i \times s'_{j_i}$ box, which can be done via the deterministic dynamic program in $O(w_1^2)$ time. There are a total of $\frac{n}{w_1} \cdot \frac{n}{\theta w_1}$ boxes, which gives a total of $O(\frac{n^2}{\theta})$ computations, which is even worse than quadratic runtime. So we need to reduce the amount of work we are doing, and here is where we take advantage of the fact that we are considering smaller boxes. Recall that we do not necessarily need to compute the exact $\mathrm{ed}(s_i, s'_{j_i})$ of each box, but we just need to *certify* each box within a factor of c for some constant c fixed over all possible boxes, θ, and input strings. So there are two ways to reduce the runtime:

- Reducing the amortized runtime to certify boxes to $o(w_1^2)$.
- Reducing the number of boxes that need to be certified to $o\left(\frac{n^2}{\theta(w_1)^2}\right)$.

It turns out that it's not always possible for either of these two methods to cover all possible inputs s and s'. The insight is that we can use a combination of these two ideas to efficiently certify an adequate cover $s_i \times s'_{j_i}$ for each i.

3.2 Reducing Amortized Runtime of Certifying Boxes

We rely on the fact that edit distance satisfies the *triangle inequality*:

Lemma 3.4 (Triangle Inequality of Edit Distance)**.** *For any strings s_1, s_2, and s_3, we have that*

$$\mathrm{ed}(s_1, s_3) \leq \mathrm{ed}(s_1, s_2) + \mathrm{ed}(s_2, s_3).$$

Note that for any box $s_i \times s'_j$, it's true that $\mathrm{ed}(s_i, s'_j) \leq w_1$. We then will proceed to certify boxes in iterations. In the kth iteration, if $\mathrm{ed}(s_i, s'_j) \leq \epsilon_k$ where $\epsilon_k = w_1 2^{-k}$ for some $1 \leq j \leq \ell/\theta$ and $k \in \{0, 1, \ldots, \log w_1\}$, then we will show that we can certify the box $(s_i \times s'_j, 5\epsilon_k)$. Over all k iterations, this determine the adequate cover for s_i with $c_1 = 10$ for all i. This procedure can be done in a way that can significantly decrease the amortized runtime to certify boxes, and is outlined explicitly in Algorithm 1. We first introduce a few definitions that will be useful in introducing the algorithm.

Definition 3.5. *Given a string z of length w_1, let $T(z, \rho)$ denote the set of windows s_i of s for which* $\mathrm{ed}(z, s_i) \leq \rho$. *Similarly, let $T'(z, \rho)$ denote the set of windows s'_j of s' for which* $\mathrm{ed}(z, s'_j) \leq \rho$.

Definition 3.6. *Given a fixed ϵ_k, say that a window s_i is d-**dense** if $|T'(s_i, \epsilon_k)| \geq d$. Otherwise, say that s_i is d-**sparse**.*

Now we present the algorithm to efficiently certify boxes:

Algorithm 1: Efficiently Certifying Boxes

```
Result: Given a fixed $\epsilon_k$, be able to certify a box $s_i \times s'_{j_i}$ that is an adequate cover, for all
        $1 \leq i \leq \ell$.
Initialization: All windows $s_i$ are unfulfilled for $1 \leq i \leq \ell$;
while There exists at least one unfulfilled window $s_i$ do
    Choose an unfulfilled window $s_i$ to be the pivot;
    for $1 \leq i_0 \leq \ell$ do
        Compute $\mathrm{ed}(s_i, s_{i_0})$ to find $T(s_i, 2\epsilon_k)$;
    end
    for $1 \leq j \leq \ell/\theta$ do
        Compute $\mathrm{ed}(s_i, s'_j)$ to find $T'(s_i, 3\epsilon_k)$;
    end
    for $s_{i_0} \in T(s_i, 2\epsilon_k)$ do
        for $s'_j \in T'(s_i, 3\epsilon_k)$ do
            Certify each box $s_{i_0} \times s'_j$ as $(s_{i_0} \times s'_j, 5\epsilon)$;
        end
    end
    for $s_{i_0} \in S_i$ do
        Mark $s_{i_0}$ as fulfilled;
    end
end
```

