Approximating Edit Distance Within Constant Factor in Truly Sub-Quadratic Time[*]

Diptarka Chakraborty[†1], Debarati Das[‡2], Elazar Goldenberg[§3], Michal Koucký[¶4], and Michael Saks[‖5]

[1,2,4]Computer Science Institute of Charles University, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
[3]The Academic College Of Tel Aviv-Yaffo, School of Computer Science, Tel Aviv-Yaffo, Israel
[5]Department of Mathematics, Rutgers University, Piscataway, NJ, USA

July 13, 2018

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. In this paper we study the computational problem of *approximating* the edit distance between a pair of strings.

The problem of computing the exact edit distance can be solved using a classical dynamic programming algorithm that runs in quadratic time. Andoni, Krauthgamer and Onak (2010) show that one can compute an approximation to the edit distance within approximation factor poly($\log(n)$) in nearly linear time. Recently, Abboud and Backurs (ITCS'17) showed that a truly sub-quadratic deterministic time $1 + o(1)$-factor approximation algorithm for the edit distance problem would imply new circuit lower bounds. This raises the question whether edit distance can be approximated within constant factor in truly sub-quadratic time.

In this paper we affirmatively answer this question: We provide an algorithm whose running time is bounded by $\widetilde{O}(n^{2-2/7})$ that approximates the edit distance up-to constant factor. Our approximation algorithm is based on a new yet simple paradigm.

[*]The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787.

[†]diptarka@iuuk.mff.cuni.cz

[‡]debaratix710@gmail.com

[§]elazargo@mta.ac.il

[¶]koucky@iuuk.mff.cuni.cz

[‖]msaks30@gmail.com

1 Introduction

The *edit distance* (aka *Levenshtein distance*) [18] 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 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. Sometimes one also requires that the algorithm finds an *alignment* of x and y, i.e., a series of edit operations that transform x into y.

The problem of computing the exact edit distance (the decision problem) can be solved in quadratic time using a classical dynamic programming based algorithm by Wagner and Fischer [23]. Despite an extensive line of research this running time has been improved only to $O(n^2/\log n)$ by Masek and Paterson [19] and recently Grabowski [15] gave a $O(n^2 \log\log n/\log^2 n)$ time algorithm which is the best known upper bound so far for computing arbitrary edit distance. Backurs and Indyk [7] indicates that this bound cannot be improved significantly unless the Strong Exponential Time Hypothesis (SETH) is false. Moreover Abboud et al. [3] showed that even shaving an arbitrarily large polylog factor from n^2 would imply that NEXP does not have non-uniform NC^1 circuits. More hardness results can be found in [2, 12].

A natural question that arises is whether one can obtain an algorithm that runs in truly sub-quadratic time and *approximates* the edit distance between given input strings. Approximating edit distance would be sufficient for many applications. The state of the art result is the work of Andoni *et al.* [5] that provides an algorithm that approximates edit distance within $(\log n)^{O(1/\epsilon)}$ factor in time $O(n^{1+\epsilon})$. They also show that an $O(\log n)$-approximation is the best possible for a large class of algorithms. Recently, Abboud and Backurs (ITCS'17) showed that a truly sub-quadratic deterministic time $1 + o(1)$-factor approximation algorithm for the edit distance problem would imply new circuit lower bounds. Very recently, Boroujeni *et al.* [11] gave a truly sub-quadratic *quantum* algorithm running in time $\widetilde{O}(n^{1.781})$ for constant factor approximation of edit distance. Their algorithm achieves its speed-up using Grover's quantum search.

In this paper we design the first truly sub-quadratic time *classical* algorithm computing constant factor approximation for edit distance. Formally our main result is as follows:

Theorem 1.1. *There is a randomized algorithm that approximates the edit distance between any pair of input strings $x, y \in \Sigma^n$ within a constant factor, runs in time $\widetilde{O}(n^{12/7})$, and has probability of error at most $1/n^5$.*

Notice, $12/7 = 1.714\ldots$ so our algorithm is faster than that of Boroujeni *et al.* [11]. We did not try to optimize the constant in the approximation factor but we prove it is at most 252. One can easily reduce the approximation factor by tweaking our algorithm, we believe it readily gives an approximation factor $5 + \epsilon$, for any $\epsilon > 0$, while increasing the running time only by a constant factor. The algorithm can be also made to output an approximate alignment of the two strings.

