Sublinear Algorithms for Gap Edit Distance

Elazar Goldenberg
The Academic College of Tel Aviv-Yaffo
Israel
elazargo@mta.ac.il

Robert Krauthgamer
Weizmann Institute of Science
Israel
robert.krauthgamer@weizmann.ac.il

Barna Saha
University of California Berkeley
USA
barnas@berkeley.edu

***Abstract*—The edit distance is a way of quantifying how similar two strings are to one another by counting the minimum number of character insertions, deletions, and substitutions required to transform one string into the other. A simple dynamic programming computes the edit distance between two strings of length n in $O(n^2)$ time, and a more sophisticated algorithm runs in time $O(n + t^2)$ when the edit distance is t [Landau, Myers and Schmidt, SICOMP 1998]. In pursuit of obtaining faster running time, the last couple of decades have seen a flurry of research on approximating edit distance, including polylogarithmic approximation in near-linear time [Andoni, Krauthgamer and Onak, FOCS 2010], and a constant-factor approximation in subquadratic time [Chakrabarty, Das, Goldenberg, Koucký and Saks, FOCS 2018].**

We study sublinear-time algorithms for small edit distance, which was investigated extensively because of its numerous applications. Our main result is an algorithm for distinguishing whether the edit distance is at most t or at least t^2 (the quadratic gap problem) in time $\tilde{O}(\frac{n}{t} + t^3)$. This time bound is sublinear roughly for all t in $[\omega(1), o(n^{1/3})]$, which was not known before. The best previous algorithms solve this problem in sublinear time only for $t = \omega(n^{1/3})$ [Andoni and Onak, STOC 2009].

Our algorithm is based on a new approach that adaptively switches between uniform sampling and reading contiguous blocks of the input strings. In contrast, all previous algorithms choose which coordinates to query non-adaptively. Moreover, it can be extended to solve the t vs $t^{2-\epsilon}$ gap problem in time $\tilde{O}(\frac{n}{t^{1-\epsilon}} + t^3)$.

***Keywords*-edit distance; sequence alignment; sublinear-time algorithms; sampling algorithms;**

I. Introduction

The *edit distance* (aka *Levenshtein distance*) [1] is a widely used distance measure between pairs of strings x, y over some alphabet Σ. It finds applications in several fields like computational biology, pattern recognition, text processing, information retrieval and many more. The edit distance between x and y, denoted by $\Delta_e(x, y)$, is defined as the minimum number of character insertions, deletions, and substitutions needed for converting x into y. Due to its immense applicability, the computational problem of computing the edit distance between two given strings x and $y \in \Sigma^n$ is of prime interest to researchers in various domains of computer science. A simple dynamic program solves this problem in time $O(n^2)$. Moreover, assuming the strong exponential time hypothesis (SETH), there does not exist any truly subquadratic algorithm for computing the edit distance [2], [3], [4], [5].

For many applications where the data is very large, a quadratic running time is prohibitive, and it is highly desirable to design faster algorithms, even approximate ones that compute a near-optimal solution. The last couple of decades have seen exciting developments in this frontier. In time $\tilde{O}(n^{1+\epsilon})$ for arbitrary $\epsilon > 0$, it is now possible to approximate the edit distance within factor $O(\log^{O(\frac{1}{\epsilon})} n)$ [6]. This bound was a culmination of earlier results where the approximation bound improved from $O(\sqrt{n})$ in linear time [7] to $O(n^{3/7})$ and $n^{1/3+o(1)}$ in quasi-linear time [8], [9], to $2^{O(\sqrt{\log n \log\log n})}$ in time $O(n2^{O(\sqrt{\log n \log\log n})})$ [10]. Recently, a breakthrough by Chakrabarty, Das, Goldenberg, Koucký and Saks [11] obtained the first constant factor approximation algorithm for computing edit distance with a subquadratic running time. However, when restricted to strictly linear time algorithms, a $\sqrt{n}$ approximation still remains the best possible [7], [12], [13]. In fact, when $\Delta_e(x, y) = t$, the algorithm by Landau, Myers and Schmidt runs in $O(n + t^2)$ time [7]. Thus for $t \leq \sqrt{n}$, the edit distance can be computed exactly in linear time. This algorithm has found widespread applications [14], [15], [16], [17] and is also known to be optimal under SETH.

Sublinear time: Following this quest for ever faster algorithms, it is natural to seek sublinear-time approximation algorithms. We study the regime of small edit distance t, which was investigated extensively in the literature because of its high relevance to many applications. In computational biology, for example, it

Part of this work was done while the authors were visiting the Simons Institute for the Theory of Computing.

R. Krauthgamer is partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, and a Minerva Foundation grant.

B. Saha is partially supported by a NSF CAREER Award 1652303, and an Alfred P. Sloan Fellowship.

2575-8454/19/$31.00 ©2019 IEEE
DOI 10.1109/FOCS.2019.00070

is often only necessary to compare genomic sequences that are highly similar and quickly get rid of sequences that are far apart, e.g., some sequencing projects target a strain or species that is closely related to an already-sequenced organism [18]. A major difficulty is that genomic sequences are comprised of highly repetitive patterns (repeats) whose frequency and placement contain important information about genetic variation, gene regulation, human disease condition, etc. [19], [20]. In a text corpora, detecting plagiarism and eliminating duplicates require identification of document pairs that are small edit distance apart. These applications can benefit from super-fast algorithms that answer whether the edit distance is below a threshold t or above $f(t)$ for some function f, known as the *gap edit distance* problem. The goal here is to design algorithms that are simultaneously highly efficient and have $f(t)$ as close to t as possible.

What is the right gap?: We focus on $f(t) = t^2$, i.e., a quadratic gap as our main test case. This is perhaps the most natural choice other than $f(t) = \Theta(t)$ (i.e., multiplicative approximation), and is also motivated by known results for linear and sublinear time.

- In linear time, the algorithm of [7] can solve the t vs t^2 gap problem. So far, no linear-time algorithm is known to beat this bound [12], [13]. Bar-Yossef, Jayram, Krauthgamer and Kumar [8] introduced the term *gap edit distance* and solved the t vs t^2 gap problem for non-repetitive strings. Their algorithm computes a constant-size sketch but still requires a linear pass over the data. This result was later improved to hold for general strings [13] via embedding into Hamming distance, but again in linear time.
- The study of sublinear-time algorithms for edit distance was initiated by Batu, Ergun, Kilian, Magen, Raskhodnikova, Rubinfeld and Sami [21], who designed an algorithm for the t vs $\Omega(n)$ gap problem, thereby solving the quadratic gap problem only for $t = \Omega(\sqrt{n})$. Currently, the best sublinear-time algorithm, by Andoni and Onak [10], solves the t vs t^2 gap problem for all $t = \omega(n^{1/3})$. For $t = n^{1/3+\epsilon}$, their running time is $n^{1-3\epsilon+o(1)}$. Solving the quadratic gap problem appears to become harder as t gets smaller because locating fewer edit operations will require more queries, and the approximation factor t gets smaller. (This is in contrast to the time bound $O(n + t^2)$ of [7].)

A. Results

We design a sublinear time algorithm for the t vs $f(t) = t^2$ gap problem. Its running time is $\tilde{O}(\frac{n}{t} + t^3)$, which is indeed sublinear for all $t \in [\tilde{\omega}(1), o(n^{1/3})]$.[1]

Theorem I.1. *There exists an algorithm that, given as input strings $x, y \in \{0,1\}^n$ and an integer $t \leq \sqrt{n}$, has query and time complexity bounded by $O(\frac{n \log n}{t} + t^3)$, and satisfies the following:*

- *If $\Delta_e(x,y) \leq t/2$ it outputs* `close` *with probability* 1.
- *If $\Delta_e(x,y) > 13t^2$ it outputs* `far` *with probability at least* 2/3.

Therefore, coupled with the result of [10], we get sublinear time-complexity for the quadratic gap problem for $t \in [\tilde{\omega}(1), o(n^{1/3})] \cup [\omega(n^{1/3}), n]$. This leaves a very interesting open question as to what happens when $t = \Theta(n^{1/3})$.

Our algorithm has two more nice features. First, sometimes one also requires that the algorithm finds an *alignment* of two strings: x and y, i.e., a series of edit operations that transform x into y. Our algorithm can succinctly represent an alignment in $\tilde{O}(t^2)$ bits even though it runs in sublinear time. Second, the algorithm can be easily extended to solve the t vs $f(t) = t^{2-\epsilon}$ gap problem by paying slightly higher in the running time/query complexity: $\tilde{O}(\frac{n}{t^{1-\epsilon}} + t^3)$.

Previous Work: Batu et al.'s algorithm distinguishes $t = n^\alpha$ vs $f(t) = \Omega(n)$ in $O(n^{\max\{2\alpha-1, \alpha/2\}})$ time for any fixed $\alpha > 1$ [21]. Their approach crucially depends on $f(t) = \Omega(n)$ and cannot distinguish between (say) $n^{0.1}$ and $n^{0.99}$. The best sublinear-time algorithm known for gap edit distance, by Andoni and Onak [10], distinguishes between $t = n^\alpha$ vs $f(t) = n^\beta$ for $\beta > \alpha$ in time $O(n^{2+\alpha-2\beta+o(1)})$. For the quadratic gap problem, i.e., $\beta = 2\alpha$, this time bound is $O(n^{2-3\alpha+o(1)})$, which becomes worse as t gets smaller (as discussed earlier). For example, when $t = n^{1/4}$, the known algorithm is not sublinear, whereas ours runs in time $\tilde{O}(n^{3/4})$.

Presence of repeated patterns make the gap edit distance problem significantly difficult to approximate. When no repetition is allowed, the state-of-the-art sublinear-time algorithms of [22] for the Ulam metric (edit distance with no repetition, which obviously requires a large alphabet) distinguish between t vs $\Theta(t)$ in $O(\frac{n}{t} + \sqrt{n})$ time, achieving a bound that is similar to the folklore sampling algorithm for approximating Hamming distance. There is a long line of work on edit distance and related problems, aiming to achieve fast running time [22], [23], [24], [25], [26], low distortion embedding [27], [28], [13], [29], small space complexity [13], [29], [8] and parallel algorithms [30]. The work of Andoni, Onak and Krauthgamer [6] achieves a sublinear asymmetric query complexity for approximating

[1] Throughout, the tilde notation $\tilde{O}(\cdot)$ and $\tilde{\omega}(\cdot)$ hide factors that are polylogarithmic in n.

edit distance; however it does not lead to any sublinear time algorithm since one of the strings must be read in its entirety.

B. Techniques

As a warmup, we start with a simple algorithm that has asymmetric query complexity – it queries $\tilde{O}(\frac{n}{t})$ positions in x, but may query the entire string y. This is comparable to the Hamming metric, where simply querying $\tilde{O}(\frac{n}{t})$ positions uniformly at random in x and the same positions in y, suffice to solve t vs $\Theta(t)$ gap. However, this simple uniform sampling fails miserably to estimate edit distance, even when there is a single character insertion or deletion. Our simple algorithm reads $\tilde{O}(\frac{n}{t})$ random positions in x, but since x_i might be matched to any y_{i+d}, $d \in [-t..+t]$, our algorithm has to read the entire string y. (In this outline, we call d a shift, and later call it a diagonal.)

Even when the entire string y is known, we cannot hope that this approach distinguishes better than t vs $f(t) = t^2$. To see this, consider two scenarios. In one scenario, y is obtained from x via t^2 substitutions. Since the algorithm samples x at a rate of $\frac{1}{t}$, we expect to see about t of these substitutions. In an alternative scenario, x is partitioned into t substrings of length $\frac{n}{t}$, and y is obtained from x by a circular shift by one position of each of the t parts (substrings). Now, the edit distance between x and y is at most $2t$, and assuming the sample of x contains at least one symbol from each part of x, the best alignment of the sampled x with y will still constitute of $O(t)$ insertions/deletions. These two cases will be indistinguishable to an algorithm that aligns the samples in x with the string y, and thus the best separation possible in this approach is t vs $f(t) = t^2$.

To avoid sampling the entire string y, one may need to sample x at a lower rate or to sample x non-uniformly in *contiguous positions* (blocks). In the former case, the separation between t and $f(t)$ will only increase. In the latter case, an algorithm that samples (say) $\frac{n}{t^2}$ blocks of length $O(t)$ in x can be shown to solve only a t vs t^2 gap even for Hamming distance, and for edit distance we will need $f(t) = t^3$.

In order to overcome these barriers, we employ both contiguous sampling and uniform sampling together, and in fact switch between them *adaptively*. The contiguous sampling suggests plausible shifts that a low-cost alignment may use. These plausible shifts are then checked probabilistically through uniform sampling. However, if we need to check every plausible shift via uniform sampling, the query (and time) complexity will again become linear. A technical observation based on [31] helps us here — if two substrings can be matched under two distinct shifts d and d', then the substrings must have a repeated pattern. In other words, the substrings are periodic with a pattern of length $|d - d'|$. The crux is that instead of checking each shift individually, we instead check for this repeated pattern via uniform sampling. When we witness a deviation from the periodicity (e.g., change in pattern), we execute a fast test to identify all shifts that "see" a mismatch (and increase our estimate of their cost). We alternate between the non-uniform and uniform sampling at an appropriate rate to achieve the desired query complexity and the running time.

In contrast, all previous sublinear/sampling algorithms, including [21], [10], [6], [22] choose which coordinates to query non-adaptively.

Organization: Section II lays the groundwork for our main algorithm. It starts by introducing (in Section II-A) the concept of a grid graph, which represents the edit distance as a shortest-path computation in a graph. It then describes (in Section II-B) the uniform sampling technique, which can be viewed as sampling of the grid graph, leading to a simple algorithm with asymmetric query complexity.

Section III presents our main result. It starts with a method (in Section III-A) that computes a shortest path using a more selective scan of the grid graph. It then describes (in Section III-C) our main algorithm, which combines the aforementioned techniques of sampling the grid graph and of scanning it more selectively.

II. Preliminaries: The Grid Graph and Uniform Sampling

Notation: Let $x \in \Sigma^n$ be a string of length n over alphabet Σ. For a set $S \subseteq [n]$, we denote by x_S the restriction of x to positions in S (in effect, we treat S as if it is ordered in the natural order). Oftentimes, S is contiguous (i.e., an interval) and then x_S is a substring of x. For $d \in [-n..n]$ and a set $S \subseteq [n]$ we define $S + d := \{s + d : s \in S\}$. As usual, $[i..j]$ denotes $\{i, \ldots, j\}$ for integers i, j.