We now verify Algorithm 1. Given a pivot s_i and any (possibly the same) window s_{i_0} in $T(s_i, 2\epsilon_k)$, we consider the set $T'(s_i, 3\epsilon_k)$. By the triangle inequality (Lemma 3.4), $T'(s_{i_0}, \epsilon_k) \subset T'(s_i, 3\epsilon_k)$ since for any s'_j in $T'(s_{i_0}, \epsilon_k)$, we have $3\epsilon_k = 2\epsilon_k + \epsilon_k \geq \text{ed}(s_i, s_{i_0}) + \text{ed}(s_{i_0}, s'_j) \geq \text{ed}(s_i, s'_j)$. Furthermore, $T'(s_i, 3\epsilon_k) \subset T'(s_{i_0}, 5\epsilon_k)$ since for any s'_j in $T'(s_i, 3\epsilon_k)$, $5\epsilon_k = 2\epsilon_k + 3\epsilon_k \geq \text{ed}(s_{i_0}, s_i) + \text{ed}(s_i, s'_j) \geq \text{ed}(s_{i_0}, s'_j)$. So $T'(s_{i_0}, \epsilon_k) \subset T'(s_i, 3\epsilon_k) \subset T'(s_{i_0}, 5\epsilon_k)$, which shows that all the boxes $s_{i_0} \times s'_j$ with $\text{ed}(s_{i_0}, s_j) \leq \epsilon_k$ is certified as $(s_{i_0} \times s'_j, 5\epsilon_k)$ in Algorithm 1.

Lemma 3.7. *Let d be such that all windows s_i of the string s are d-dense. Then Algorithm 1 will run in $O(\frac{n^2}{d\theta^2})$ time. In particular, if $d = \omega(\theta^{-2})$, then this is subquadratic time.*

Proof. For each iteration of the while loop, after a pivot s_i is selected, the runtime is $O(w_1^2 \cdot \frac{n}{\theta w_1}) = O(\frac{nw_1}{\theta})$, dominated by the time to compute the edit distance of all the boxes in window s_i. Now, since given any two pivots s_{i_1} and s_{i_2}, we have $\text{ed}(s_{i_1}, s_{i_2}) \geq 2\epsilon_k$ by necessity, so it follows that the sets $T'(s_{i_1}, \epsilon_k)$ and $T'(s_{i_2}, \epsilon_k)$ of the windows of s' are disjoint. Since there are a total of $n/(\theta w_1)$ windows of s', there are at most $n/(dw_1\theta)$ pivots, for a total runtime of $O(\frac{nw_1}{\theta} \cdot \frac{n}{\theta w_1 d}) = O(\frac{n^2}{d\theta^2})$ time, as desired. □

We can iteratively run Algorithm 1 in $\log w_1 + 1$ rounds for ϵ_k over all $k = 0, 1, \ldots, \log w_1$. For each window s_i, we choose a box $s_i \times s'_j$ that is certified in iteration k_i such that no other box in s_i's window was certified in an iteration after k_i. Then these boxes (s_i, s'_j) produces an *adequate approximating sequence* of certified boxes (where we omit the computations to show that the sequence is adequately bounded), and given that all windows are d-dense over all iterations k, then Algorithm 1 runs in $O(\frac{n^2 \log n}{\theta^2 d})$ time, which is subquadratic given the right choice of parameters for d and θ. Unfortunately, the fact that s_i is d-dense for all windows s_i and over all of our iterations k is simply not true. For example, for iterations of large k where ϵ_k is small, this can fail to hold for all windows s_i. So while running Algorithm 1, if we encounter a pivot s_i that is d-dense, then we may proceed as described in the algorithm. But when s_i is d-sparse, then we will actually edit Algorithm 1 to use the *diagonal extension algorithm* as described in the next subsection. This will reduce the work in certifying boxes by reducing the number of boxes that need to be certified instead of reducing the amortized runtime of certifying boxes, thus allowing the algorithm to run in subquadratic time when again given the right choice of parameters.