The running time of the algorithm depends on the actual edit distance: If k is the edit distance of the input strings x and y, and $\theta = k/n$ is their normalized edit distance, then our algorithm runs in time $\widetilde{O}(n^{12/7}\theta^{4/7}) = \widetilde{O}(n^{8/7}k^{4/7})$. This should be compared to the best known algorithm for the exact edit distance of Landau *et al.* [17] that runs in time $O(n + k^2)$. (Our analysis of the running time of our algorithm breaks down for too small θ, so in that range we actually use the algorithm of Landau *et al.*)

We can also improve the running time of our algorithm for the price of worse (constant) approximation factor. We can use our algorithm recursively to answer edit distance queries on short strings made by our algorithm. This gives a constant factor approximation algorithm running in time $\widetilde{O}(n^{\frac{1+\sqrt{5}}{2}+\epsilon}) = O(n^{1.618+\epsilon})$, for any fixed $\epsilon > 0$, for edit distance $k \geq n^{1-\beta}$, where $\beta > 0$ is a constant depending on ϵ. We elaborate on this and further improvements in Section 2.

The main idea that our algorithm is based on is as follows: If x and y are of small (normalized) edit distance θ then in the optimal alignment of x and y, many substrings of x are matched with substrings of y of edit distance roughly θ. Our approach is based on sampling random substrings of x and finding for them good matches in y. Then we extend the found matches to larger substrings. This saves time if there are not too many candidates in y that match a sampled substring of x. Our key insight is that one can save also in the case when there are too many candidates for the match. In that case we obtain a substantial information about a non-trivial part of y (and x). In that case we can remove close matches from y and x to reduce their size. In this way we either eventually run out of x, or we find good matches for most of x. This uses an idea closely related to periodicity of strings that was used previously in [14, 13]. The actual details are a little bit more tricky. We elaborate more on our approach in Section 1.2.

1.1 Related work

The question of approximating edit distance has been studied extensively. Landau at al.[17] provide a $\sqrt{n}$-approximation that runs in linear time. This is done by showing an exact algorithm that runs in time $O(n+k^2)$, where k is the edit distance between the input strings. A series of subsequent works improved this approximation factor first to $n^{3/7}$ [8], then to $n^{1/3+o(1)}$ [10] and later to $2^{\widetilde{O}(\sqrt{\log n})}$ [6] while keeping the running time of the algorithm almost linear. Batu *et al.* [9] provided an $O(n^{1-\alpha})$-approximation algorithm that runs in time $O(n^{\max\{\frac{\alpha}{2}, 2\alpha-1\}})$. The approximation factor was further improved to $(\log n)^{O(1/\epsilon)}$, for every $\epsilon > 0$ where the approximation algorithm runs in $n^{1+\epsilon}$ time [5].

The idea behind the algorithm provided by [5] is sampling n^ϵ characters from x and then computing the edit distance between the sampled part of x and y using the dynamic programming algorithm in a way that preserves the distance between the original strings. Using this method one can only approximate the edit distance between the input strings but not find a global alignment converting x into y. Contrary, our algorithm does output such an alignment. Moreover, if one wants to use [5] to get a constant factor approximation then it has to sample a constant fraction of x. However, since in [5] the simple dynamic programming is applied on the sampled string, then this approach cannot be used directly to find a constant factor approximation algorithm in truly sub-quadratic time.

Another way to efficiently approximate edit distance (used in [6]) is based on embedding edit distance metric into Hamming metric. The idea is to first apply embedding algorithm on two input strings and then compute the hamming distance between resulting strings in linear time. In this technique the *distortion factor* of the embedding translates into the approximation factor. However this technique again fails to produce a constant factor approximation to edit distance since a lower bound of $\Omega(\log n)$ on the distortion factor is already known [16].

Recently, Boroujeni *et al.* [11] studied the question of approximating edit distance in the quantum computational model. They show the existence of a quantum algorithm that approximates edit distance within $(7+\epsilon)$ factor whose run time is $\widetilde{O}(n^{2-1/7}/\epsilon)$. They obtain also a faster $\widetilde{O}(n^{1.781})$-time quantum algorithm for some constant factor approximation of edit distance. Our algorithm is classical and faster. Hence with the current state of knowledge, quantum machinery has no advantage over classical computation for computing and approximating edit distance (up to a constant factor).

A special case of computing edit distance is when input strings have no repeated characters. Alternatively we can think of input strings x and y as permutations on n characters. In that case the Ulam distance between x and y is the minimum number of character insertion and deletion operations required to convert x into y. Using a simple reduction to the problem of finding *longest increasing sequence*, Ulam distance can be computed in $O(n \log n)$ time. A recent result [20] provides a randomized algorithm that approximates Ulam distance within $(1+\epsilon)$ factor with high probability in time $\widetilde{O}_\epsilon(\sqrt{n} + n/k)$, where k is the Ulam distance between the input strings.