A string $x \in \Sigma^n$ is called *periodic* with *period length* $m < n$ and *period pattern* $p \in \Sigma^m$ if $x = p^{\lfloor n/m \rfloor} \circ q$, where $q = p_{[1..(n \bmod p)]}$ and $\circ$ means concatenation of strings. Here and throughout we assume that $(n \bmod p)$ returns a value in the range $[1..p]$ (rather than $[0..p-1]$ as usual).

A. Edit Distance as a Shortest Path in a Grid Graph

Given an input $x, y \in \{0,1\}^n$ to the edit distance problem, it is natural to consider the following directed graph $G_{x,y}$, which we refer to as the grid graph. It has vertex set $[0..n] \times [-n..n]$, The graph has the following weighted edges (provided both endpoints are indeed vertices):

(i) Deletion edges: $(i,d) \to (i+1, d-1)$ with weight 1 corresponding to a character deletion.
(ii) Insertion edges: $(i,d) \to (i, d+1)$ with weight 1 corresponding to character insertions.
(iii) Matching/substitution edges: $(i,d) \to (i+1,d)$ with weight either 0 or 1 depending on whether $x_{i+1} = y_{i+d+1}$ or not. Such an edge corresponds to a character match/substitution.

See Figure 1 for illustration. Throughout, we call i the *row* and d the *diagonal* of a vertex (i,d), and refer to weight also as cost.

We assign to each vertex (i,d) a cost $c(i,d)$ according to the shortest path (meaning of minimum total weight) from $(0,0)$ to (i,d). The following two lemmas can be easily proven using standard dynamic programming arguments; we omit the details.

Lemma II.1. *One can compute the cost of every vertex in the grid graph by scanning the vertices in row order (i.e., from 0 to n) and inside each row scanning the diagonals in order from $-n$ to n. Moreover, computing the cost of each vertex (i,d) requires inspecting only 3 earlier vertices, namely, $(i-1,d)$, $(i-1,d+1)$ and $(i,d-1)$.*

Lemma II.2. *There is a one-to-one correspondence between paths from $(0,0)$ to (i,d) to alignments from $x_{[1..i]}$ into $y_{[1..i+d]}$ (an alignment is a set of character deletions, insertions and substitutions that converts one string to the other). Moreover, each path's weight is equal to the corresponding alignment's cost, and thus*

$$c(i,d) = \Delta_e(x_{[1..i]}, y_{[1..i+d]}).$$

Observe that when the edit distance is bounded by a parameter t, the optimal path goes only through vertices in $[0..n] \times [-t..t]$, and thus the algorithm can be restricted to this range. We sometimes refer to the two vertices $(0,0)$ and $(n,0)$ as the *source* and *sink*, respectively.

B. Uniform Sampling of the Rows (and Asymmetric Query Complexity)

As a warm-up, we now describe a simple randomized algorithm that, given as input two strings $x, y \in \{0,1\}^n$ and a parameter $t \le \sqrt{n}$, distinguishes (with high probability) whether $\Delta_e(x,y) \le t$ or $\Delta_e(x,y) = \Omega(t^2)$. The algorithm has asymmetric query complexity: it queries x at a rate of $\frac{\log n}{t}$ but may query y in its entirety.

This algorithm is based on a sampled version of the grid graph, denoted G_S, constructed as follows. First, pick a random set $S \subseteq [n]$ where each row $i \in [n]$ is included in S independently with probability $\frac{\log n}{t}$, and add also row 0 to S. Let $s = |S|$ and denote the rows in S by $0 = i_1 < \cdots < i_s$. Now let the vertex set of G_S be $S \times [-t..t] \cup \{(n,0)\}$, and connect each vertex (i_j, d), for $i_j \in S$ and $d \in [-t..t]$, to the following set of vertices (provided they are indeed vertices): (i) (i_{j+1}, d) with weight $\mathbb{1}_{\{x_{i_j+1} \neq y_{i_j+d+1}\}}$; and (ii) $(i_j, d+1)$ with weight 1 and (iii) $(i_{j+1}, d-1)$ with weight 1. Finally, connect each vertex (i_s, d) to the sink $(n,0)$ with an edge of weight $|d|$.

The algorithm constructs G_S and then computes the shortest path from $(0,0)$ to $(n,0)$. If its cost is at most t then the algorithm outputs `close`, otherwise it outputs `far`.

Lemma II.3. *For all $x, y \in \Sigma^n$,*

- *if $\Delta_e(x,y) \le t$ then with probability 1 the algorithm outputs `close`;*
- *if $\Delta_e(x,y) > 6t^2$ then with probability at least 2/3 the algorithm outputs `far`.*

We sketch here the proof, deferring details to Appendix A.

Close case $\Delta_e(x,y) \le t$: It suffices to show that for every source-to-sink path τ in $G_{x,y}$ (the original grid graph) there is in G_S a corresponding source-to-sink path τ_S that has the same or lower cost. To do this, start by letting τ_S visit the same set of vertices as τ visits in row 0, starting of course at the source $(0,0)$, and then extend τ_S iteratively from row i_j to i_{j+1}, as follows. Denote the last vertex visited by τ_S on row i_j by (i_j, d_S), and the vertices visited by τ on row i_{j+1} by $(i_{j+1}, d) \ldots, (i_{j+1}, d+\ell)$. Now if $d_S \le d+\ell$, then extend τ_S by appending $(i_{j+1}, d_S) \ldots, (i_{j+1}, d+\ell)$. Otherwise, extend it by appending $(i_{j+1}, d-1)$. Finally, after τ_S visits row i_S, extend it by appending the sink $(n,0)$.

Denote by $c_{G_{x,y}}(\cdot)$ the cost of a path in $G_{x,y}$, and by $c_{G_S}(\cdot)$ the cost of a path in G_S. We can then prove that $c_{G_S}(\tau_S) \le c_{G_{x,y}}(\tau)$, see Claim A.1 for details.

Far case $\Delta_e(x,y) \ge 6t^2$: We need the next claim, which follows easily by the independence in sampling rows to S. Let $\Delta_H(\cdot,\cdot)$ denote the Hamming distance between two strings.

Claim II.4. *Fix $x, y \in \Sigma^n$, $i \in [n]$ and $d \in [-t..t]$. Let $i' \ge i$ be the minimum such that*

$$\Delta_H(x_{[i..i']}, y_{[i..i']+d}) \ge 3t,$$

and let $H = \left\{ j \in [i..i'] : \; x_j \neq y_{j+d} \right\}$ be the corresponding set of Hamming errors. If no such i' exists then set $i' = \infty$.

Define $B(i,d)$ to be the event that $i' < \infty$ and that no row from H is sampled, i.e., $H \cap S = \emptyset$. Then

$$\Pr[B(i,d)] \le \left(1 - \frac{\log n}{t}\right)^{3t} < \frac{1}{3n(2t+1)}.$$

By a union bound over the set of all possible rows i and diagonals d, we get that except with probability

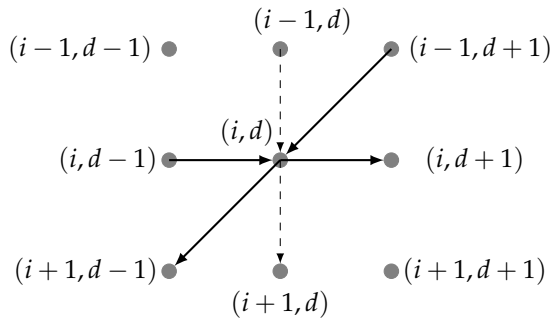

Figure 1. A typical vertex in $G_{x,y}$ has 3 incoming and 3 outgoing edges. Thick edges have cost 1 corresponding to deletion/insertion, dashed edges have cost 0/1 corresponding to substitution.

$n(2t+1)\frac{1}{3n(2t+1)} \leq \frac{1}{3}$, none of the events $B(i,d)$ happens. We conclude the proof by showing that whenever this happens, every path in G_S from the source $(0,0)$ to the sink $(n,0)$ has cost strictly larger than t, and therefore our algorithm outputs `far`.

C. Generalized Grid Graph

It is instructive, and in fact needed for the algorithm we describe in Section III-C, to consider the following generalization of G_S (and of G_{xy}). A *generalized grid graph* has the same vertices and edges as G_S (which was defined in Section II-B), except that the edges of type (iii) have arbitrary weights from the domain $\{0,1\}$. This is in contrast to G_S, where all these weights are derived from the two strings x,y and thus have various correlations, e.g., a single x_{i_j} affects the weight of many edges. The next lemma shows that such graphs have a lot of structure.

Lemma II.5. *Consider a generalized grid graph and denote its rows sequentially by $0,1,2,\ldots,|S|-1$. Then the cost difference between a vertex (i,d) and its in-neighbors is bounded by:*

$$0 \leq c(i,d) - c(i-1,d) \leq 1 \quad (1)$$
$$-1 \leq c(i,d) - c(i,d-1) \leq 1 \quad (2)$$
$$-1 \leq c(i,d) - c(i-1,d+1) \leq 1 \quad (3)$$

Proof: The three upper bounds are immediate from the triangle inequality, hence we only need to prove the three lower bounds.

We prove the lower bound in (1) by induction on the grid vertices (i,d) in lexicographic order (i.e., their row is the primary key and their diagonal is secondary). For the inductive step, consider (i,d) and assume the lower bound holds for all previous vertices. The cost of (i,d) is the minimum of three values coming from its in-neighbors, and at least one of these values is tight. We thus have three cases. In the first one, the value coming from in-neighbor $(i-1,d)$ is tight, then $c(i,d) - c(i-1,d) \in \{0,1\}$ and we are done. The second case is when the value coming from in-neighbor $(i,d-1)$ is tight. Using this fact, applying the induction hypothesis to $(i,d-1)$, and then the upper bound in (2), we have

$$c(i,d) = c(i,d-1)+1 \geq c(i-1,d-1)+1 \geq c(i-1,d),$$

as required. The third case, where the value coming from in-neighbor $(i-1,d+1)$ is tight, is proved similarly, and this concludes the proof of (1).

The lower bound in (2) follows easily by using the upper bound in (3) and then the monotonicity property (1), we indeed obtain $c(i,d-1) \leq c(i-1,d)+1 \leq c(i,d)+1$. The lower bound in (3) follows similarly $c(i-1,d+1) \leq c(i-1,d)+1 \leq c(i,d)+1$. ∎

III. Sublinear Algorithm for Quadratic Gap

In this section we present our main sublinear algorithm, which combines two techniques. The first one, explained in Section II, is to sample uniformly rows in the grid graph $G_{x,y}$ and compute a shortest path in the resulting (sampled) graph G_S. The second technique scans the grid graph more selectively in the sense of skipping some vertices in an adaptive manner. While this technique is known, e.g., from [7], our version (presented in Section III-A) differs from previous work because it scans the graph row by row. We then explain (in Section III-B) our main technical insight that allows to adaptively switch between the two aforementioned techniques, which correspond to uniform sampling and reading contiguous blocks (from x,y). We are then ready to present the algorithm itself (in Section III-C), followed by an analysis of its correctness and time/query complexity (in Section III-D), and a discussion of some extensions.

A. Selective Scan of the Grid Graph

We will make use of a technique developed in [32], [7], [33], [34] that scans the grid graph $G_{x,y}$ more selectively, and yet is guaranteed to compute a shortest

path from $(0,0)$ to $(n,0)$. By itself, this technique does not yield asymptotic improvement over a naive scan of all the vertices, however it is crucial to our actual algorithm (and also to the algorithm of [7], which uses a variant of this technique where the grid graph is scanned in "waves" rather than row by row). Along the way, we introduce three notions (dominated, potent and active) that may apply to a diagonal d at row i, i.e., to a vertex (i,d). To simplify the exposition, we do not discuss all the boundary cases, and objects that do not exist (like row -1 or character x_{n+1}) should be ignored (e.g., omitted from a minimization formula).

Dominated vertices: Let (i,d) be a vertex in $G_{x,y}$ of cost $h = c(i,d)$. If any of its in-neighbors $(i,d-1)$ and $(i-1,d+1)$ has cost $h-1$ then we say that (i,d) is *dominated* by that in-neighbor.

The following observation may allow us to "skip" a dominated vertex when computing a shortest path. Suppose that (i,d) is dominated by $(i,d-1)$, see Figure 2 for illustration (when dominated by $(i-1,d+1)$, an analogous argument applies). If diagonal $d-1$ *has a match* at row $i+1$, defined as $x_{i+1} = y_{i+1+(d-1)}$, then there exists a shortest path to $(i+1,d)$ that avoids vertex (i,d), by going for example through $(i,d-1) \to (i+1,d-1) \to (i+1,d)$. Notice that vertex $(i+1,d)$ must be dominated too. If, however, diagonal $d-1$ *has a mismatch* at row $i+1$, defined as $x_{i+1} \neq y_{i+1+(d-1)}$, then it might be that every shortest path to $(i+1,d)$ passes through (i,d). Notice that now $(i+1,d)$ may or may not be dominated.

Potent vertices: To formalize the above observation, we now define potent vertices and assert that it suffices to inspect only such vertices.

Definition III.1. *We say that diagonal d is* potent *at row i if it satisfies the two requirements:*

- *if (i,d) is dominated by $(i,d-1)$, then we require that diagonal $d-1$ is potent at row i and has a mismatch at row $i+1$; and*
- *if (i,d) is dominated by $(i-1,d+1)$ then we require that diagonal $d+1$ is potent at row $i-1$ and has a mismatch at row i.*

Notice that if (i,d) is not dominated, then both requirements are vacuous, and thus (i,d) is potent. In particular, the source $(0,0)$ is not dominated and thus potent.

The three lemmas below show that information whether vertices are potent is very useful for computing shortest paths from the source, i.e., vertex costs. Informally, the first lemma shows that our algorithm can restrict its attention to paths that consist of potent vertices (at least one, because the source is potent by definition) followed by non-potent vertices (zero or more). In particular, for a potent vertex, the path to it passes only through potent vertices. The next two lemmas show that information whether a vertex is potent or not can sometimes make the computation of vertex costs trivial. The proofs of all three lemmas appear in Appendix A.