One issue is that we need to be able to determine if a pivot s_i is d-dense or d-sparse for a given ϵ_k before computing the $\text{ed}(s_i, s'_j)$ for all windows s'_j of s', since for a d-sparse pivot, computing $n/(\theta w_1)$ edit distances in $O(nw_1/\theta)$ time certifies more boxes and takes longer than what we will allow for the diagonal extension algorithm. This is where the power of randomized algorithms come in – more specifically, we will randomly sample windows s'_j of s', and use that to estimate $|T'(s_i, \epsilon_k)|$. We wish to test if $|T'(s_i, \epsilon_k)| \geq d$, that is, if a $\frac{d}{n/(\theta w_1)} = \frac{\theta w_1 d}{n}$ fraction of the windows s'_j of s' satisfy $\text{ed}(s_i, s'_j) \leq \epsilon_k$. To allow for sampling error, we will say a pivot is *declared dense* if at least $p = \frac{\theta w_1 d}{2n}$ of the sampled s'_j satisfy $\text{ed}(s_i, s'_j) \leq \epsilon_k$, and otherwise the pivot is *declared sparse*. Then we require $\theta(\frac{1}{p}) = \theta(\frac{n}{\theta w_1 d})$ samples so that with high probability, all d-dense pivots are declared dense, and that all $d/4$-sparse pivots are declared sparse. So for *declared dense* pivots, we run Algorithm 1 as is, and over all such iterations the total runtime will be subquadratic with the correct parameters. For all declared sparse pivots, we will instead run the diagonal extension algorithm which can run in subquadratic time as well.

3.3 Reducing Number of Boxes to be Certified for Sparse Pivots

We now decrease the runtime by considering *large boxes* of side length $w_2 > w_1/\theta$, and now refer to boxes of side length w_1 as *small boxes*. We now partition s into n/w_2 non-overlapping windows $t_1, \ldots, t_{n/w_2}$ of size w_2, and also divide s' into $n/(\theta w_2)$ overlapping windows $t'_1, \ldots, t'_{n/(\theta w_2)}$ of size w_2, with both top and bottom edges at multiples of θw_2. These are referred to as *large windows*, and windows of length w_1 are *small windows*. We redefine Definition 3.2 to allow for certifying large boxes $t_i \times t'_j$. Definition 3.3 is adjusted accordingly, so an *adequate approximating sequence* is now a sequence of small and large certified boxes, and $t_i \times t'_j$ is an *adequate cover* of the minimal subpath τ_{t_i} of τ in a large window t_i if the corresponding vertical distances are at most $c_1(\text{cost}(\tau_{t_i}) + \theta w_2)$ for some constant c_1.

We will be able to bootstrap certifying a small box into certifying a large box. More explicitly, we define the *diagonal extension* of a small box $s_i \times s'_j$ to a large box $t_i \times t'_j$ which contains the small box:

Definition 3.8. *Let $s_i \times s'_j$ be a small box, and t_i be a large window of s such that s_i is a substring of t_i. Then the large box $t_i \times t'_j$ is the **diagonal extension** of $s_i \times s'_j$ if the main diagonal of $s_i \times s'_j$ is a subsegment of the main diagonal of $t_i \times t'_j$ (see Figure 4).*

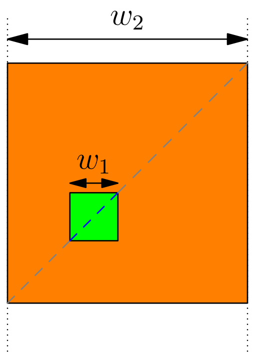

Figure 4: The diagonal extension of a box $s_i \times s'_j$ to a box in the window t_i.