A recent, active line of work achieves tight lower bounds for fundamental problems under various computational assumptions including Strong Exponential Time Hypothesis (SETH). A celebrated result of Backurs and Indyk [7] proves that computing the edit distance cannot be done in truly sub-quadratic time unless the SETH is false. This was further extended by [4] showing that no truly sub-quadratic algorithm exists even under weaker computational assumptions. Recently, Abboud and Backurs [1] showed that a truly sub-quadratic deterministic time $1+o(1)$-factor approximation algorithm for the edit distance problem would imply new circuit lower bounds. This can be thought of as a barrier to obtaining better approximation factor.

1.2 Our Technique

Previous approximation algorithms share a similar meta structure: As a first stage a sampling procedure or a dimension reduction is used in order to obtain much shorter substrings with similar edit distance. In the second stage known edit distance algorithms are applied on the shrunken strings. Finally, the output of the algorithm is used to estimate the edit distance between the input strings. Our approach differs from this.

We will not compute the edit distance directly, rather we will focus on solving the gap version of edit distance that is for given θ verifying whether the edit distance of two input strings is smaller than θ or substantially larger than θ. To approximate the edit distance we can then try various θ of the form 2^{-i} to find the right cost.

We view the problem of estimating the edit distance between two strings x and y of length[1] n as the problem of finding a shortest path in the edit distance graph $G_{x,y}$ associated with them. The graph $G_{x,y}$ is a directed grid graph with vertices $\{0,1,\ldots,n\} \times \{0,1,\ldots,n\}$ with vertical and horizontal edges of cost 1 and each diagonal edge $(i-1,j-1) \to (i,j)$ of cost 0 or 1 depending on whether the corresponding symbols x_i and y_j match or not. The cost of the shortest path between the vertex $(0,0)$ (*source*) and the vertex (n,n) (*sink*) equals the edit distance between x and y. So our goal is to estimate its cost from above.

A naïve way to estimate the shortest source to sink path in $G_{x,y}$ would be to break the graph into pieces of size $w \times w$ (*w-boxes*), compute the shortest paths between the corners of each box, and then reduce the graph to the corner vertices connected by edges weighted by their shortest paths. The new graph will have size $O(n^2/w^2)$ and we will be able to compute the shortest path from its source to sink in quasi-linear time in its size thanks to its specific structure.

So if we were able to quickly estimate the shortest paths in all the boxes we would get a fast

[1]Throughout the paper we assume that both x and y are of the same length n. This assumption can be easily removed.

algorithm. However, to estimate all these costs by the usual quadratic algorithm, we would need to spend time $O(n^2)$ overall. Moreover, to obtain a good approximation we will need to consider also overlapping boxes. If the normalized cost of the path we are looking for is θ, for $\theta \in [0,1]$, then we need two consecutive $w \times w$ boxes to overlap by at least $(1-\theta)w$ in order to have an overall additive error of the approximation $O(\theta)$. So we need to cover the graph by overlapping boxes as seen in Fig. 3.

Covering in Fig. 3 would give $O(n^2/w^2\theta)$ boxes but we can decrease their number to $O(n^2/w^2)$ using the idea of Ukkonen [22] as we need to focus only on boxes that lie within a strip of width $O(\theta n)$ along the main diagonal of $G_{x,y}$. Hence, we only need to focus on $O(n^2/w^2)$ boxes in total that are horizontally aligned into strips of width w and vertically aligned to multiples of θw. This still does not achieve even the running time $O(kn) = O(\theta n^2)$ of Ukkonen's algorithm.

Covering the path. To reduce the total running time, we will need to reduce the number of boxes for which we compute the edit distance. We will break $G_{x,y}$ into strips of width w_1 (w_1-*strips*) and group adjacent strips into strips of width w_2 (w_2-*strips*), $w_1 \ll w_2$. Our goal is to find a set of boxes with an estimate on their cost that would cover a given shortest path τ under the assumption that the path is of cost at most θ and without knowing τ. We call a box with an estimate on its cost a *certified box.*

By τ-cost of a strip we understand the cost of τ restricted to that strip normalized relative to the width of the strip. It is easy to see that the average τ-cost of w_1-strips is at most θ as all the strips are of the same width. By Markov inequality, in a randomly chosen w_1-strip the cost of τ is at most $k\theta$ with probability at least $1 - 1/k$. In a w_1-strip of τ-cost $\leq k\theta$ there will be a $w_1 \times w_1$ box aligned vertically to the multiples of θw_1 that *tightly covers* τ and is of source to sink cost of at most $O(k\theta)$. By tightly covering τ we mean that τ enters the box close to its source and leaves close to its sink. (Assuming θ is small, the path must use mostly diagonal edges of cost zero.)