Lemma III.2. *Every vertex in the grid graph $G_{x,y}$ has a shortest path from $(0,0)$, in which non-potent vertices appear only after potent vertices.*

Lemma III.3. *If (i,d) is non-potent, then $c(i+1,d) = c(i,d)$.*

Lemma III.4. *Suppose (i,d) is potent. If $(i+1,d)$ has a mismatch, then $c(i+1,d) = c(i,d)+1$. Otherwise (it has a match), $c(i+1,d) = c(i,d)$ and $(i+1,d)$ is potent.*

Remark III.5. *Lemmas III.2, III.3 and III.4 hold also for a generalized grid graph (as defined in Section II-C).*

The next challenge is to find the potent vertices algorithmically without scanning the entire grid graph.

Active vertices: We now describe an algorithm that computes iteratively for each row $i = 0,1,\ldots,n$ a list $\mathcal{D}_i$ of diagonals that we call *active* diagonals. The idea is that $\mathcal{D}_i$ will be a superset of the potent diagonals at row i, and that scanning each list $\mathcal{D}_i$ will create $\mathcal{D}_{i+1}$ for the next row. However, this description is an oversimplification, because $\mathcal{D}_i$ itself might change while being scanned, as explained next.

Formally, the algorithm starts with the following initialization. It sets $\mathcal{D}_0 = \{0\}$ and every other list $\mathcal{D}_i = \emptyset$. It then creates an array c_{A} to store the costs of every diagonal (at the current row, see more below), and sets $c_{\mathrm{A}}[d] = |d|$ for every $d \in [-n..n]$.

The algorithm then iterates over the rows $i = 0,1,\ldots,n$, where each iteration i scans $\mathcal{D}_i$ in increasing order. To process each scanned $d \in \mathcal{D}_i$, the algorithm first determines whether (i,d) is potent using Definition III.1, except for vertex $(0,0)$ which is potent by definition. This requires comparing the cost of (i,d) to its two in-neighbors. [2] We show in Lemma III.7 that at this point we have: $c_{\mathrm{A}}[d] = c(i,d)$, $c_{\mathrm{A}}[d-1] = c(i+1,d-1)$, and $c_{\mathrm{A}}[d+1] = c(i,d+1)$. In order to check whether (i,d) is potent, we need the values of $c(i,d-1)$ and $c(i-1,d+1)$ along with the information whether $(i,d-1)$ and $(i-1,d+1)$ are potent. By maintaining $c_{\mathrm{A}}[d]$ values for the last two computed rows and two simple boolean arrays to indicate whether a diagonal d' is potent at the current row and the previous row (by default a diagonal is not potent), the necessary information to compute whether (i,d) is dominated and potent is available to us.

[2] If an in-neighbor does not exist (e.g., out of range), we consider it to be non-dominating.

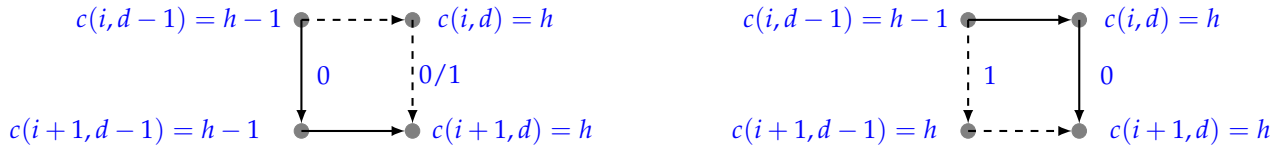

Figure 2. When (i,d) is dominated by $(i,d-1)$, diagonal $d-1$ can have a match at row $i+1$ (shown on left) or a mismatch (on right). Edges used by a shortest path to $(i+1,d)$ are drawn as solid, and other edges are dashed.

If (i,d) is dominated, the algorithm further checks whether each dominating in-neighbor is potent and whether it has a mismatch at the relevant row.Now c_A is updated: if (i,d) is determined to be potent and $(i+1,d)$ has a mismatch, then $c_A[d]$ is incremented by 1.

If (i,d) is not potent, then d is discarded from $\mathcal{D}_i$, and the algorithm proceeds to scan the next diagonal in $\mathcal{D}_i$. Otherwise, i.e., (i,d) is potent, the algorithm adds d to $\mathcal{D}_{i+1}$ (because it may be potent at row $i+1$), and then checks whether $(i+1,d)$ has a mismatch; if it has, then $d+1$ is added to $\mathcal{D}_i$ and $d-1$ added to $\mathcal{D}_{i+1}$ (because these vertices may be potent). Observe that processing $d \in \mathcal{D}_i$ may cause adding to $\mathcal{D}_i$ diagonal $d+1$, which obviously should be processed next; this means that $\mathcal{D}_i$ is scanned adaptively, but still in increasing order.

The correctness of this procedure is based on the next two lemmas, whose proofs appear in Appendix A.

Lemma III.6. *In this procedure, every potent vertex (i,d) is eventually inserted to $\mathcal{D}_i$. Moreover, at the end of iteration i (scanning $\mathcal{D}_i$), the diagonals in $\mathcal{D}_i$ represent exactly the potent vertices at row i.*

Lemma III.7. *Prior to iteration $i+1$ (or equivalently after iteration i), every $c_A[d']$ stores the value $c(i+1,d')$. More precisely, after processing diagonal $d \in \mathcal{D}_i$, each $c_A[d']$ has value $c(i,d')$ for $d' > d$ and $c(i+1,d')$ for $d' \le d$.*

B. Transitioning between Sampling Modes

Recall that the algorithm presented in Section II-B has asymmetric query complexity — it samples string x uniformly at rate $\frac{\log n}{t}$ but may query the entire string y, as it compares each sampled coordinate x_i against y_{i+d} for all possible $d \in [-t..t]$. In order to improve the query complexity to $\tilde{O}(\frac{n}{t}+t^3)$ and prove Theorem I.1, our algorithm will alternate between uniform sampling and contiguous (non-uniform) sampling. However, during the sampling mode, it is still prohibitive to compare each sampled coordinate x_i against y_{i+d} for all possible $d \in [-t..t]$. Instead, we would like to leverage the information given by $\mathcal{D}_i$. If $|\mathcal{D}_i| = 1$, it suffices to compare x_i against y_{i+d} only for the single $d \in \mathcal{D}_i$. And if $|\mathcal{D}_i| > 1$, then Lemma III.8 (part a) below guarantees that both x and y follow the same periodic pattern (coming into the current row), in which case we switch to uniform sampling, and merely "verify" that the periodicity continues (in the next rows) by comparing these samples to our pattern. Obviously, this verification is probabilistic, but by Claim II.4, with high probability it holds up to $O(t)$ Hamming errors (equivalently, character substitutions). When we see a sample that deviates from the periodicity in either x or y, we recompute $\mathcal{D}_i$ and start over, using Lemma III.8 (part b) to argue that almost all diagonals in $\mathcal{D}_i$ "must see" edit operations, which "count against" our budget of t edit operations.

Lemma III.8. *Let $x,y \in \Sigma^n$, let $i \in [n]$ and let $\mathcal{D} \subseteq [-t..t]$ be a set of diagonals of size $|D| > 1$. Define $g = \gcd\{d-d' : d > d' \in \mathcal{D}\}$, $p = x_{[i-g+1..i]}$ and $m = \max \mathcal{D} - \min \mathcal{D}$.*

(a) If

$$\forall d \in \mathcal{D}, \qquad x_{[i-2m+1..i]} = y_{[i-2m+1..i]+d} \tag{4}$$

then $x_{[i-2m+1..i]}$ and $y_{[i-2m+1+\min \mathcal{D}..i+\max \mathcal{D}]}$ are both periodic with the same period pattern p.

(b) Assume the conclusion of part (a) holds (i.e., $x_{[i-2m+1..i]}$ and $y_{[i-2m+1+\min \mathcal{D}..i+\max \mathcal{D}]}$ are both periodic with same period pattern p) but either $x_{i+1} \neq p_1$ or $y_{i+1+\max \mathcal{D}} \neq p_1$ (observe that $p_1 = p_{(i+1-2m+1) \bmod g}$). Then each diagonal in $\mathcal{D}$ except perhaps at most one, has a mismatch in at least one of the m rows $i+1,\ldots,i+m$.

The proof of this lemma appears in Section III-D, after the description of our algorithm. We remark that part (a) has been established in [31] and was used there for a different purpose.

C. Our Sublinear Algorithm

At a high level, the algorithm first picks a random set $S \subseteq [n]$, by including in S each row independently with probability $\frac{\log n}{t}$, and then proceeds in rounds, where each round processes one row. When a round processes row $i \in [n]$, we will say that it scans row i, as it will process some (but not all) of its vertices (i,d), always in increasing order of d. We shall also call it round i (although it need not be the i-th round).

The algorithm has two modes of scanning the rows, called *contiguous* and *sampling*. Roughly speaking, the

sampling mode scans only sampled rows, i.e., $i \in S$, and at each such round it reads only two characters x_i and y_{i+d} for a *single* active diagonal $d \in \mathcal{D}_i$. These two characters are compared not to each other (unless $|\mathcal{D}_i| = 1$), but rather to a pattern determined in previous rounds. The goal is to examine whether a certain periodicity in x and y is broken, in which case the algorithm switches to the contiguous mode. Observe that the sampling mode is rather lightweight – it scans rows at a rate of $1/t$ and performs minimal computation per row.

In contrast, the contiguous mode scans the rows one by one, and at each such round i, it compares x_i to y_{i+d} for every active diagonal $d \in \mathcal{D}_i$.

This mode is applied in bulks of at least $O(t)$ consecutive rows, until in a bulk of $O(t)$ the set of active diagonals does not change (which means that none of the active diagonals sees a mismatch along these lines). In the case that the set of active diagonals does not change we deduce a periodicity structure on the corresponding parts of x and y and the algorithm switches to the sampling mode.

We can now describe the algorithm in full detail. Let $i_1 < \ldots < i_{|S|}$ denote the rows selected into S. While scanning the rows, the algorithm maintains a counter, i for the current row initialized to $i = i_1$. as the algorithm is started in the sampling mode. The algorithm maintains also two lists of diagonals, $\mathcal{D}_i$ of active diagonals at the current round i and $\mathcal{D}_{\text{next}}$ that is constructed for the next round (which is either $i+1$ or the next row in S, depending on the mode), initialized to $\mathcal{D}_{i_1} = \{0\}$ and $\mathcal{D}_{\text{next}} = \emptyset$. During its execution, the algorithm computes and stores an array c_{A} to store an estimate of cost of every diagonal at the current row. It is initialized by setting $c_{\text{A}}[d] = |d|$ for every $d \in [-t..t]$.

The Contiguous Mode: In the contiguous mode, the algorithm scans the rows one by one. At each round i in this mode, the algorithm scans the active diagonals $d \in \mathcal{D}_i$ in increasing order, and each such d is processed as follows. If $c_{\text{A}}(d) > t - |d|$ then this diagonal d is ignored, i.e., its processing is concluded. Otherwise, the algorithm checks whether diagonal d is potent for row i, as defined in Definition III.1 but with respect to cost function c_{A} (instead of c), implemented by comparing $c_{\text{A}}[d]$ to the value of $c_{\text{A}}[d+1]$ and $c_{\text{A}}[d-1]$ in previous rows.[3] If d is indeed potent, then it is added to $\mathcal{D}_{\text{next}}$, and if in addition, $x_{i+1} \neq y_{i+d+1}$, then $d+1$ is added to $\mathcal{D}_i$ and $d-1$ is added to $\mathcal{D}_{\text{next}}$ and $c_{\text{A}}[d]$ is incremented by 1. This completes the processing of $d \in \mathcal{D}_i$.

[3]This check is implemented similarly to Section III-A, by maintaining the costs computed in the previous two rows and the list of potent diagonals at those rows.

When the algorithm finishes scanning $\mathcal{D}_i$, it has to decide about the next round. If along the last $2(\max \mathcal{D}_i - \min \mathcal{D}_i)$ rows there exists a row i', in which a mismatch was found at some diagonal $d \in \mathcal{D}_{i'}$, then the algorithm increments i by 1, sets $\mathcal{D}_i = \mathcal{D}_{\text{next}}$ and $\mathcal{D}_{\text{next}} = \emptyset$, and proceeds to the next round (for this new value of i), staying in the contiguous mode.

Otherwise (no mismatch was found along these rows), the algorithm prepares to switch to the sampling mode by computing three variables:

$$g = \gcd\{d - d'\}_{d \neq d' \in \mathcal{D}_i}$$
$$i_{\text{pat}} = i - 2(\max \mathcal{D}_i - \min \mathcal{D}_i) + 1,$$
$$p = x_{[i_{\text{pat}}..i_{\text{pat}}+g-1]},$$

and then increases i to the next row in S (smallest one after the current i), sets $\mathcal{D}_i = \mathcal{D}_{\text{next}}$ and $\mathcal{D}_{\text{next}} = \emptyset$, and proceeds to the next round (for this new value of i) but in the sampling mode.

The Sampling Mode: In the sampling mode, the algorithm processes only sampled rows $i \in S$. The sampling mode performs one of two checks, either *periodicity check* or *shift check*, depending on whether $|\mathcal{D}_i| > 1$ or not.

(i) Periodicity check: This check is applied only if $|\mathcal{D}_i| > 1$. The algorithm first checks whether both x_{i+1} and $y_{i+\max \mathcal{D}_i+1}$ are equal to $p_{(i-i_{\text{pat}}+1) \bmod g}$. If both match (are equal), the algorithm finds the next row i_{next} in S (smallest one after the current i), sets $\mathcal{D}_{i_{\text{next}}} = \mathcal{D}_i$, increases i to i_{next}, and proceeds to the next round (for this new value of i), still in the sampling mode (to perform a periodicity check because again $|\mathcal{D}_i| > 1$).

Otherwise (at least one of the two comparisons fails), the algorithm employs binary search to detect a row $j \in [i_{\text{pat}} + 2(\max \mathcal{D}_i - \min \mathcal{D}_i)..i]$ with a "period transition", defined as j satisfying the two conditions:

1) for all $j' \in [j - 2(\max \mathcal{D}_i - \min \mathcal{D}_i)..j]$, we have $x_{j'} = y_{j'+\max \mathcal{D}_i} = p_{(j'-i_{\text{pat}}) \bmod g}$; and
2) either x_{j+1} or $y_{j+\max \mathcal{D}_i+1}$ is not equal to $p_{(j+1-i_{\text{pat}}) \bmod g}$.