Note that there may be issues with the definition of a diagonal extension of a box $s_i \times s'_j$ near the bottom and top of the grid that we choose to ignore for the sake of simplicity. We are now ready to introduce the Diagonal Extension Algorithm:

Algorithm 2: Diagonal Extension Algorithm

```
Result: Given a declared sparse small window $s_i$, let $t_i$ be the large window of $s$ for which $s_i$ is
        a substring of $t_i$. We find the diagonal extension of some small box $s_{i_0} \times s'_{j_0}$ (possibly
        not in the window $s_i$) inside the large window $t_i$ which gives a certified large box
        $t_i \times t'_j$ that is an adequate cover of $\tau_{t_i}$, the minimal subpath of $\tau$ in $t_i$.
Initialization: A declared sparse small window $s_i$ inside a large window $t_i$;
For some constant $c$, randomly choose $c(\log n)^2$ windows $s_{i_1}, \dots, s_{i_{c(\log n)^2}}$ among the $w_2/w_1$
  small windows within $t_i$;
for $1 \le i_x \le c(\log n)^2$ do
    for $1 \le s'_j \le \ell/\theta$ do
        Compute $\mathrm{ed}(s_{i_x}, s'_j)$;
    end
    Store the set $J_{i_x}$ of small windows $s'_j$ of $s'$ which give the $d$ smallest edit distances among
      $\mathrm{ed}(s_{i_x}, s'_j)$ for this fixed $i_x$;
end
for $1 \le i_x \le c(\log n)^2$ do
    foreach window $s'_{j_{i_x}}$ of $J_{i_x}$ do
        Compute the edit distance of the diagonal extension $t_i \times t'_j$ of each small box $s_{i_x} \times s'_{j_{i_x}}$,
          and certify these large boxes with the exact edit distance;
    end
end
One of the large boxes $t_i \times t'_j$ of the $c(\log n)^2 \cdot d$ diagonal extensions computed serves as the
  adequate cover of $t_i$ with high probability;
```

We omit the complex analysis and proof of correctness of Algorithm 2, instead opting to provide the high level intuition of why it should work. Recall that $|T'(s_i, \epsilon_k)| \le d$ is almost certainly true when s_i is declared sparse. If the certified large boxes produced from Algorithm 2 do not produce an adequate cover, then this means that all of the large boxes $t_i \times t'_j$ are far from the subpath τ_{t_i} of τ. This means that the $cd(\log n)^2$ small boxes $s_i \times s'_j$ whose diagonal extensions form these large boxes are likely to have relatively low edit distance, while still being far from the subpath τ_{s_i} within the small window s_i. Here lies the strength of s_i being declared sparse – there are few small boxes in s_i with low edit distance (as $|T'(s_i, \epsilon_k)| \leq d$), and these should tend to lie close to τ_{s_i}.

Finally, we analyze the runtime of Algorithm 2. Across the $c(\log n)^2$ small windows of s, it takes $\tilde{O}(w_1^2 \cdot \frac{n}{\theta w_1}) = \tilde{O}(\frac{nw_1}{\theta})$ time to compute the edit distances of all small boxes within those windows. It takes $\tilde{O}(dw_2^2)$ time to do the large box edit distance computations on the diagonal extensions. Algorithm 2 may be run up to n/w_2 times, one for each large window of s, for a total runtime for declared sparse pivots of $\tilde{O}(n^2 \frac{w_1}{\theta w_2} + ndw_2)$. Again, with suitable parameters, this is subquadratic. For the right choice of θ, picking $w_1 = \theta^{-2/7} n^{1/7}, w_2 = \theta^{1/7} n^{3/7}$, and $d = \theta^{3/7} n^{2/7}$ will balance the runtimes of Algorithm 1 for declared dense pivots and Algorithm 2 for declared sparse pivots so that the runtime of the overall Covering Algorithm is $\tilde{O}(n^{12/7}\theta^{4/7})$, as mentioned previously in Theorem 3.1.