Hence, for each $\epsilon_i = \theta 2^i$, we will try to identify vertically aligned boxes in w_1-strips of cost at most ϵ_i. If we do this for each $i = 0, \dots, \log 1/\theta$ and every w_1-strip we will find a good approximation of τ by $w_1 \times w_1$ boxes (see Fig. 1). Still this will not provide us with any time savings.

Sparse case. To save time we will pick a w_1-strip at random in each w_2-strip, and we will find all vertically aligned boxes of cost at most ϵ_i in that strip. If the number of these low-cost w_1-boxes in a particular w_1-strip will be less than a certain chosen threshold d, we will *diagonally extend* each of the low-cost w_1-boxes into a box of size $w_2 \times w_2$ that spans the containing w_2-strip. For this $w_2 \times w_2$ box (*diagonal extension*) we will compute its cost.

If the original w_1-box tightly covers τ then its diagonal extension will also tightly cover τ and its cost will correspond to the τ-cost of the containing w_2-strip. (The tighter the τ is covered by the extension, the better its cost approximates the τ-cost of the w_2-strip. As we do not know the tightness of the cover, we do not know how relevant is the cost of the extension. However, we can make $O(\log\theta)$ deterministic guesses of the tightness and one of them will be a good one. The reader might want to come back to this point later.)

If we were lucky and each w_1-strip that we sampled were *sparse* (contained at most d boxes of edit distance $\leq \epsilon_i$) we would spend the total time $O(d(w_2)^2 n/w_2) = O(dnw_2)$ on computing the certified w_2-boxes. Here, $(w_2)^2$ is the trivial upper bound on computing the cost of the w_2-box. Hence, we would achieve savings if $d \cdot w_2 \ll n$.

Dense case. However, what if all the sampled w_1-strips are *dense*, i.e., contain more that d w_1-boxes of cost at most ϵ_i? If that happens, we are also in a good shape. Let x_I be the substring of x corresponding to a dense w_1-strip. We identify all the (relevant overlapping) substrings of

y, $y_{J_1}, y_{J_2}, \ldots, y_{J_\ell}$, such that their edit distance from x_I is at most $3\epsilon_i$. We also identify all the (relevant non-overlapping) substrings of x, $x_{I_1}, x_{I_2}, \ldots, x_{I_k}$, such that their edit distance from x_I is at most $2\epsilon_i$. By the triangle inequality we know that all pairs of y_{J_j}'s with x_{I_j}'s are of edit distance at most $5\epsilon_i$. Hence, we identify $\ell \cdot k$ w_1-boxes of small edit distance while spending only $O((w_1)^2 n/w_1) = O(nw_1)$ time. The closeness thresholds $3\epsilon_i$ and $2\epsilon_i$ guarantee that for each x_{I_j} we identify all the relevant substrings of y that are at distance at most ϵ_i from it. They must be among y_{J_j}'s. Hence, neither of the strings $x_{I_1}, x_{I_2}, \ldots, x_{I_k}$, needs to be processed anymore so we remove the corresponding w_1-strips from further consideration (for this ϵ_i). We found all the certified boxes we needed in those strips.

The choice of the closeness parameters also guarantees that the next dense w_1-strip that will be sampled will correspond to x_I whose substrings of y at distance at most ϵ_i are disjoint from those of the previous x_I's (see Fig. 2). So we can encounter at most $n/(w_1 \cdot d)$ dense w_1-strips in total as the number of relevant substrings of y is at most $O(n/w_1)$. So the total time spent in processing dense strips is bounded by $O((n/w_1 d) \cdot nw_1) = O(n^2/d)$. For suitably chosen parameters w_1, w_2, d we save in both the dense and sparse cases.

Sampling. There are still several obstacles one needs to deal with. The main obstacle is that when we sample a sparse w_1-strip for given ϵ_i we do not know if the actual τ-cost of the strip is below this threshold. If the τ-cost is above the threshold then the boxes found and their diagonal extensions are most likely completely irrelevant for the path τ. There are two possible remedies to this we know of. One could sample $1/\theta$ sparse w_1-strips in each w_2-strip to make sure that we find a relevant w_1-strip if there is one. This would solve the correctness problem but it would blow-up time for small θ. The other solution that we actually use in our algorithm is based on a magic combinatorial lemma which implies that we need to sample only $O(\log^2 n)$ sparse w_1-strips for each w_2-strip and each ϵ_i.