We later prove that such a j must exist. The algorithm then finds all the diagonals $d \in \mathcal{D}_i$ that have a mismatch in at least one row in the range $[j..j + \max \mathcal{D}_i - \min \mathcal{D}_i]$. Lemma III.8 shows that this event (at least one mismatch) must occur for all the diagonals in $\mathcal{D}_i$ except perhaps one diagonal, which we denote by d^* (if exists). Now the algorithm sets

$$\forall d \in \mathcal{D}_i, d \neq d^*, \qquad c_{\text{A}}[d] = c_{\text{A}}[d] + 1; \tag{5}$$

and adds d, $d+1$, and $d-1$ to $\mathcal{D}_{\text{next}}$.

If d^* exists, the algorithm further samples a new set $S^* \subseteq [i_{\text{pat}}..i]$ at rate $\frac{\log n}{t}$, and for each row $j' \in S^*$ it compares $x_{j'}$ to $y_{j'+d^*}$. If no mismatch is found in

S^*, then the algorithm adds d^* to $\mathcal{D}_{\text{next}}$. Otherwise (a mismatch is found), it sets $c_A[d^*] = c_A[d^*] + 1$ and adds d^*, $d^* + 1$, and $d^* - 1$ to $\mathcal{D}_{\text{next}}$.

Finally, the algorithm increments i by 1, sets $\mathcal{D}_i = \mathcal{D}_{\text{next}}$ and $\mathcal{D}_{\text{next}} = \emptyset$, and proceeds to the next round (for this new value of i) but in the contiguous mode.

(ii) Shift check: This check is applied only if $|\mathcal{D}_i| = 1$; let d be the unique diagonal in $\mathcal{D}_i$. The algorithm compares x_{i+1} and y_{i+1+d}. If they match (are equal), the algorithm increases i to the next row in S (smallest one after the current i), sets $\mathcal{D}_i = \{d\}$ and proceeds to the next round (for this new value of i), still in the sampling mode (to perform a shift check because again $|\mathcal{D}_i| = 1$).

Otherwise (they do not match), the algorithm sets $c_A[d] = c_A[d] + 1$ and adds d, $d + 1$, and $d - 1$ to $\mathcal{D}_{\text{next}}$. The algorithm then increments i by 1, sets $\mathcal{D}_i = \mathcal{D}_{\text{next}}$ and $\mathcal{D}_{\text{next}} = \emptyset$, and proceeds to the next round (for this new value of i) but in the contiguous mode.

Stopping Condition: The algorithm halts and outputs `far` if at any point the value of $c_A[0]$ reaches $t + 1$. If the algorithm completes processing the rows (the last one is in S or in $[n]$, depending on the mode), and still $c_A[0] \le t$, then the algorithm halts and outputs `close`.

D. Analysis

Let us start by proving lemma III.8. For convenience, let us restate it:

Lemma (III.8). *Let $x, y \in \Sigma^n$, let $i \in [n]$ and let $\mathcal{D} \subseteq [-t..t]$ be a set of diagonals of size $|D| > 1$. Define $g = \gcd\{d - d' : d > d' \in \mathcal{D}\}$, $p = x_{[i-g+1..i]}$ and $m = \max \mathcal{D} - \min \mathcal{D}$.*

(a) If

$$\forall d \in \mathcal{D}, \qquad x_{[i-2m+1..i]} = y_{[i-2m+1..i]+d} \tag{6}$$

then $x_{[i-2m+1..i]}$ and $y_{[i-2m+1+\min \mathcal{D}..i+\max \mathcal{D}]}$ are both periodic with the same period pattern p.

(b) Assume the conclusion of part (a) holds (i.e., $x_{[i-2m+1..i]}$ and $y_{[i-2m+1+\min \mathcal{D}..i+\max \mathcal{D}]}$ are both periodic with same period pattern p) but either $x_{i+1} \neq p_1$ or $y_{i+1+\max \mathcal{D}} \neq p_1$ (observe that $p_1 = p_{(i+1-2m+1) \bmod g}$). Then each diagonal in $\mathcal{D}$ except perhaps at most one, has a mismatch in at least one of the m rows $i + 1, \ldots, i + m$.

Proof: To prove the first part, consider two diagonals $d_1 < d_2$ in $\mathcal{D}$. We can see that $x_{[i-2m+1..i]}$ has period length $d_2 - d_1$, by using (6) twice (for all relevant positions j)

$$\forall j \in [i - 2m + 1..i - (d_2 - d_1)], \qquad x_j = y_{j+d_2} = x_{j+(d_2-d_1)}.$$

Now use the fact that if a string s is periodic with two period lengths $l \neq l'$, then it is also periodic with period length $\gcd\{l, l'\}$.[4] It follows that x has period length $g = \gcd\{d - d' : d > d' \in \mathcal{D}\}$, hence its period pattern is $p = x_{\{i-g+1,\ldots,i\}}$.

By applying a similar argument to y, we obtain that it too has period length g, and its period pattern is $y_{[i+\max \mathcal{D}-g+1..i+\max \mathcal{D}]}$. Moreover, by applying (6) to $d = \max \mathcal{D}$ we see that the period patterns of x and of y are equal

$$p = x_{[i-g+1..i]} = y_{[i+\max \mathcal{D}-g+1..i+\max \mathcal{D}]}.$$

Let us now prove the second part. Assume first that $x_{i+1} \neq p_1$. Then for every diagonal $d \in \mathcal{D} \setminus \{\max \mathcal{D}\}$ we have $y_{i+d+1} = y_{i+\min \mathcal{D}+1} = p_1$ (because $d - \min \mathcal{D}$ is a multiple of the period g and all these positions are inside the periodic part of y), and we see that diagonal d has a mismatch at row $i + 1$.

Next, assume that $x_{i+1} = p_1 \neq y_{i+\max \mathcal{D}+1}$. Let $i_x \ge i + 1$ be the smallest row "deviating" from the pattern p, i.e., $x_{i_x} \neq p_{(i_x-i) \bmod g}$, letting $i_x = \infty$ if no such row exists. Our assumption implies that actually $i_x > i + 1$. We proceed by a case analysis.

If $i_x \ge i + 1 + (\max \mathcal{D} - \min \mathcal{D})$, then we can show that every diagonal $d \in \mathcal{D} \setminus \{\min \mathcal{D}\}$ has a mismatch at row $i + 1 + (\max \mathcal{D} - d)$. Indeed, x is periodic up to that row because $i + 1 + (\max \mathcal{D} - d) < i + 1 + (\max \mathcal{D} - \min \mathcal{D}) \le i_x$, and thus

$$x_{i+1+\max \mathcal{D}-d} = x_{i+1} = p_1 \neq y_{i+1+\max \mathcal{D}}.$$

And since $0 \le \max \mathcal{D} - d < m$, the row with the mismatch is indeed in the claimed range $i + 1, \ldots, i + m$.

Otherwise, we have $i + 1 < i_x < i + \max \mathcal{D} - \min \mathcal{D} + 1$. For every $d \in \mathcal{D}$ satisfying $d > d' := i + 1 + \max \mathcal{D} - i_x$ there is a mismatch at row $i + 1 + \max \mathcal{D} - d$ as x is still periodic at that position because $i + 1 + \max \mathcal{D} - d < i + 1 + \max \mathcal{D} - d' = i_x$, and thus

$$x_{i+1+\max \mathcal{D}-d} = x_{i+1} = p_1 \neq y_{i+1+\max \mathcal{D}}.$$

For every $d \in \mathcal{D}$ satisfying $d < d'$, we have a mismatch at row i_x, because x is not periodic at i_x and thus $x_{i_x} \neq p_{(i_x-i) \bmod g}$, while y is still periodic at position $i_x + d < i_x + d' = i + 1 + \max \mathcal{D}$ and thus

$$y_{i_x+d} = p_{(i_x+d-(i+\min \mathcal{D})) \bmod g} = p_{(i_x-i) \bmod g} \neq x_{i_x}.$$

In both cases, $d > d'$ and $d < d'$, the mismatched row is indeed in the claimed range $i + 1, \ldots, i + m$, and together the two cases include all but at most one diagonal in $\mathcal{D}$.

■

We show next that when the algorithm is in the sampling mode and executes a binary search to find

[4] To see this, use Bézout's identity to write $\gcd\{l, l'\} = tl + t'l'$ for integers t, t', then show that $s_j = s_{j+tl+t'l'}$ by a sequence of $|t| + |t'|$ equalities, ordered so as to stay inside s.

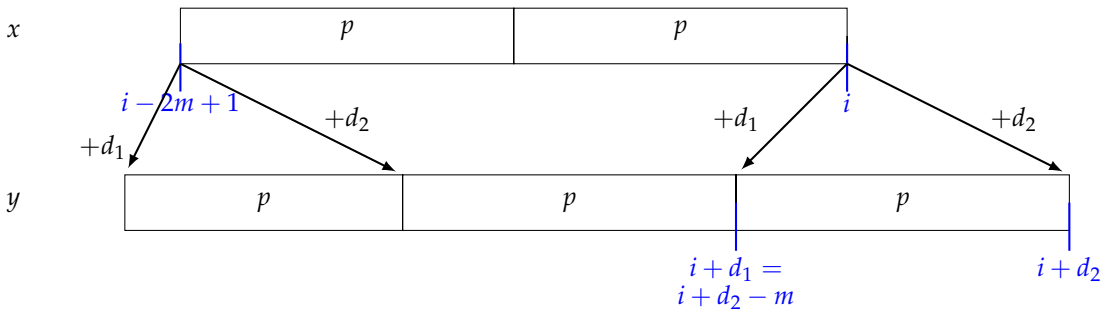

Figure 3. Matching the strings along different diagonals d_1, d_2.

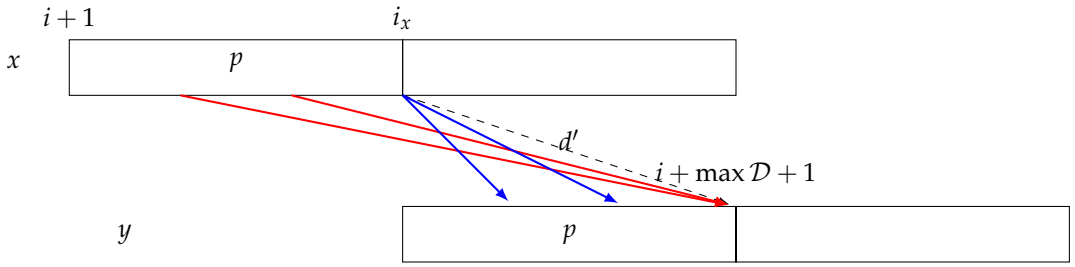

Figure 4. Red lines represent mismatches on diagonals $d > d'$, and blue lines represent mismatches on diagonals $d < d'$

a period transition, then it always succeeds.

Claim III.9. *Suppose at round i the algorithm stays at the sampling mode performing a periodicity check and it detects a period violation, that is either $x_{i+1} \neq p_{(i+1-i_{\text{pat}}) \bmod g}$ or $y_{i+\max\mathcal{D}+1} \neq p_{(i+1-i_{\text{pat}}) \bmod g}$, then:*

1) *There exists a row $j \in [i_{pat} + 2(\max\mathcal{D} - \min\mathcal{D})..i]$ with a period transition, that is,*

$$\forall j' \in [j - 2(\max\mathcal{D} - \min\mathcal{D})..j],$$
$$x_{j'} = y_{j'+\max\mathcal{D}} = p_{(j'-i_{pat}) \bmod g} \text{ and}$$
$$\text{either } x_{j+1} \neq p_{(j+1-i_{pat}) \bmod g}$$
$$\text{or } y_{j+\max\mathcal{D}+1} \neq p_{(j+1-i_{pat}) \bmod g}$$

2) *Let j be the row from the previous item, then for all $d \in \mathcal{D}$ but perhaps at most one diagonal d^*, there exists a row $j' \in [j..j + \max\mathcal{D} - \min\mathcal{D}]$ such that $x_{j'} \neq y_{j'+d}$.*

Proof: We first prove part (1). Since the algorithm enters the sampling mode only after on the last $2(\max\mathcal{D} - \min\mathcal{D})$ (where $\mathcal{D}$ denotes the set of potent diagonals when the algorithm enters the sampling mode) no mismatch has occurred for each d in the potent diagonals, then our set $\mathcal{D}$ satisfies the condition of Lemma III.8 at row i_{pat}. Note that by Lemma III.4, the set $\mathcal{D}$ did not change. By the lemma we get $x_{[i_{\text{pat}}..i_{\text{pat}}+2(\max\mathcal{D}-\min\mathcal{D})]} = p \circ \cdots \circ p$ and $y_{[i_{\text{pat}}..i_{\text{pat}}+2(\max\mathcal{D}-\min\mathcal{D})]+\max\mathcal{D}} = p \circ \cdots \circ p$. Since at row $i+1$ at least one of the values $x_{i+1}, y_{i+\max\mathcal{D}+1}$ does not match the corresponding character in p, then between i and i_{pat}, there exists an index j on which we have a period transition.

Part (2) follows directly by the second item of Lemma III.8. ∎

Correctness Analysis: We now analyze the correctness of the algorithm in the two cases, `close` that is $\Delta_e(x,y) \leq t/2$ and `far` that is $\Delta_e(x,y) > 13t^2$, more specifically we prove the following lemmas:

Lemma III.10. *Let $x, y \in \{0,1\}^n$. If $\Delta_e(x,y) \leq t/2$ then with probability 1 the algorithm outputs `close`.*

Lemma III.11. *If $\Delta_e(x,y) > 13t^2$ then with probability at least $2/3$ the algorithm outputs `far`.*

E. Proof of Close Case $\Delta_e(x,y) \leq t/2$

Proof of Lemma III.10: Let x, y be such that $\Delta_e(x,y) \leq t/2$. We show that with probability 1 the algorithm outputs `close`. For this sake, we build a grid graph $G' = (V', E')$ and define a new cost function $c' : E' \to \mathbb{N} \cup \{0\}$. We show that there exists a path connecting the sink and the source in G' whose cost c' is at most $t/2$. The set V' will include all the vertices that the algorithm scans. Then, we define a new cost function, denoted c_{ALG} on the E'. We claim that the costs assigned by the algorithm are consistent with c_{ALG}. Finally, we connect the costs c' and c_{ALG}.