4 Better Approximation Ratio in a Smoothed Setting

While there have been significant advances in constant factor approximation ratios up to $(3 + \epsilon)$ [4], it is considered unlikely that a triangle inequality based approach would beat a 2-approximation. As a step towards a better approximation ratio, Boroujeni et al. [5] analyze the following *smoothed setting*, i.e. a setting between average and worst case analysis. We first describe this smoothed setting, then give intuition for why these assumptions make the approximation task easier.

In our discussion of the edit distance problem so far, we require two input strings s, s'. In the smoothed setting, the adversary first chooses a string $\tilde{s}$. This string is randomly permuted, forming the first input string s. The adversary can then observe the smoothed output s and construct the second input s'. To see why this is harder than the average case analysis, i.e. when both input strings are generated uniformly from the alphabet, we show that this smoothed setting gives upper bounds on the edit distance for worst case inputs. In particular, for a worst case input (t, t'), if the adversary sets $\tilde{s} = t$, then s will be a permuted version of t. Then we can permute t' in the same way, and $\mathrm{ed}(t, t')$ will give an upper bound on $\mathrm{ed}(s, s')$.

Intuitively, we are introducing a limited amount of randomness in the input. This helps our analysis due to the meta-principle that random variables concentrate around their expectation. To that end, consider the following construction of a new random string $\bar{s}$. Take the *empirical distribution* $\hat{p}_s(\cdot)$ of characters in s, i.e. for $\sigma \in \Sigma$ we have

$$\hat{p}_s(\sigma) = \frac{1}{n} \sum_{i=1}^{n} 1_{s_i = \sigma}.$$

We sample n i.i.d. characters from this empirical distribution, i.e. $\bar{s} \sim \hat{p}_s^n$. This random string $\bar{s}$ is more convenient for analysis. In particular, the strategy is to instead study the edit distance problem for $\mathrm{ed}(\bar{s}, s')$, and then show that we can bootstrap this analysis for the original s. We omit the details of this reduction, but the high level idea is that with high probability the frequencies of characters in $\bar{s}$ and s are not too different.

In this simpler random setting, we can prove tight concentration properties about $\mathrm{ed}(\bar{s}, s')$. To formalize this intuition, we note that edit distance changes by at most 1 when we change one character in the input strings. By standard concentration inequalities (which we omit), we can derive the following lemma:

Lemma 4.1 (Concentration of edit distance)**.** *For arbitrary c, we have*

$$\mathbb{P}[|\mathrm{ed}(\bar{s}, s') - \mathbb{E}_{\bar{s}}[\mathrm{ed}(\bar{s}, s')]| \geq c] \leq 2 \exp\left(\frac{-2c^2}{|\bar{s}|}\right).$$

It is this property which is exploited to detect sparsity gaps in the window graph defined in Section 4.2.

4.1 Discretization Techniques

In order to achieve a $1+o(1)$ approximation ratio rather than a constant factor approximation ratio, our algorithm will be able to obtain a $1+\epsilon$ approximation ratio, and we can e.g. take $\epsilon = 1/\log n$. Since our final algorithm will be polynomial in $1/\epsilon$, this only changes the runtime by a polylog factor, which is negligible if our goal is a truly subquadratic algorithm.

To that end, we discretize our search space into integral powers of $(1+\epsilon)$. Then we will prove that our algorithm can detect in which interval $[(1+\epsilon)^k, (1+\epsilon)^{k+1})$ the answer lies in, thus implying that our answer will be at most $1+\epsilon$ off the true answer.

4.2 Windows and the Window Graph

Similar to the algorithm outlined in Section 3, the algorithm of Boroujeni et al. [5] attempts to solve the edit distance problems for smaller subproblems (windows). A window simply represents a substring of one of the input strings s, s'. We follow the notational convention that unprimed windows w are substrings of s and primed windows w' are substrings of s'. The algorithm breaks up the strings into two sets of windows W_s, $W_{s'}$. Although the windows can overlap, it is helpful for intuition to assume windows are disjoint; this distinction is only relevant for some technical portions of the algorithm which we omit.