As ϵ_i ranges over all its possibilities, this guarantees that one of two cases will happen. Either all relevant w_1-boxes that tightly cover τ will be eventually enumerated in some dense strip as ϵ_i increases to a level beyond the τ-cost of the strip and as that strip becomes dense. Or we will sample a sparse w_1-strip with a w_1-box of cost at most ϵ_i that tightly covers τ so its diagonal extension will also tightly cover τ. In the former case, as the strip might become dense only for an ϵ_i far larger than the actual τ-cost of the strip one needs to provide a careful analysis of the overall over-approximation given by the certified w_1-boxes found this way. This is encapsulated in Lemma 4.4.

Implementation. Our actual algorithm implements the above described strategy. In Phase I it identifies all relevant certified boxes with an (over-)estimate on their cost and in Phase II it runs a shortest paths algorithm on those boxes. A proper choice of the parameters gives a sub-quadratic algorithm.

Our algorithm for Phase I processes $G_{x,y}$ in blocks of size roughly $2\theta n \times 2\theta n$ along the main diagonal of $G_{x,y}$. These blocks contain all relevant boxes we need to look at. For each block and each ϵ_i it first identifies and processes all the dense w_1-strips in that block. This happens in procedure DenseStripRemoval(). For each strip in turn, we estimate its density by sampling random boxes in it. For strips that seem dense we remove them and all their siblings while producing a list of certified boxes with estimate $5\epsilon_i$ on their cost. (The running time spent on estimating density of all the strips in DenseStripRemoval() is of the same order as the upper bound on the time needed to process all the dense strips.) We are left with a set of sparse w_1-strips that are processed by SparseStripExtensionSampling(). This procedure samples for each w_2-strip $O(\log^2 n)$ w_1-strips, calculates the diagonal extensions of the low cost w_1-boxes contained in them and outputs the certified diagonal extensions. (For each diagonal extension we output the extension with its true cost increased by 2^{-i}, for $i = 0, \dots, \log 1/\theta$, to account for the possible tightness/looseness of their cover.)

The Phase II shortest path algorithm runs in quasi-linear time using the particular structure of the graph and the fact that the graph contains linear number of edges in terms of the number of its vertices.

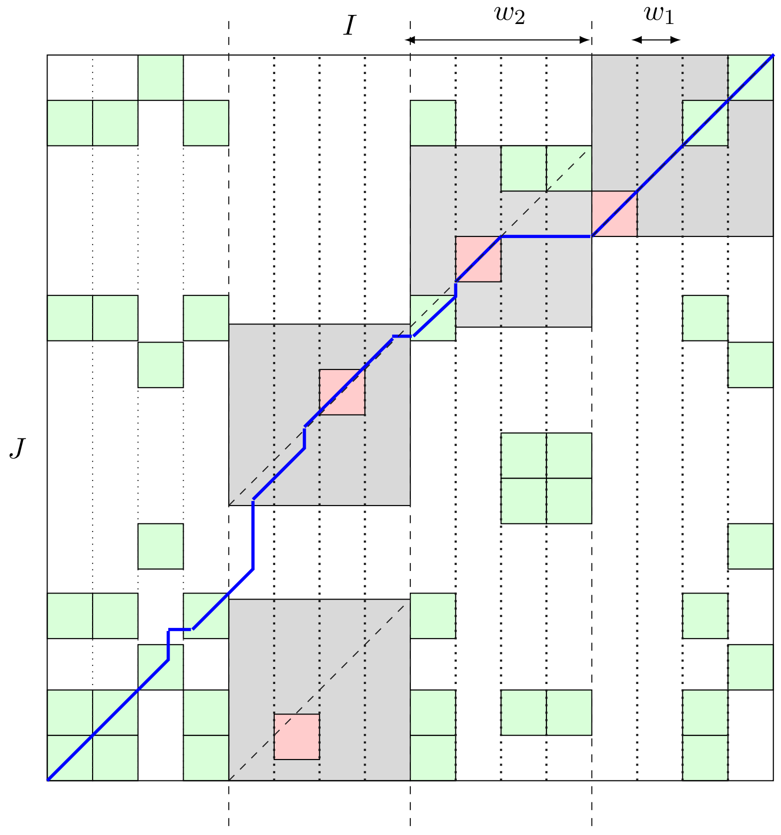


Figure 1: Illustration of the Covering Algorithm: Green boxes are low cost boxes in dense w_1-strips, while the pink ones are in sparse w_1-strips. The blue line corresponds to the path τ that we are trying to cover. In each w_2-strip, τ is covered by either a collection of many w_1-boxes or it is covered by a diagonal extension of a low cost w_1-box. The various boxes might overlap vertically which is not shown in the picture.