Graph Construction.: The graph is built as follows: Recall S denote the sampled rows by the algorithm.

Let S' be the set of rows i such that either $i \in S$ or the algorithm scans the row i during the contiguous mode or i is a row that the algorithm scans after finding a period transition during binary search or during a shift check. We set

$$V' = S' \times [-t..t] \cup \{(0,0),(n,0)\}.$$

Let us describe the cost c' for each edge in G'. Let $i \in S'$. Let us first deal with the boundaries. We connect $(0,0)$ to $(i_1,0)$ where i_1 is the smallest row in S' by an edge of cost 0. Let $i_{|S'|}$ be the largest row in S', we connect each vertex $(i_{|S'|},d)$, into $(n,0)$ by an edge of cost $|d|$.

Let $i \in S'$ which is not the last row in S' and let i_{next} be the next row in S'. Each vertex (i,d) is connected to:

$$(i,d) \to (i,d+1),$$
$$(i,d) \to (i_{\text{next}},d-1) \text{ and}$$
$$(i,d) \to (i_{\text{next}},d).$$

The first two edges are associated with cost 1. The cost of $(i,d) \to (i_{\text{next}},d)$ is defined as: $\mathbb{1}_{\{x_{i+1} \neq y_{i+d+1}\}}$.

Corollary III.12. *For every path τ in $G_{x,y}$ from $(0,0)$ to $(n,0)$, there exists a corresponding path τ_S in G' from $(0,0)$ to $(n,0)$ such that $c'(\tau_S) \leq c(\tau)$.*

This follows directly by Claim A.1. Next we would like to connect the costs c_{A} and c'. For this sake we first define a cost function c_{ALG} on the edge set E' and then claim it is consistent with the costs assigned by the algorithm.

Defining c_{ALG}: We define a cost function c_{ALG} on the edges of G' as follows.

Let $i \in S'$. Let us first deal with the boundaries. We connect $(0,0)$ to $(i_1,0)$ where i_1 is the smallest row in S' by an edge of cost 0. Let $i_{|S'|}$ be the largest row in S', we connect each vertex $(i_{|S'|},d)$, into $(n,0)$ by an edge of cost $|d|$.

Let $i \in S'$ which is not the last row in S' and let i_{next} be the next row in S'. The edges $(i,d) \to (i,d+1),(i_{\text{next}},d+1) \to (i,d)$ are associated with cost 1. The cost of $(i,d) \to (i_{\text{next}},d)$ is defined as follows:

Case 1: i is a contiguous round: Then its cost is $\mathbb{1}_{x_{i+1} \neq y_{i+d+1}}$.

Case 2: i is a sampling round: If the consistency check passes then the cost is 0. Otherwise, recall that the algorithm first detects a row j on which there is a period transition and then finds all the diagonals $d \in \mathcal{D}$ that have a mismatch in at least one row in the range $[j..j+\max\mathcal{D}-\min\mathcal{D}]$. For each of these diagonals we assign $c_{\text{ALG}}((i,d) \to (i_{\text{next}},d)) = 1$. Recall that if d does not have a mismatch in either of these rows in this range, then the algorithm samples another set S^* and checks whether d has a mismatch in S^*. If the later test passes then the cost is 0 and otherwise it is 1. For the rest of the diagonals ($d \notin \mathcal{D}$) we set $c_{\text{ALG}}((i,d) \to (i_{\text{next}},d)) = 0$.

We next connect c_{A} and c_{ALG}, for this we extend c_{ALG} to V' by setting $c_{\text{ALG}}(v)$ as the shortest path cost (with respect to c_{ALG}) connecting $(0,0)$ and (i,d). We next prove:

Lemma III.13. *Let i be a row scanned by the algorithm, prior to iteration i_{next} (or equivalently after iteration i), every $c_A[d']$ stores the value $c_{ALG}(i_{\text{next}},d')$. More precisely, after processing diagonal $d \in \mathcal{D}_i$, each $c_A[d']$ has value $c_{ALG}(i,d')$ for $d' > d$ and $c_{ALG}(i_{\text{next}},d')$ for $d' \leq d$.*

Proof: To prove the lemma we use the following definition: We say that diagonal d is c_{ALG}*-potent* at row i if it satisfies the two requirements:

- if (i,d) is dominated by $(i,d-1)$ (with respect to c_{ALG}), then we require that diagonal $d-1$ is potent at row i and $c_{\text{ALG}}((i,d-1) \to (i_{\text{next}},d-1)) = 1$; and
- if (i,d) is dominated by $(i_{\text{last}}-1,d+1)$ then we require that diagonal $d+1$ is potent at row i_{last} and $c_{\text{ALG}}((i_{\text{last}},d+1) \to (i,d+1)) = 1$, where i_{last} is the largest row in S' smaller than i.

We prove only the first assertion, as the second one is an immediate consequence of it. The proof is by induction on the grid vertices $(i,d) \in V'$ in lexicographic order (i.e., their row is the primary key and their diagonal is secondary). The base case is the time before processing vertex $(0,0)$; at this time, c_{A} stores its initial values, i.e., $c_{\text{A}}[d] = |d|$, which is equal to $c(0,d)$ for all $d \geq 0$. For $d < 0$, the base case is the time before processing $(-d,d)$, because we should only consider vertices reachable from $(0,0)$; at this time, $c_{\text{A}}[d] = |d|$ is still the initialized value and it is equal to $c(d,-d) = -d$.

For the inductive step, we need to show $c_{\text{A}}[d]$ is updated according to c_{ALG}. But using the induction hypothesis, we only need to show $c_{\text{A}}[d]$ is updated from $c_{\text{ALG}}(i,d)$ to $c_{\text{ALG}}(i_{\text{next}},d)$. To this end, suppose first that vertex (i,d) is non-potent. Then by Lemma III.3 we have $c_{\text{ALG}}(i+1,d) = c_{\text{ALG}}(i,d)$, (regardless of the cost $c_{\text{ALG}}((i,d) \to (i_{\text{next}},d))$) and the algorithm indeed does not modify $c_{\text{A}}[d]$. Suppose next (i,d) is potent: Observe that the algorithm increases $c_{\text{A}}[d]$ by 1 only if $c_{\text{ALG}}((i,d) \to (i_{\text{next}},d)) = 1$, in which case by Lemma III.4 we have: $c_{\text{ALG}}(i_{\text{next}},d) = c_{\text{ALG}}(i,d)+1$, so by the induction hypothesis $c_{\text{A}}[d]$ is incremented to the correct value. On the other hand, the algorithm does not change the value of $c_{\text{A}}[d]$ by 1 if $c_{\text{ALG}}((i,d) \to (i_{\text{next}},d)) = 0$, in which case by Lemma III.4 we have $c_{\text{ALG}}(i_{\text{next}},d) = c_{\text{ALG}}(i,d)$, and again by the induction hypothesis $c_{\text{A}}[d]$ matches the correct value. ■

By abusing notations, we define a cost function c' on the vertices of V' by setting $c'(v')$ as the cost of shortest path connecting $(0,0)$ and v', for each $v' \in V'$.

Finally we connect the cost c' and c_{ALG}, namely: Let τ_S be a shortest path in G' with respect to the cost function c' connecting $(0,0)$ and $(n,0)$. We conclude the proof by claiming $c_{\mathrm{ALG}}(\tau_S) \leq 2c'(\tau_S)$.

Claim III.14. $c_{ALG}(\tau_S) \leq 2c'(\tau_S)$.

Proof: The proof proceeds by induction on the rows in S'. In particular, we show that for each row $i \in S'$ and each diagonal d traversed by τ_S at row i, we have: $c_{\mathrm{ALG}}(i,d) \leq 2c'(i,d)$. The base case is $i = 0$, the only diagonal τ_S traverses at row 0 is 0 for the cost is 0 with respect to c_{ALG}, c'.

The induction step: let $i \in S'$, let $(i_{\mathrm{last}}, d_{\mathrm{last}})$ be the last vertex on τ_S before moving to row i, and let (i,d) be the first diagonal that τ_S traverses at row i (observe that $d_{\mathrm{last}} \in \{d+1, d\}$). We first prove that $c_{\mathrm{ALG}}(i,d) \leq 2c'(i,d)$. Then we prove it for the rest of the diagonals traversed by τ_S at row i.

Case 0- $(d_{\mathrm{last}} \neq d)$: In this case, $c'(i,d) - c'(i_{\mathrm{last}}, d_{\mathrm{last}}) = 1$. Since for each neighboring diagonal the difference in c_{ALG} cost is at most 1, then $c_{\mathrm{ALG}}(i,d) \leq c_{\mathrm{ALG}}(i_{\mathrm{last}}, d_{\mathrm{last}}) + 1$, the claim follows.

Case 1- i **is a contiguous round**:

Case 1.2.1- $(d_{\mathrm{last}} = d)$ **and** $(i_{\mathbf{last}}, d_{\mathrm{last}})$ **is** $c_{\mathbf{ALG}}$**-potent**: In this case $c_{\mathrm{ALG}}(i,d) - c_{\mathrm{ALG}}(i_{\mathrm{last}}, d_{\mathrm{last}}) = c'(i,d) - c'(i_{\mathrm{last}}, d_{\mathrm{last}}) = \mathbb{1}_{x_{i_{\mathrm{last}}+1} \neq y_{i_{\mathrm{last}}+d_{\mathrm{last}}+1}}$, the claim follows.

Case 1.2.2-$(d_{\mathrm{last}} = d)$ **and** $(i_{\mathbf{last}}, d_{\mathrm{last}})$ **is not** $c_{\mathbf{ALG}}$**-potent**: In that case from Lemma III.3, $c_{\mathrm{ALG}}(i_{\mathrm{last}}, d_{\mathrm{last}}) = c_{\mathrm{ALG}}(i,d)$. In c' the difference between the costs may be either 0 or 1, the claim follows.

Case 2- i **is a sampling round**:

Case 2.1- $(d_{\mathrm{last}} = d)$**,** i **is a sampling round and the periodicity check passes at row** i**:** Observe that the cost c_{ALG} is not incremented over the last row in either of the diagonals at row i. While in c' it may be increased, the claim follows.

Case 2.2- $(d_{\mathrm{last}} = d)$**,** i **is a sampling round and the periodicity check fails at row** i**:** If $(i_{\mathrm{last}}, d_{\mathrm{last}})$ is non-potent then the proof follows by the argument used in case 1.2.2. Let us prove the claim for the case $(i_{\mathrm{last}}, d_{\mathrm{last}})$ being potent. In this case the algorithm performs a binary search to detect a period transition at row j. The algorithm sets $c_{\mathrm{ALG}}(i,d) = c_{\mathrm{ALG}}(i_{\mathrm{last}}, d) + 1$ if diagonal d has a mismatch at some row $j' \in [j..j + \max \mathcal{D} - \min \mathcal{D}]$. For the special diagonal d^* that has no mismatch at none of the rows j', the algorithm samples another set S' and increments the cost if it finds a mismatch along one of the rows in S'.

Let us first analyze the diagonals d that have a mismatch at some row j': Recall that i_{pat} is the first row on which the algorithm switched to sampling mode. First observe that $\ell \in [i_{\mathrm{pat}}..i)$ we have: $c_{\mathrm{ALG}}(\ell, d) = c_{\mathrm{ALG}}(i_{\mathrm{last}}, d_{\mathrm{last}})$ since the costs were not incremented.

If τ_S passes through (j', d), then $c'(j', d)$ was incremented by 1, while in $c_{\mathrm{ALG}}(j', d)$, it was not incremented at row j' and only at row i. So the contribution of the mismatch with respect to both cost functions is the same. If τ_S was not traversing through (j', d): At a later row j'', τ_S transits to diagonal d and pays a cost 1 for the transition. In c_{ALG} we pay for this transition twice: once at the transition row, and second time at row i (note that each such a transition is counted only once). The claim follows. The analysis of the special diagonal d^* follows by the same argument.

If τ_S traverses other diagonals at row i then with respect to c' it pays a cost 1 on each diagonal transition. However, on c_{ALG} we may pay either 0 or 1 cost. Therefore, the claim holds for all vertices at τ_S touching row i. ■

We conclude the proof by claiming that the algorithm outputs `close`. By Corollary III.12, in G' there exists a shortest path to the sink of cost $\leq t/2$ with respect to the cost c'. Therefore, by Claim III.14 there exists a shortest path to the sink of cost $\leq t$ with respect to the cost c_{ALG}. By Lemma III.13 after processing the last row of S' the cost of $c_{\mathrm{A}}[0]$ equals to the cost of the shortest path to $(i_{|S'|}, 0)$ is at most t. Therefore, by monotonicity the cost along $c_{\mathrm{A}}[0]$ does not exceed t and the algorithm outputs `close`. This completes the proof of Lemma III.10. ■

F. Proof of Far Case $\Delta_e(x,y) > 13t^2$

For the purposes of analysis, we think of the algorithm as first independently sampling two sets S_1, S_2, where each set $S_j, j = 1,2$ is drawn such that each row is inserted into S_j independently with probability $\frac{\log n}{t}$. While staying at the sampling mode, the algorithm uses S_1 for *periodicity check* and the set S_2 for shift check.

To conclude the proof we rely on the following claim, which is a variant of Claim II.4. Roughly speaking, we define bad events to capture scenarios in which the algorithm might fail, either that on a sampling round on a single diagonal we skip too many mismatches, or if we have one diagonal we skip too many period violations.