A key object to analyze for the algorithm is a *window graph* with threshold ξ. A window graph is simply a bipartite graph separated into parts $W_s, W_{s'}$ whose nodes represent windows. Formally, the window graph G_ξ satisfies the property that (w_i, w'_j) is an edge if and only if $\mathrm{ed}(w_i, w'_j) \geq \xi$. At a high level, for an approximation ratio of $1+\epsilon$ (here $\epsilon > 0$ does not have to be constant) we will sweep out thresholds ξ as powers of $(1+\epsilon)$. Eventually, we will run a gap detecting algorithm which works by random sampling. For each ξ, we will detect if the graph is particularly sparse or not. Depending on the outcome, we will be able to leave behind a negligible fraction of edges, thus sparsifying the graph (with high probability).

More specifically, we have the following lemma, a simplified variant of Lemma 2.7 in [5]. To set it up, we consider window sets $\bar{W}$ and W', where each window (viewed as a string) is constructed by the process described in Section 4. In particular, the windows W' are adversarial, whereas $\bar{W}$ consists of n/m windows of size m, with each window $\bar{w} \in \bar{W}$ constructed by independently sampling m characters from the empirical distribution of s. Note that the windows in $\bar{W}$ are not actually substrings of $\bar{s}$.

Lemma 4.2. *Let $\bar{W}, W'$ be constructed as above. For $\epsilon > 0$, an edit distance threshold n^τ and violator threshold n^κ (under some technical conditions on ϵ, τ, and κ), with high probability, one of the two events holds:*

- *We have $\mathrm{ed}(\bar{w}, w') \geq (1+\epsilon)n^\tau$ for at most n^κ many random windows $\bar{w}$.*
- *We have $\mathrm{ed}(\bar{w}, w') \leq n^\tau/(1+\epsilon)$ for at most n^κ many random windows $\bar{w}$.*

Proof sketch. Denote $n' = |w'|$. We consider the set $\Sigma^{n'}$, the set of all strings of length n' formed by characters in Σ. These represent all possible adversarial windows that could be formed. We partition $\Sigma^{n'}$ into two sets L and R, based on whether the expected edit distance is small or not (with respect to the randomness of windows in $\bar{W}$). In particular, string $\ell \in \mathsf{L}$ if

$$\mathbb{E}_{\bar{w}}[\mathrm{ed}(\ell, \bar{w})] \leq n^\tau.$$

Here, $\bar{w}$ is a random string sampled from the empirical distribution as outlined earlier. The key is that as these strings ℓ are on average not too far from $\bar{w}$, they are unlikely in aggregate to be far from any $\bar{w} \in \bar{W}$. To that end, Lemma 4.1 implies that if ℓ is uniformly drawn from L, we have

$$\mathbb{P}[\mathrm{ed}(\ell, \bar{w}) > (1+\epsilon)n^\tau] \leq \exp\left(-\frac{2n^{2\tau}\epsilon^2}{m}\right). \tag{1}$$

Now we are in the home stretch. The idea is to define for ℓ randomly picked from L the bad event E_ℓ, which occurs whenever there are more than $n^{n/m}$ windows $\bar{w} \in \bar{W}$ such that $\mathrm{ed}(\ell, \bar{w}) > (1+\epsilon)n^\tau$.

But $\mathbb{P}[E_\ell]$ can be bounded by a combination of Equation (1) and the union bound. There are at most $n^{mn^{n/m}}$ ways to pick $n^{n/m}$ strings of size m, and by independence we can multiply failure probabilities. So we have by the union bound that

$$\mathbb{P}[E_\ell] \leq n^{mn^{n/m}} \exp\left(\frac{2\epsilon^2 n^{n/m+2\tau}}{m} \right).$$

Finally, for sufficiently large n, there are at most $n^{n'}$ strings of length n', so $|\mathsf{L}| \leq n^{n'}$. With some parameter tuning, we can show that $|\mathsf{L}| \cdot \mathbb{P}[E_\ell] = o(1)$, and conclude that event E_ℓ fails for at most n^κ choices of ℓ with high probability. $\square$

This lemma and the proof thereof motivates the following definitions.