2 Main results

Our main algorithm focuses on deciding whether the normalized edit distance $\Delta_{\text{edit}}(x, y)$ of two input strings x and y is above or below a given threshold θ. If it is below the threshold we want

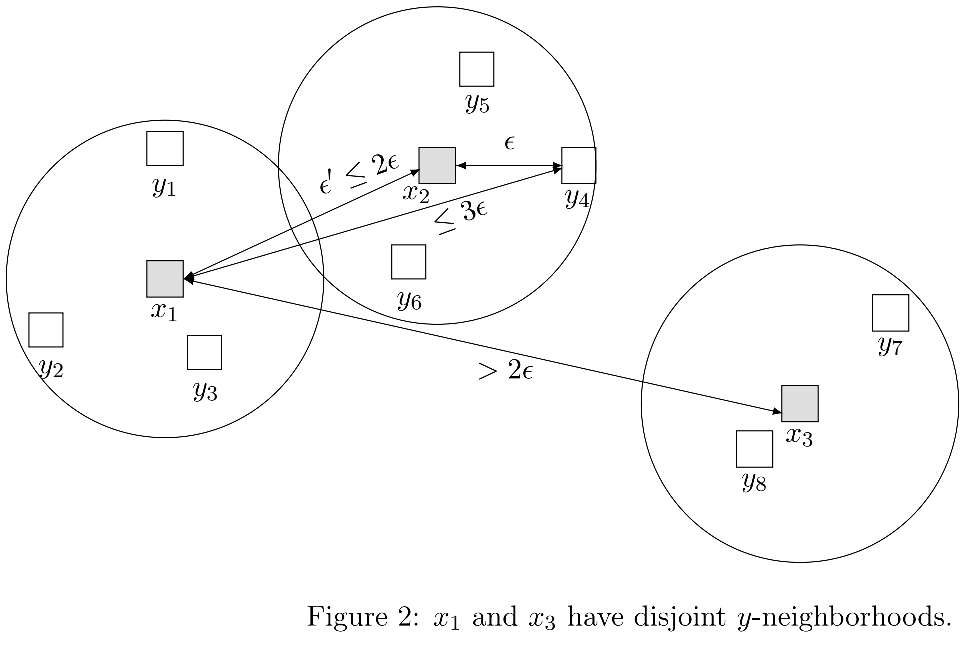


Figure 2: x_1 and x_3 have disjoint y-neighborhoods.

to find a witness (path/alignment) of cost at most $O(\theta)$. So our main focus is on solving the *gap* version of edit distance and we will show how to solve it in sub-quadratic time. We define the Gap-Edit Distance problem as follows.

Gap-Edit Distance(x, y, θ, c): Given two strings x and y each of length n, $\theta \in (0, 1)$, and $c \in [1, \infty)$, the problem is to decide whether $\Delta_{\text{edit}}(x, y) \leq \theta$ or $\Delta_{\text{edit}}(x, y) > c\theta$. More specifically, if $\Delta_{\text{edit}}(x, y) \leq \theta$ one should output 1, and if $\Delta_{\text{edit}}(x, y) > c\theta$ one should output 0, otherwise one may output an arbitrary answer.

Note, for any length n, $\theta \in (0, 1]$ and $c \in [1, \infty)$, the above definition of Gap-Edit Distance$(\cdot, \cdot, \theta, c)$ actually provides a family $\mathcal{F}$ of functions. We say that an algorithm computes Gap-Edit Distance$(\cdot, \cdot, \theta, c)$ if the algorithm computes some function $f \in \mathcal{F}$. For a randomized algorithm we allow the algorithm to provide inconsistent answers on inputs that do not satisfy the promise. Our main result is the following theorem.

Theorem 2.1. *There is a randomized algorithm that given any two strings x and y each of length n, and a real number θ such that $1/n \leq \theta \leq 1$, computes* Gap-Edit Distance$(x, y, \theta, 251)$ *in time $\widetilde{O}(n^{2-2/7}\theta^{4/7})$ with probability of error at most $1/n^6$.*

Most of our paper is devoted to the presentation of the algorithm for solving Gap-Edit Distance$(x, y, \theta,\ 251)$. As a corollary of the above theorem we get an algorithm that approximates the edit distance between any two strings within $O(1)$-factor and runs in sub-quadratic time.