Claim III.15. *Let* $x, y \in \Sigma^n$ *let* $i \in [n]$*, and let* $S_j \subseteq [n]$*,* $j = 1,2$ *be sets drawn by including each row in* S *independently with probability* $\frac{\log n}{t}$*.*

Let $g \le 2t+1$, $p \in \{0,1\}^g$ and $d \in [-t..t]$. Let:

$$i_{d,i} = \min_{r \ge i} \Delta_H(x_{[i..r]}, y_{[i..r]+d}) > 4t,$$
$$I_{d,i} = \left\{ j \in [i..i_{d,i}] : \ x_j \neq y_{j+d} \right\};$$
$$i_{x,p,i} = \min_{r \ge i} \Delta_H(x_{[i..r]}, p^*) > 4t,$$
$$I_{x,p,i} = \left\{ j \in [i..i_{x,p,i}] : \ x_j \neq p_{j-i \bmod p} \right\};$$
$$i_{y,p,i} = \min_{r \ge i} \Delta_H(y_{[i..r]}, p^*) > 4t,$$
$$I_{y,p,i} = \left\{ j \in [i..i_{y,p,i}] : \ y_j \neq p_{j-i \bmod p} \right\};$$

where $\Delta_H(x_{[i..r]}, p^*)$ is the Hamming distance between $x_{[i..r]}$ and the corresponding periodic string with period pattern p of length $r-i$. If in either of the above definitions no such r exists, we set $i_{d,i}, i_{x,p,i}, i_{y,p,i} = \infty$.

We define an event $B_S(i,d)$ as the event where $i_{d,i} \neq \infty$ and $S \cap I_{d,i} = \emptyset$, similarly we define $B_S(i,x,p)$ as the event where $i_{x,p,i} \neq \infty$ and $S \cap I_{x,p,i} = \emptyset$ (and $B_S(i,y,p)$ is defined similarly). Then:

$$\Pr[B_S(i,d)] < \frac{1}{9n(2t+1)}$$

and similarly $\Pr[B_S(i,x,p)], \Pr[B_S(i,y,p)]$ are at most $\frac{1}{9n(2t+1)}$.

The proof of Claim III.15 follows immediately by Chernoff bound. Let us conclude the proof using the claim. First by union bound on the set of all possible rows and diagonals we get that except with probability $n(2t+1)\frac{1}{3n(2t+1)} \le \frac{1}{9}$, none of the events $B_{S_1}(i,d)$ happens. Moreover, using a union bound on the value of all possible $i \ge 2t+1$ and $j \in [0..2t+1]$ we have that that except with probability $2(n(2t+1)\frac{1}{9n(2t+1)}) \le \frac{2}{9}$, none of the events $B_{S_2}(i, x_{[i-j..i]}, x), B_{S_2}(i, y_{[i-j..i]}, y)$ happens. So overall none of the events mentioned above happen with probability at least $2/3$. We conclude the proof by showing that in such a case, the algorithm outputs `far` with probability 1.

Assume for sake of contradiction that none of the events described above happen and still the algorithm outputs `close`. We may view the algorithm as computing the shortest path τ on the grid graph $G_{S'}$, with respect to the cost function c_{ALG} defined on the proof of Claim III.10.

The path τ divides the set of rows processed by the algorithm into intervals $I'_0, \ldots, I'_k \subseteq S$ such that (i) I'_j and I'_{j+1}, $j \in [0, k-1]$ can intersect on at most single row, and (ii) for each interval I'_ℓ, τ traverses along vertices on diagonal d_ℓ while paying cost 0 (so moving between intervals correspond to either substitutions or diagonal transitions implying insertions and deletions).

Observe that $k \le t - |d_k|$ and $\max I_k$ is the largest round processed by the algorithm.

Let us define another set of intersecting intervals $I_0, \ldots, I_k \subseteq [n]$ as follows: $I_0 = [0..\max I'_0]$, $I_\ell = I'_\ell$ and $I_k = [\min I'_k..n]$.

Consider the path τ in $G_{x,y}$ that on rows in I_ℓ follows diagonal d_ℓ, while paying the cost along all diagonal edges, and then finally it traverses to $(n,0)$. Let us denote by τ_{I_ℓ} the cost of τ confined to the rows in I_ℓ, which equals: $\Delta_H(x_{I_\ell}, y_{I_\ell + d_\ell})$.

Claim III.16. *For all $\ell \in [k]$ we have:*

$$c(\tau_{I_\ell}) = \Delta_H(x_{I_\ell}, y_{I_\ell + d_\ell}) < 12t.$$

Proof: Let us consider an interval I_ℓ: Observe that once the diagonal d_ℓ is part of the diagonal on which the algorithm detects a mismatch followed by period transition (end of a sampling mode), then the current interval is over. Hence we can break each interval I_ℓ into its prefix I^p_ℓ that contains the rows on which the algorithm performs a shift check. And its suffix I^s_ℓ that contains the rows on which the algorithm performs periodicity check and finally encounters a mismatch on d_ℓ.

We conclude the claim by proving that $\Delta_H(x_{I^p_\ell}, y_{I^p_\ell + d_\ell}) < 4t$ and $\Delta_H(x_{I^s_\ell}, y_{I^s_\ell + d_\ell}) < 8t$.

Observe that since we assume that the event $B_{S_1}(\min I_\ell, d_\ell)$ did not happen then by definition $\Delta_H(x_{I^p_\ell}, y_{I^p_\ell + d_\ell}) < 4t$. Next since $B_{S_2}(i_\ell, x_{[i_\ell - g..i-1]}, x)$ and $B_{S_2}(i_\ell + \max \mathcal{D}, x_{[i_\ell - g..i-1]}, y)$ did not happen then we get: $\Delta_H(x_{[i_\ell + 1..\max I_\ell]}, p^*) < 4t$ and $\Delta_H(y_{[i_\ell + 1..\max I_\ell]} + d_\ell, p^*) < 4t$ where p^* as usual denotes repeated concatenation of p. By triangle inequality we get that $\Delta_H(x_{[i_\ell+1..\max I_\ell]}, y_{[i_\ell+1..\max I_\ell]+d_\ell}) < 8t$, as claimed. ■

In total this implies that the cost of τ is at most $k \times 12t + k + t - |d_k| \le 12t^2 + t \le 13t^2$, contradicting the fact that $\Delta_e(x,y) > 13t^2$.

G. Query Complexity Analysis

Claim III.17. *The query complexity of the algorithm presented in Section III-C is bounded by $\tilde{O}(\frac{n}{t} + t^3)$.*

Proof: We bound the number of queries by showing that while staying at the contiguous mode the algorithm queries the input in $O(t^3 \log n)$ locations, and while staying on the sampling mode, it queries $O(\frac{n \log n}{t})$ locations with high probability.

Observe that the algorithm enters into the contiguous mode whenever it encounters a mismatch on at least one of the potent diagonals. Also observe that after $2(2t+1)$ rounds on which the algorithm stays at the contiguous mode, either there exists a mismatch

on at least one of the potent diagonals or the algorithm shifts to the sampling mode. Therefore, for each $2(2t+1)$ consecutive rows, spent on the contiguous mode we can match a mismatch on at least one of the potent diagonals. Also observe that (Lemma III.4) each diagonal can have at most $O(t)$ mismatches while it is potent. Therefore, the number of $2(2t+1)$ consecutive rounds on which the algorithm stays at the contiguous mode is bounded by $O(t^2)$.

Notice also that on each block I of $2(2t+1)$ consecutive rounds on which the algorithm stays at the contiguous mode, it samples only $O(t)$ characters from both x and y (as it needs to compare from x_I against values in $y_{\{\min I-t,\ldots,\max I+t\}}$). Therefore, in total the number of queries used while staying at the contiguous mode is bounded by $O(t^3)$.

Regrading the sampling mode, at each sampling round, if both periodicity checks pass then the algorithm makes only 1 queries into both strings x, y. So such rounds can contribute at most $|S|$ queries (which is bounded by $O(\frac{n\log n}{t})$ with high probability). Otherwise, it makes $O(t\log n)$ additional queries and by Claim III.9 we are guaranteed to find at least one mismatch on one of the potent diagonals. Therefore, by the same argument used for the contiguous search the number of queries made by during the periodicity check of the sampling mode is bounded by $O(t^3\log n)$. The number of rows sampled by the shift mode is bounded by $O(\frac{n\log n}{t})$ with high probability. Hence, the claim follows. ∎

H. Time Complexity Analysis

Claim III.18. *The time complexity of the algorithm presented in Section III-C is bounded by* $\tilde{O}(\frac{n}{t}+t^3)$.

Proof: Recall that the algorithm has two modes: the sampling mode and the contiguous mode. Let us first analyze the running time of the algorithm while staying at the contiguous mode. Naively, while staying at that mode, the algorithm has to pay at each round $O(|\mathcal{D}|) = O(t)$ operations. Since there are at most $O(t^3)$ such rounds, then the total time spent at this mode is bounded by $O(t^4)$. However, we can utilize suffix tree machinery to accelerate the computation process.

Observe that whenever the algorithm enters into the contiguous mode, then at each round i and for each of the potent diagonals d it has to update the cost based on whether $x_{i+1} = y_{i+d+1}$. Also observe that whenever the values match, the diagonal is retained as potent at the next round $i+1$. So if we have an efficient way to determine at row i and for diagonal d, what is the maximal row $i_{max} \geq i$ such that: $x_{[i,\ldots,i_{max}]} = y_{[i,\ldots,i_{max}]+d}$. Using suffix trees machinery one can solve this problem in $O(1)$-time. However, building the suffix tree requires querying the entire strings x, y.

Nevertheless, we may apply the suffix trees machinery only on substrings of x, y of $5t$-length at a time. This ideas also have been implemented in see [29], [31] in the context of edit distance computation in a streaming fashion.

Using suffix trees, we pay $O(t)$-time generating the (truncated) suffix trees. Then, given a row i and for diagonal d, using the suffix tree we can find the maximal row $i_{max} \in \{i,\ldots,i+5t\}$ such that: $x_{[i,\ldots,i_{max}]} = y_{[i,\ldots,i_{max}]+d}$. For a given i, let $\mathcal{D}$ be the set of potent diagonals, and let k_i be the mismatches encountered on the potent diagonals along the next $5t$-rows. Then the running time of the algorithm is bounded by $O(t+k_i^2)$. In total, if we divide the contiguous rounds into blocks of length $5t$ we get that the running time of the contiguous mode is bounded by $t^2(O(t)) + \sum k_i^2 = O(t^3)$.

While staying at the sampling mode at each round the algorithm performs only $O(1)$-operations, provided that the period was kept. This contributes $O(\frac{n\log n}{t})$ factor to the running time. If the period was violated, then the algorithm first performs a binary search to detect a period transition, which requires $O(t\log n)$-time. Then it again can use suffix tree machinery to find out the list of diagonals having a mismatch in the next $2(2t+1)$ rows. This takes $O(t)$-time, and will be applied at most $O(t^2)$-times (as whenever this happens one of the potent diagonals increases its cost).

Summarizing, using suffix tree machinery, the algorithm can be implemented so that its running time complexity is bounded by $\tilde{O}(\frac{n}{t}+t^3)$, as claimed. ∎

I. Distinguishing t vs. $O(t^{2-\epsilon})$-gap in time $\tilde{O}(\frac{n}{t^{1-\epsilon}}+t^3)$

Proposition III.19. *There exists an algorithm that, given as input strings* $x, y \in \{0,1\}^n$ *and an integer* $t \leq \sqrt{n}$*, has query and time complexity bounded by* $\tilde{O}(\frac{n}{t^{1-\epsilon}}+t^3)$*, and satisfies the following:*

- *If* $\Delta_e(x,y) \leq t/2$ *it outputs* `close` *with probability* 1.
- *If* $\Delta_e(x,y) = \Omega(t^{2-\epsilon})$ *it outputs* `far` *with probability at least* $2/3$.

Let us briefly sketch the proof. We keep the same algorithm structure, where the only change is in the rate of sampling: instead of sampling at a rate $\tilde{O}(\frac{1}{t})$, our algorithm samples at a rate $\tilde{O}(\frac{1}{t^{1-\epsilon}})$. That will increase the query and running complexity of the modified algorithm into $\tilde{O}(\frac{n}{t^{1-\epsilon}}+t^3)$.

The proof of correctness follows by the same arguments, where one have to show that while sampling at this rate with high probability the algorithm detects mismatches and period violations at a rate of $\frac{1}{t^{1-\epsilon}}$.

J. Succinct Representation of an Alignment

Our algorithm can succinctly represent an alignment in $\tilde{O}(t^2)$ bits. Note that our algorithm has at most $O(t^2)$ sampling mode, therefore at most $O(t^2)$ contiguous mode. During the ith contiguous mode, we can represent an alignment using $\tilde{O}(k_i)$ bits if k_i is the number of edit operations paid during that contiguous mode. Since, in each sampling mode, the alignment is given by a single diagonal, it can be represented in $\tilde{O}(1)$ bits. Thus, over all the contiguous and sampling modes, representing the alignment requires $\tilde{O}(\sum_i k_i + t^2) = \tilde{O}(t^2)$ bits.

REFERENCES

[1] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[2] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, ser. STOC '15, 2015, pp. 51–58.

[3] A. Abboud, A. Backurs, and V. V. Williams, "Tight hardness results for LCS and other sequence similarity measures," in *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, 2015, pp. 59–78.

[4] K. Bringmann and M. Künnemann, "Quadratic conditional lower bounds for string problems and dynamic time warping," in *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, 2015, pp. 79–97.

[5] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams, "Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, 2016, pp. 375–388.

[6] A. Andoni, R. Krauthgamer, and K. Onak, "Polylogarithmic approximation for edit distance and the asymmetric query complexity," in *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, 2010, pp. 377–386.

[7] G. M. Landau, E. W. Myers, and J. P. Schmidt, "Incremental string comparison," *SIAM J. Comput.*, vol. 27, no. 2, pp. 557–582, 1998.

[8] Z. Bar-Yossef, T. Jayram, R. Krauthgamer, and R. Kumar, "Approximating edit distance efficiently," in *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, Oct 2004, pp. 550–559.

[9] T. Batu, F. Ergun, and C. Sahinalp, "Oblivious string embeddings and edit distance approximations," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, ser. SODA '06. SIAM, 2006, pp. 792–801. [Online]. Available: http://dl.acm.org/citation.cfm?id=1109557.1109644

[10] A. Andoni and K. Onak, "Approximating edit distance in near-linear time," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. ACM, 2009, pp. 199–204.

[11] D. Chakraborty, D. Das, E. Goldenberg, M. Koucký, and M. E. Saks, "Approximating edit distance within constant factor in truly sub-quadratic time," in *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, 2018, pp. 979–990.

[12] B. Saha, "The Dyck language edit distance problem in near-linear time," in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, 2014, pp. 611–620.

[13] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, 2016, pp. 712–725.

[14] A. Apostolico and R. Giancarlo, "Sequence alignment in molecular biology," *Journal of Computational Biology*, vol. 5, no. 2, pp. 173–196, 1998.