Definition 4.3. *A window $w' \in W_{s'}$ is* high degree *with respect to threshold ξ and window size λ if the edit distance between w' and almost all windows $w \in W_s$ of size λ are at most $\xi(1+\epsilon)$. Similarly, w' is* low degree *if the edit distance between w' and almost all windows $w \in W_s$ of size λ is more than $\frac{\xi}{1+\epsilon}$.*

Lemma 4.2 implies that with high probability, almost all windows are either high degree or low degree for any ξ. This allows us to exhaustively search through the possible thresholds, as detailed by the discretization technique described in Section 4.1.

4.3 Putting the Pieces Together

We now go over the actual algorithm, going into slightly more detail than the meta level algorithm. We also highlight the differences between this algorithm and the algorithm described in previous sections. First, we explain the main parameters of the algorithm; in Section 4.4 we optimize the runtime as a function of these parameters.

- Two size parameters $x' < x$, both in the interval $(0, 1)$. Here, x controls the size of the windows and the number of windows. The parameter x' is used for the subroutine to handle low degree windows.
- A threshold $0 < \delta < 1$, which dictates when we use Landau et al's algorithm as described in Section 1, which runs in $O(n + n^{2\delta})$. When the solution size is small, we use this algorithm to compute the solution exactly. Hence our algorithm only needs to handle large solution sizes. For the optimal runtime, then, we expect $\delta \approx 1$.
- The parameter ϵ controls the approximation ratio, which is $1 + O(\epsilon)$. As described in Section 4.1, we take $\epsilon = 1/\log n$, so that we get a $1 + o(1)$ approximation ratio. Also, since the runtimes of the various subroutines only hide polylog factors of ϵ, they do not affect our ultimate polynomial upperbound runtime.

With this notation in mind, we can describe the algorithm. Throughout the description of the algorithm, we omit most of the exact runtime statements, as they are rather heavy in notation.

1. The first step is to precondition the algorithm by deterministically constructing windows for W_s and $W_{s'}$ which satisfy several nice properties. We omit the rather involved details of the construction, which may be found in [4]. The most important are:
   (1) An exact solution to edit distance between the windows W_s and $W_{s'}$ implies a truly subquadratic $1 + O(\epsilon)$ approximation of $\mathrm{ed}(s, s')$.
   (2) There are not too many total windows. More precisely, there are $\tilde{O}_\epsilon(n^{x+(1-\delta)})$ windows. Since we expect $\delta \approx 1$ in the optimal solution, this is sublinear in n.
   (3) The minimum and maximum window size are $(\epsilon n^{1-x}, n^{1-x})$, respectively.
   (4) There are only $\tilde{O}(1/\epsilon)$ different window sizes.

Let us comment on these properties. Clearly (1) is desirable, as it reduces the global edit distance problem to the window edit distance problem. Furthermore, it composes nicely with approximation algorithms for the window edit distance problem. (2) is also intuitive: if there are superlinearly many windows, then computing all edit distance pairs would be superquadratic with the naive algorithm. Properties (3) and (4) ensure that we do not need to run too many rounds of search for our algorithm with our discretization technique. More precisely, for handling high and low degree windows, we grid search through all thresholds ξ that are powers of $(1+\epsilon)$ and all window sizes λ. Then (3) and (4) ensure that there are in total $\tilde{O}_\epsilon(1)$ such combinations.