Corollary 2.2 (Restatement of Theorem 1.1)**.** *There is a randomized algorithm that given any two strings x and y each of length n, outputs a value $\kappa \in (0, 1]$ such that $\Delta_{edit}(x, y) \leq \kappa \leq 252\Delta_{edit}(x, y)$ with probability at least $1 - 1/n^5$. Furthermore, if $\Delta_{edit}(x, y) = \theta$, then the algorithm runs in time $\widetilde{O}(n^{2-2/7}\theta^{4/7})$.*

Proof sketch. Fix $\gamma = 1/251$ and define a sequence of parameters $\theta_0, \theta_1, \dots$ as follows: for $i = 0, \dots, \lceil \log_{1+\gamma} n \rceil$, set $\theta_i = (1 + \gamma)^i/n$. Find the smallest i such that Gap-Edit Distance$(x, y, \theta_i,$

251) = 1 using the algorithm from Theorem 2.1. Output $\kappa = \theta_j$. It follows from the definition of Gap-Edit Distance that $\Delta_{\text{edit}}(x, y) \leq \kappa \leq 252\Delta_{\text{edit}}(x, y)$. This completes the proof. □

Further improvements.

Approximation factor. For the sake of simplicity of the presentation we did not optimize the approximation factor in our algorithms. Our algorithms can be easily adjusted to guarantee an approximation factor $5+\epsilon$, for any fixed $\epsilon > 0$, while increasing the running time only by a constant factor. To do that one needs to chose various estimates (such as $\theta, \epsilon_i, \dots$) in smaller increments, align boxes with finer granularity and use boxes which are not only squares but also rectangles of few prescribed ratios between their width and height (constant number of their shapes). That leads to finding a finer approximation to the shortest path in our covering algorithm. The factor 5 comes from the triangle inequality used in the dense case. It is plausible that one can improve also on that factor.

Running time. The algorithm from Theorem 2.1 involves estimating edit distance of strings of length n^δ for some $\delta < 1$. The bulk of its running time comes from this estimation. To compute those estimates we use Ukkonen's algorithm with running time $O(kn)$. Once we have our algorithm from Theorem 2.1 we can use it instead of Ukkonen's algorithm to compute the estimates. This will worsen the approximation factor (in essence multiply it by the approximation factor of the estimation algorithm) but it will reduce the overall running time. This can be done almost in a black-box fashion but one needs to change setting of the internal parameters (w_1, w_2, d) of the algorithm from Theorem 2.1 to maximize the savings. One can iterate this process several (constant number of) times to get faster and faster algorithms with somewhat worse and worse approximation factor. Because of the way our analysis of our algorithm depends on θ, by the iterations we do not get a faster edit distance algorithm for the full range of $\theta \in [0, 1]$ but rather only for θ close to 1. (This is perhaps an artifact of our current simple analysis rather than an inherent limitation. We defer details to the full version of our paper.) We can state it as the following theorem.

Theorem 2.3. *For any $\epsilon > 0$ there are constants $c > 1$ and $\beta \in (0, 1)$ such that there is a randomized algorithm running in time $O(n^{\frac{1+\sqrt{5}}{2}+\epsilon})$ that given two strings x and y each of length n, outputs a value $\kappa \in (0, 1]$ such that $\Delta_{\text{edit}}(x, y) \leq \kappa \leq c\Delta_{\text{edit}}(x, y) + n^{1-\beta}$ with probability at least $1 - 1/n$.*

In particular, we will be able to solve the Gap-Edit Distance in that time for $\theta > n^{-\beta}$. The conditional hardness results for edit distance [7] map instances of SAT to instances of edit distance with θ very close to 1. Hence, solving Gap-Edit Distance is interesting in particular in the range where we do get faster algorithms.

In the special case of one level of the recursion we obtain an algorithm for the full range of $\theta \in [0, 1]$ that runs in time $\widetilde{O}(n^{2-98/277}\theta^{54/277}) = \widetilde{O}(n^{1.647}\theta^{0.195})$. Notice, $\frac{1+\sqrt{5}}{2} = 1.618\dots$.

We can improve the running time even further, close to $O(n^{3/2})$ using additional techniques. Unwinding the recursion one step one notices that the algorithm performs certain closely related edit distance estimations multiple times. One could design a data structure to speed-up these estimates using what was previously calculated. This will improve the running time. We leave details of this improvement to the full version of the paper.