[15] G. Navarro, "Approximate text searching," Ph.D. dissertation, University of Chile, 1998. [Online]. Available: https://users.dcc.uchile.cl/ gnavarro/ps/thesis98e.pdf

[16] A. Bolshoy, Z. Volkovich, V. Kirzhner, and Z. Barzily, *Genome Clustering: From Linguistic Models to Classification of Genetic Texts*. Springer Science & Business Media, 2010, vol. 286.

[17] M. Crochemore, G. Fici, R. Mercaş, and S. P. Pissis, "Linear-time sequence comparison using minimal absent words & applications," in *LATIN 2016: Theoretical Informatics*. Springer, 2016, pp. 334–346.

[18] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, "Alignment of whole genomes," *Nucleic acids research*, vol. 27, no. 11, pp. 2369–2376, 1999.

[19] T. J. Treangen and S. L. Salzberg, "Repetitive dna and next-generation sequencing: computational challenges and solutions," *Nature Reviews Genetics*, vol. 13, no. 1, p. 36, 2012.

[20] D. Sokol and F. Atagun, "Tredd - A database for tandem repeats over the edit distance," *Database*, vol. 2010, 2010.

[21] T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami, "A sublinear algorithm for weakly approximating edit distance," in *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '03. ACM, 2003, pp. 316–324.

[22] A. Andoni and H. L. Nguyen, "Near-optimal sublinear time algorithms for ulam distance," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 2010, pp. 76–86.

[23] A. Andoni, P. Indyk, D. Katabi, and H. Hassanieh, "Shift finding in sub-linear time," in *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013*, 2013, pp. 457–465.

[24] B. Saha, "Fast & space-efficient approximations of language edit distance and RNA folding: An amnesic dynamic programming approach," in *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, 2017, pp. 295–306.

[25] M. Boroujeni, S. Ehsani, M. Ghodsi, M. T. Hajiaghayi, and S. Seddighin, "Approximating edit distance in truly subquadratic time: Quantum and mapreduce," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, 2018, pp. 1170–1189.

[26] M. Hajiaghayi, M. Seddighin, S. Seddighin, and X. Sun, "Approximating LCS in linear time: Beating the $\sqrt{n}$ barrier," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, 2019, pp. 1181–1200.

[27] R. Ostrovsky and Y. Rabani, "Low distortion embeddings for edit distance," *J. ACM*, vol. 54, no. 5, pp. 23+, Oct. 2007.

[28] R. Krauthgamer and Y. Rabani, "Improved lower bounds for embeddings into l_1," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, 2006, pp. 1010–1017. [Online]. Available: http://dl.acm.org/citation.cfm?id=1109557.1109669

[29] D. Belazzougui and Q. Zhang, "Edit distance: Sketching, streaming, and document exchange," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, 2016, pp. 51–60.

[30] M. Hajiaghayi, S. Seddighin, and X. Sun, "Massively parallel approximation algorithms for edit distance and longest common subsequence," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, 2019, pp. 1654–1672.

[31] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for computing edit distance without exploiting suffix trees," *CoRR*, vol. abs/1607.03718, 2016. [Online]. Available: http://arxiv.org/abs/1607.03718

[32] E. Ukkonen, "Algorithms for approximate string matching," *Inf. Control*, vol. 64, no. 1-3, pp. 100–118, Mar. 1985.

[33] G. M. Landau and U. Vishkin, "Fast string matching with k differences," *J. Comput. Syst. Sci.*, vol. 37, no. 1, pp. 63–78, 1988.

[34] E. W. Myers, "An O(ND) difference algorithm and its variations," *Algorithmica*, vol. 1, no. 2, pp. 251–266, 1986.

APPENDIX

A. PROOF OF LEMMA II.3

Lemma (II.3). *Let $\Delta_e(x,y) \in \{0,1\}^n$.*

If $\Delta_e(x,y) \leq t$ then with probability 1 the algorithm outputs `close`.

If $\Delta_e(x,y) > 6t^2$ then with probability at least 2/3 the algorithm outputs `far`.

Proof of Lemma II.3: To prove the first part, suppose $\Delta_e(x,y) \leq t$. We will prove for all $S \subseteq [n]$ (and thus with probability 1), for every source-to-sink path τ in the original grid graph $G_{x,y}$, there is in G_S a corresponding source-to-sink path τ_S of the same or lower cost. It would then follow that G_S contains a path from the source $(0,0)$ to the sink $(n,0)$ of cost at most t, and the algorithm outputs `close`.

Given $S \subseteq [n]$ and a path τ in $G_{x,y}$, construct the corresponding path τ_S in G_S as follows. Suppose the vertices τ traverses in row 0 are $(0,0),\ldots,(0,d_0)$; then let τ_S start at $(0,0)$ and traverse the exact same vertices. Now we describe how to extend τ_S iteratively for $j = 0,\ldots,s-1$, where $i_0 = 0$ by convention. Denote the last vertex τ_S traverses in row i_j by (i_j, d_S), and suppose the vertices τ traverses in row i_{j+1} are $(i_{j+1},d),\ldots,(i_{j+1},d+\ell)$. Now we have two cases: if $d_S \leq d+\ell$, extend τ_S by appending $(i_{j+1},d_S)\ldots,(i_{j+1},d+\ell)$; otherwise, extend it by appending (i_{j+1},d_S-1). Finally, denote the last vertex τ_S traverses in row i_s by (i_s,d_S); then extend τ_S by append $(n,0)$, which uses an edge of cost $|d_S|$.

Claim A.1. $c_{G_S}(\tau_S) \leq c_{G_{x,y}}(\tau)$.

Proof of Claim A.1: For each $j = 0,\ldots,s$, let $d_\tau(j)$ denote the last diagonal visited by τ at row i_j, and let $d_{\tau_S}(j)$ be similarly for the path τ_S. Denote by $c_\tau(j)$ the cost of the prefix of τ up to $(i_j, d_\tau(j))$, and similarly $c_{\tau_S}(j)$ for path τ_S and vertex $(i_j, d_{\tau_S}(j))$. We will prove show the following bound on τ_S

$$\forall j = 0,\ldots,s, \qquad c_{\tau_S}(j) + d_{\tau_S}(j) \leq c_\tau(j) + d_\tau(j). \quad (7)$$

Let us now show how this bound implies the claim. By construction, the last edge in τ_S goes from row i_s (the last row in S) to the sink and has cost $|d_{\tau_S}(s)|$, and together with (7) in the case $j = s$, we have

$$\begin{aligned} c_{G_S}(\tau_S) &= c_{\tau_S}(s) + |d_{\tau_S}(s)| \\ &\leq c_\tau(s) + d_\tau(s) - d_{\tau_S}(s) + |d_{\tau_S}(s)|. \end{aligned}$$

Now if $d_{\tau_S}(s) \geq 0$, the last two summands above cancel and we continue

$$= c_\tau(s) + d_\tau(s) + 0 \leq c_\tau(s) + |d_\tau(s)|;$$

otherwise, we have $d_\tau(s) \le d_{\tau_S}(s) < 0$ and we continue

$$\le c_\tau(s) + 0 + |d_{\tau_S}(s)| \le c_\tau(s) + |d_\tau(s)|.$$

In both cases, we obtain $c_{G_S}(\tau_S) \le c_\tau(s) + |d_\tau(s)| \le c_{G_{x,y}}(\tau)$, which proves the claim.

We proceed to proving (7) by induction on j. The base case $j = 0$ holds trivially (with equality), because τ_S is constructed to be identical to τ in row $i_0 = 0$. For the inductive step, we actually show

$$\forall j = 0, \ldots, s-1, \qquad \Delta_j c_{\tau_S} + \Delta_j d_{\tau_S} \le \Delta_j c_\tau + \Delta_j d_\tau, \quad (8)$$

where $\Delta_j f := f(j+1) - f(j)$ for $f(j)$ being any of the 4 terms appearing in (7). The last inequality clearly implies the inductive step. (Alternatively, we can replace the induction by a telescopic sum.)

Now to prove (8), fix $j \in \{0, \ldots, s-1\}$, and observe that in the desired inequality, the LHS is about the subpath of τ_S from row i_j to row i_{j+1}, and similarly the RHS is about the subpath of τ. More precisely, these subpaths are taken to "start" and "end" at the last vertex visited in each row. Assume first that $LHS = 0$. Observe that $\Delta_j c_\tau$ is the cost along that subpath of τ, and every edge in it that increments/decrements the diagonal has cost 1, hence $\Delta_j c_\tau \ge \left|\Delta_j d_\tau\right| \ge -\left|\Delta_j d_\tau\right|$. This proves that in this case, indeed $RHS \ge 0 = LHS$.

Assume next that $LHS > 0$. Suppose towards contradiction that the first edge in that subpath of τ_S (from row i_j to row i_{j+1}) decreases the diagonal; then by its construction, τ_S visits no additional vertices on this row, hence the said subpath of τ_S consists of only one edge, and we see that $LHS = \Delta_j c_{\tau_S} + \Delta_j d_{\tau_S} = 1 - 1 = 0$, which contradicts our assumption. We thus know that the first edge in that subpath of τ_S does not change the diagonal. Clearly, any additional edges in this subpath, if any, must stay in the same row and increment the diagonal, hence their number is exactly $\Delta_j d_{\tau_S} \ge 0$. Observe the fact $d_{\tau_S}(j) \ge d_\tau(j)$, which follows from the construction of τ_S. We can also verify the fact $d_{\tau_S}(j+1) = d_\tau(j+1)$; indeed, one direction ($\ge$) is just the previous inequality (but for $j+1$), and the other direction ($\le$) holds in our case where the first edge of the subpath does not change the diagonal. Combining these two facts, we have $\Delta_j d_{\tau_S} \le \Delta_j d_\tau$. Moreover, the foregoing discussion implies that

$$0 \le \Delta_j d_{\tau_S} \le \Delta_j d_\tau \le \Delta_j c_\tau. \quad (9)$$

Observe that by the foregoing discussion and the definition of G_S,

$$\Delta_j c_{\tau_S} = \mathbb{1}_{\{x_{i_j+1} \ne y_{i_j + d_{\tau_S}(j)+1}\}} + \Delta_j d_{\tau_S} \le 1 + \Delta_j d_{\tau_S}, \quad (10)$$

and let us argue next, by a case analysis, that

$$\Delta_j c_{\tau_S} \le \Delta_j c_\tau, \quad (11)$$

Case 1 of proving (11) is when $d_\tau(j) < d_{\tau_S}(j)$ (i.e., our first fact above holds with strict inequality). Then the derivation of (9) actually gives a stronger bound $\Delta_j d_{\tau_S} + 1 \le \Delta_j d_\tau$. Combining this with (9) and (10) we obtain (11).

Case 2 is when the first edge in that subpath of τ decrements the diagonal. Then later steps in the subpath must increment the diagonal (because the net difference is $\Delta_j(\tau) \ge 0$), and again we obtain a stronger cost bound $\Delta_j c_\tau \ge \Delta_j d_\tau + 2$. Combining this with (9) and (10) we obtain (11).

Case 3 is the remaining scenario, where $d_\tau(j) = d_{\tau_S}(j)$ and the first edge in that subpath of τ does not change the diagonal, and thus has cost $\mathbb{1}_{\{x_{i_j+1} \ne y_{i_j + d_\tau(j)+1}\}}$. Hence,

$$\Delta_j c_\tau \ge \mathbb{1}_{\{x_{i_j+1} \ne y_{i_j + d_\tau(j)+1}\}} + \Delta_j d_\tau.$$

Combining this with (10) implies (11).

Finally, having established (11), we combine it with (9) to derive (8), which we called $LHS \le RHS$, and proves the case $LHS > 0$. This completes the proof of the inductive step and of Claim A.1. ∎

This completes the proof of the first part of Lemma II.3.

To prove the second part, suppose that $\Delta_e(x,y) > 6t^2$. Using Claim II.4 and applying a union bound on all possible rows and diagonals, we get that except with probability $n(2t+1)\frac{1}{3n(2t+1)} \le \frac{1}{3}$, none of the events $B(i,d)$ happens. We conclude the proof by showing that in this case, the shortest path connecting $(0,0)$ and $(n,0)$ has cost strictly larger than t, and therefore our algorithm outputs `far`.

Assume towards contradiction that none of the events $B(i,d)$ happens and yet the shortest path in G_S from $(0,0)$ to $(n,0)$, denoted τ_S, has cost at most t. From τ_S, we construct a new path τ in $G_{x,y}$ as follows. (i) For each edge (i_j, d) to $(i_j, d+1)$ in τ_S, we include the same edge in τ. (ii) For each edge (i_j, d) to $(i_{j+1}, d-1)$, we include the edges corresponding to the path $(i_j, d), (i_j+1, d), \ldots, (i_{j+1}-1, d)$ followed by an edge to $(i_{j+1}, d-1)$. (iii) For each edge (i_j, d) to (i_{j+1}, d), we include the edges corresponding to the path $(i_j, d), (i_j+1, d), \ldots, (i_{j+1}, d)$.

Notice that in case (i), τ_S pays a cost of 1 and so does τ. In case (ii), τ_S pays a cost of 1 and since $B(i_j, d)$ did not happen, τ pays a cost of at most $3t + (d' - d)$.

Now consider the maximal subpaths that are formed in τ by the edges in case (iii). Each of these maximal subpaths can be indexed by a contiguous collection of rows and a single diagonal. Let us denote them by

$(I_1, d_1) \ldots, (I_k, d_k)$. Let $I'_j \subseteq I_j$ be the rows in $I_j \cap S$ for $j = 1, \ldots, k$.

Therefore in τ_S, there is a path through diagonal d_j and rows in I'_j. Let this path pay e_j edit cost in τ_S. Note that they are all from substitution edits. Let $Z = \{z_1, z_2, \ldots, z_{e_j}\}$ be the rows in I'_j in increasing order such that τ_S pays an edit cost on the outgoing edge from (z, d_j) for all $z \in Z$. Since, we avoided the events $B(\min I'_j, d_j), B(z_1 + 2, d_j), \ldots, B(z_{e_j} + 2, d_j)$, the total substitution cost paid in τ while traversing through (I_j, d_j) is at most $3t(e_j + 1)$.