2. Next, we determine whether each window $w' \in W_{s'}$ is high degree or low degree with respect to a threshold ξ and window size λ using random sampling. We sample non-overlapping windows of W_s, each of length λ. For each of these sampled w, we compute $\mathrm{ed}(w, w')$, and take the median value. We report that w' is high degree if the median value is greater than $\xi(1+\epsilon)$, and analogously for reporting low degree.

3. Now we handle the high degree windows. In fact, there's nothing to do here — for a high degree window w' with respect to threshold ξ and window size λ, we just report all edges from w' to windows w of size λ. For correctness, we apply Lemma 4.2 to multiple nonoverlapping subsets. With a bit more work, one can show that we only report a small number of *false positives*, i.e. edges (w, w') that actually have $\mathrm{ed}(w, w') > \xi(1+\epsilon)$.

4. Next, we handle the low degree windows. Since the windows are low degree, we attempt to find almost all windows $w \in W_s$ of size λ with $\mathrm{ed}(w, w') < \epsilon$. Similarly to Section 3.3, we consider extensions, and there are two possibilities we have to deal with. Consider the following thought experiment. Using the construction of step 1, devise slightly larger windows of size $n^{1-x'}$ in step 1. Now consider the number of low degree windows that are contained in the same large window.

   - If there are not too many adjacent low degree windows, we ignore them altogether. Similarly to step 2, this produces negligible error.
   - Otherwise, when we sample random windows with probability $O(\frac{\log n}{\epsilon^2 n^{x-x'-(1-\delta)}})$, we hit one of the low degree windows whp. By trying all possibilities for ξ, λ we get an expected sample size of $\tilde{O}_\epsilon(n^{x'+2(1-\delta)})$.

5. Finally, we refine the solution via dynamic programming. We have omitted the details in this paper, but remark that this is the same as the dynamic program referenced in Section 3.1. However, a complication arises in that we might now find fake solutions due to the false positives from step 3. To remedy this, we check all the partial solutions in the DP solution, and count how many false positives there are. Since $\mathrm{ed}(w, w')$ is at most the maximum window size n^{1-x}, if there few false positives, then the error incurred to the global edit distance solution is also small, and we neglect to deal with these false positives.

   Otherwise, there are many false positives, and we remove all these false positive edges from the window graph and rerun the DP. We give a more formal runtime bound below.

4.4 Optimizing Runtime

A formal analysis provides precise runtimes in terms of the parameters described in Section 4.3. For the sake of expositional clarity, we omit these proofs, but we refer the interested reader to [5]. We can then combine the various hypotheses for the correctness of the algorithm. As each comes in the form of a linear constraint on these parameters. The runtime can thus optimized with an LP. When solved, we obtain an expected runtime of $O(n^{1.898})$, which is truly subquadratic. This can be strengthened to a high probability guarantee.

Theorem 4.4. *There exists a randomized algorithm which runs in expected $O(n^{1.898})$ for smoothed ED and approximates the solution to $1 + o(1)$. Furthermore, there is a randomized algorithm with the same big-O runtime and approximation factor that fails with probability at most $O(1/n^3)$.*

5 Conclusion

In this paper, we summarized key ideas from recent subquadratic time approximation algorithms for edit distance, first in the worst-case setting and then in a smoothed setting. The primary insight used in both papers is to divide the problem into subproblems and then use sparsification. A follow-up question might be whether there exists an alternative to Lemma 4.2 without relying on the assumption of the smoothed setting, which could motivate a similar $1 + o(1)$ approximation in the worst-case setting.

References

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78. IEEE, 2015.

[2] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 375–388, 2016.

[3] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58, 2015.

[4] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. *Journal of the ACM (JACM)*, 68(3):1–41, 2021.

[5] Mahdi Boroujeni, Masoud Seddighin, and Saeed Seddighin. Improved algorithms for edit distance and lcs: beyond worst case. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1601–1620. SIAM, 2020.

[6] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *Journal of the ACM (JACM)*, 67(6):1–22, 2020.

[7] Gad M Landau, Eugene W Myers, and Jeanette P Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.