Therefore, over all k, τ pays a total substitution cost of $\sum_{j=1}^{k} 3t(e_j + 1) \leq 3t^2 + 3t\sum_{j=1}^{j} e_j$, since $k \leq t$. Now adding the cost from case (i) and (ii), the overall cost paid by τ is at most $6t^2$. This contradicts the assumption that $\Delta_e(x, y) > 6t^2$.

This completes the proof of Lemma II.3 (both parts). ■

B. Missing Proofs from Section III

Proof of Lemma III.2: We proceed by induction on the grid vertices in lexicographic order (i.e., their row is the primary key and their diagonal is secondary). The base case is the source $(0, 0)$, which follows trivially. To prove the inductive step, consider a vertex v and assume the claim holds for all previous vertices. We now split into two cases.

Case (a): v is dominated. By applying the induction hypothesis to the in-neighbor that dominates v, we obtain a shortest path $(0, 0) = v_0, v_1, \ldots, v_l$ to the in-neighbor v_l that dominates v, hence the cost of this path is $c(v_l) \leq c(v) - 1$. By appending that path with v, we obtain a shortest path to v because its cost is at most $c(v_l) + 1 \leq c(v)$. It remains to show the ordering requirement that all non-potent vertices in this path appear after all potent vertices. If v is non-potent, this is immediate because v is appended. If v is potent then by Definition III.1 the dominating vertex v_l must be potent, and again the ordering requirement follows immediately.

Case (b): $v = (i, d)$ is not dominated, and in particular it is potent. Then a shortest-path to v must be coming from $w_1 = (i - 1, d)$. If this w_1 is potent, then we can obtain a shortest path to w_1 by the induction hypothesis, and appending v to this path satisfies the ordering requirement and gives a shortest path because its cost is at most $c(w_1) + 1 \leq c(v)$. So assume henceforth that w_1 is non-potent, and let us show a contradiction. Then by Definition III.1 it must be dominated by some in-neighbor w_2, and moreover either w_2 is non-potent or it has an outgoing edge of cost 0, i.e., a matching edge (or both). If w_2 is non-potent, then by the same argument (as for w_1), it must be dominated by some in-neighbor w_3. Repeat this argument until reaching the first w_l, $l \geq 2$, that is potent, and thus w_l must have an outgoing edge of cost 0. The path's construction and the monotonicity property (1) imply that $c(v) \geq c(w_1) \geq c(w_2) - 1 \geq \cdots \geq c(w_l) - (l - 1)$. Observe that the path $w_l \to \cdots \to w_2 \to w_1 \to v$ has l edges, the first $l - 1$ are insertion/deletion edges, and the last one is a matching or substitution edge. Consider an alternative path from w_l, that uses first the outgoing edge of cost 0, and then $l - 1$ insertion/deletion edges in exact correspondence with the $l - 1$ edges $w_l \to w_{l-1}, \ldots, w_2 \to w_1$. It is easy to verify that also this alternative path reaches v. and its cost is exactly $l - 1$. Appending this path to an arbitrary shortest-path to w_l, yields a path of cost $c(w_l) + l - 1 \leq c(v)$, i.e., a shortest-path to v that enters v via an insertion/deletion edge. This means that v is dominated, in contradiction to our assumption. ■

Proof of Lemma III.3: First, if $c((i, d) \to (i, d + 1)) = 0$, then $c(i + 1, d) \leq c(i, d) + c((i, d) \to (i, d + 1)) = c(i, d)$. On the other hand, from the monotonicity property (1), $c(i + 1, d) \geq c(i, d)$. Therefore, whenever $c((i, d) \to (i, d + 1)) = 0$, we have $c(i, d) = c(i, d + 1)$.

Therefore, let us assume, $c((i, d) \to (i, d + 1)) = 1$. We prove the claim by a double induction on the grid vertices (i, d) in lexicographic order (i.e., their row is the primary key and their diagonal is secondary).

Base case: Note that by definition $(0, 0)$ is potent. Let (i, d) be the *first* vertex in the lexicography order which is non-potent. Then (i, d) must be dominated by either $(i, d - 1)$ or $(i - 1, d + 1)$ both of which are potent (if they exist). Let w.l.o.g., $(i, d - 1)$ dominates (i, d), then if $c((i, d - 1) \to (i + 1, d - 1)) = 1$, it must hold for (i, d) to be non-potent that $(i - 1, d + 1)$ dominates (i, d) and has $c((i - 1, d + 1) \to (i, d + 1)) = 0$. Thus consider $(i, d - 1)$ dominates (i, d) and has $c((i, d - 1) \to (i + 1, d - 1)) = 0$ (the other case is identical). Then $c(i + 1, d - 1) = c(i, d - 1)$. Overall, we have

$$\begin{aligned} c(i, d) \leq c(i + 1, d) &\leq c(i + 1, d - 1) + 1 \\ &= c(i, d - 1) + 1 = c(i, d), \end{aligned}$$

where the last equality follows from $(i, d - 1)$ dominating (i, d). Therefore, $c(i + 1, d) = c(i, d)$.

For the inductive step, assume the claim is true for all vertices (i', d') with $i' < i$ and consider row i. Let d be the *first* diagonal on row i such that (i, d) is non potent. Since (i, d) is non-potent, it must be dominated by either $(i, d - 1)$ or $(i - 1, d + 1)$. If (i, d) is only dominated by $(i, d - 1)$, then since $(i, d - 1)$ is potent (note d is the first diagonal on row i which is non-potent), then for (i, d) to be non-potent, it must hold $c((i, d - 1) \to (i + 1, d - 1)) = 0$. Now following the same argument as in the base case, we get

$c(i+1,d) = c(i,d)$.

Thus assume, either (i,d) is not dominated by $(i,d-1)$ or $c((i,d-1) \to (i+1,d-1)) = 1$. Then, $(i-1,d+1)$ dominates (i,d), that is $c(i-1,d+1) = c(i,d) - 1$. If $(i-1,d+1)$ is potent, then for (i,d) to be non-potent, we must have $c((i-1,d+1) \to (i,d+1)) = 0$. This implies $c(i,d+1) = c(i-1,d+1)$. Otherwise, $(i-1,d+1)$ dominates (i,d) but $(i-1,d+1)$ is non-potent. Then from the induction hypothesis, $c(i,d+1) = c(i-1,d+1)$. Overall, we have

$$\begin{aligned} c(i,d) \le c(i+1,d) &\le c(i,d+1)+1 \\ &= c(i-1,d+1)+1 = c(i,d). \end{aligned}$$

Therefore, $c(i+1,d) = c(i,d)$. Thus, the claim holds for the first non-potent diagonal on row i. Suppose, again by the inductive hypothesis, the claim holds for all r'th potent diagonal on row i with $r' < r$, and consider the r-th non-potent diagonal d_r at row i, $r > 1$.

If (i,d_r-1) is potent, then the claim follows from the same argument as above, as if d_r is the first diagonal on row i to be non-potent.

Otherwise, (i,d_r-1) is non-potent. Then, by the induction hypothesis, $c(i+1,d_r-1) = c(i,d_r-1)$. If (i,d_r-1) dominates (i,d_r), then we have

$$\begin{aligned} c(i,d_r) \le c(i+1,d_r) &\le c(i+1,d_r-1)+1 \\ &= c(i,d_r-1)+1 = c(i,d_r). \end{aligned}$$

Therefore, $c(i+1,d_r) = c(i,d_r)$.

Otherwise, (i,d_r-1) does not dominate (i,d_r). Hence (i,d_r) must be dominated by $(i-1,d_r+1)$. In that case, if $(i-1,d_r+1)$ is potent, then we must have $c((i-1,d_r+1) \to (i,d_r+1) = 0$. This implies $c(i,d_r+1) = c(i-1,d_r+1)$. On the other hand, if $(i-1,d_r+1)$ is non-potent, then from the induction hypothesis $c(i,d_r+1) = c(i-1,d_r+1)$. We have

$$\begin{aligned} c(i,d_r) \le c(i+1,d_r) &\le c(i,d_r+1)+1 \\ &= c(i-1,d_r+1)+1 = c(i,d_r). \end{aligned}$$

Therefore, we have $c(i+1,d_r) = c(i,d_r)$. The lemma follows. ■

Proof of Lemma III.4: We prove this by induction on the grid vertices (i,d) in lexicographic order (i.e., their row is the primary key and their diagonal is secondary). The base case $(i,d) = (0,0)$ is immediate, because the mismatch guarantees that $c(1,0) \ge 1$.

For the inductive step, consider (i,d) and assume the claim holds for all previous vertices. Suppose first that $(i+1,d)$ has a mismatch, and let us show that $c(i+1,d) = c(i,d)+1$. By (1), it suffices to show that $c(i+1,d) \ge c(i,d)+1$, which due to the mismatch requires proving a lower bound on the cost of two in-neighbors of $(i+1,d)$, namely,

$$\min\{c(i+1,d-1), c(i,d+1)\} \ge c(i,d).$$

This is equivalent to two inequalities, and we start with the inequality $c(i+1,d-1) \ge c(i,d)$. Assume first that (i,d) is not dominated by $(i,d-1)$. This together with the monotonicity property (1) implies $c(i,d) \le c(i,d-1) \le c(i+1,d-1)$, as required. Assume next that (i,d) is dominated by $(i,d-1)$. Then by Definition III.1, diagonal $d-1$ is potent at row i and has a mismatch at the next row $i+1$. Applying the induction hypothesis to $(i,d-1)$ and using the bounded difference property (2), we have $c(i+1,d-1) = c(i,d-1)+1 \ge c(i,d)$, as required. Thus, both cases satisfy the required inequality.

The second inequality $c(i,d+1) \ge c(i,d)$ is proved by a similar argument with two cases depending on whether (i,d) is dominated by $(i-1,d+1)$. This concludes the inductive step in this case.

Suppose next that $(i+1,d)$ has a match. Then this vertex has an incoming edge of cost 0, hence $c(i+1,d) \le c(i,d)$. The other direction $c(i+1,d) \ge c(i,d)$ follows by (1). It remains to prove that $(i+1,d)$ is potent. First, assume (i,d) is not dominated by $(i,d-1)$. This along with the monotonicity property (1) implies $c(i+1,d-1) \ge c(i,d-1) \ge c(i,d) = c(i+1,d)$, i.e. $(i+1,d-1)$ does not dominate $(i+1,d)$.

Otherwise, assume $(i,d-1)$ dominates (i,d), then from Definition III.1, $(i,d-1)$ must be potent and there must be a mismatch at row i. Hence, from the first part of this lemma, $c(i+1,d-1) = c(i,d-1)+1 \ge c(i,d) = c(i+1,d)$, i.e., $(i+1,d-1)$ cannot dominate $(i+1,d)$.

Similarly, $(i,d+1)$ cannot dominate (i,d). Therefore, $(i+1,d)$ is potent. This completes the inductive step and proves the lemma. ■

Proof of Lemma III.6: We prove the first claim (about insertion to $\mathcal{D}_i$) by induction on the grid vertices (i,d) in lexicographic order (i.e., their row is the primary key and their diagonal is secondary). The base case is row $i = 0$, at which the only potent diagonal is $d = 0$, and indeed it is inserted into $\mathcal{D}_0$ as initialization.

For the inductive step, consider a potent vertex (i,d) and assume the claim holds for all previous vertices. By Lemma III.2, there is a shortest path to (i,d) consisting only of potent vertices. This path enters (i,d) from one of its three in-neighbors $(i-1,d)$, $(i-1,d+1)$, and $(i,d-1)$, which then must be potent too. We now have three cases.

Suppose first the shortest path enters from $(i-1,d)$. As mentioned above, this vertex must be potent, and then by the induction hypothesis, d is inserted to the list $\mathcal{D}_{i-1}$. It follows that when $(i-1,d)$ is scanned, the

algorithm will see it is potent and insert d to $\mathcal{D}_i$.

Second, assume the shortest path enters from $(i, d-1)$. As mentioned above, it must be potent, and then by the induction hypothesis, $d-1$ is inserted to the list $\mathcal{D}_i$. Observe that (i,d), which is potent, must be dominated by $(i,d-1)$ because of the shortest path, hence diagonal $d-1$ has a mismatch at the next row $i+1$, and thus when $(i,d-1)$ is scanned, the algorithm will insert d to $\mathcal{D}_i$.

Third, assume the shortest path enters from $(i-1, d+1)$. Similarly to the previous case, this vertex must be potent and then by the induction hypothesis, $d+1$ is inserted to the list $\mathcal{D}_{i-1}$. Since (i,d) is potent and dominated by $(i-1,d+1)$, and diagonal $d+1$ must have a mismatch at the next row i, and thus when $(i-1,d+1)$ is scanned, the algorithm will insert d to $\mathcal{D}_i$.

Finally, to prove the second claim, observe that during the scan of $\mathcal{D}_i$, every diagonal d that is found to be not potent is removed from the list. ■

Proof of Lemma III.7: We prove only the first assertion, as the second one is an immediate consequence of it. The proof is by induction on the grid vertices (i,d) in lexicographic order (i.e., their row is the primary key and their diagonal is secondary). The base case is the time before processing vertex $(0,0)$; at this time, c_{A} stores its initial values, i.e., $c_{\mathrm{A}}[d] = |d|$, which is equal to $c(0,d)$ for all $d \geq 0$. For $d < 0$, the base case is the time before processing $(-d,d)$, because we should only consider vertices reachable from $(0,0)$; at this time, $c_{\mathrm{A}}[d] = |d|$ is still the initialized value and it is equal to $c(d,-d) = -d$.

For the inductive step, we need to show that the processing of diagonal $d \in \mathcal{D}_i$ updates the array c_{A} correctly. But using the induction hypothesis, we only need to show $c_{\mathrm{A}}[d]$ is updated from $c(i,d)$ to $c(i+1,d)$. To this end, suppose first that vertex (i,d) is non-potent. Then by Lemma III.3 we have $c(i+1,d) = c(i,d)$, and the algorithm indeed does not modify $c_{\mathrm{A}}[d]$. Suppose next (i,d) is potent and let us use Lemma III.4: If $(i+1,d)$ has a mismatch then $c(i+1,d) = c(i,d)+1$, and the algorithm indeed increments $c_{\mathrm{A}}[d]$ by 1; and if $(i+1,d)$ has a match then $c(i+1,d) = c(i,d)$, and the algorithm indeed does not change c_{A} at all. ■