2.1 Organization of the Paper

In Section 3 we present necessary definitions and prove some simple preliminary claims that are used later. In Section 4 we present the Phase I covering algorithm used to find a cover of a path by low-cost certified boxes. In Section 5 we present the Phase II algorithm used to find a shortest path in the graph obtained from Phase I. Section 6 presents the actual reduction between the output of Phase I and input to Phase II. Finally, in Section 7 we combine the parts obtained in the three previous sections into our main algorithm.

3 Preliminaries

Notations. We use $\mathbb{N}$ to denote the set of natural numbers including zero, and for $k \in \mathbb{N}$, we let $[k] = \{1, 2, 3, \ldots, k\}$. All the logarithms we consider throughout this paper are based 2. For any directed graph G we let $V(G)$ denote the set of vertices of G and $E(G)$ denote the set of edges of G.

Grid graph. For intervals $I, J \subseteq \mathbb{N}$, a *grid graph* G on $I \times J$ is a directed weighted graph with vertices $V = I \times J$, where for each $(i,j) \in V$ there is a *horizontal* edge $(i,j) \to (i+1,j)$ of cost one if $i+1 \in I$, there is a *vertical* edge $(i,j) \to (i,j+1)$ of cost one if $j+1 \in J$, and there is a *diagonal* edge $(i,j) \to (i+1,j+1)$ of cost zero or one if $(i+1,j+1) \in V$. The *source* of G, denoted by $\mathrm{source}(G)$, is $(\min I, \min J)$, and the *sink*, denoted by $\mathrm{sink}(G)$, is $(\max I, \max J)$. The *border vertices* of G are all its vertices of in-degree or out-degree less than three. For an integer $d \in \mathbb{Z}$, a *diagonal* d of G is the set of vertices $\{(i+\min I, i+d+\min J) \in I \times J;\ i \in \mathbb{N}\}$. We refer to diagonal 0 as the main diagonal of G. For a grid graph G on $A \times B$ we may refer to A by I_G and to B by J_G. For an interval $I \subseteq \mathbb{N}$ we let $\mu(I) = |I| - 1$. We will be mostly concerned with the *horizontal size* of G so we let $\mu(G) = \mu(I_G)$.

Cost of a path. For a path τ in G, starting at some vertex (i,j) and ending in a vertex (i',j'), we let $\mu(\tau) = i' - i$ denote the horizontal distance traveled by τ. The cost of path τ is the sum of the costs of its edges divided by $\mu(\tau)$ and we denote it by $\mathrm{cost}_G(\tau)$. (The cost is infinite if $\mu(\tau) = 0$.) When G is clear from the context, we will omit the subscript G from the notation and write $\mathrm{cost}(\tau)$. The cost of the grid graph G is the cost of the cheapest path from its source to its sink. Hence, we measure the cost of paths and graphs relative to their horizontal size. We denote the cost of the grid graph G by $\mathrm{cost}(G)$.

Edit distance measure and the edit distance graph. The *edit distance* between two strings x and y of length n and m resp., is defined as the minimum number of character insertion, deletion and substitution operations required to convert x into y. For the sake of simplicity throughout this paper we assume that $m = n$, i.e., x and y are of the same length. Our arguments can be generalized for the case of $m \neq n$ in a straightforward way. Throughout this paper we consider the *normalized edit distance* that is we divide the value of the edit distance by n. We denote the normalized edit distance of x and y by $\Delta_{\mathrm{edit}}(x,y)$.

For strings x and y of length n, their *edit distance graph* $G_{x,y}$ is a grid graph on $\{0,1,\ldots,n\} \times \{0,1,\ldots,n\}$ where each edge $(i,j) \to (i+1,j+1)$ has cost zero if and only if $x_{i+1} = y_{j+1}$. For $I = \{i, i+1, \ldots, i+k\}$, we let x_I denote the substring $x_{i+1}x_{i+2}\ldots x_{i+k}$. Notice that for any two intervals $I, J \subseteq \{0, \ldots, n\}$, G_{x_I, y_J} is isomorphic to the subgraph of $G_{x,y}$ induced on $I \times J$. It is now easy to observe the following.

Proposition 3.1. *Consider any two strings x and y of length n, and let $G = G_{x,y}$. Then for any two non-empty intervals $I, J \subseteq \{0, \ldots, n\}$ of the same size*

$$\Delta_{edit}(x_I, y_I) = cost(G(I, J)),$$

where $G(I, J)$ is the subgraph of G induced on $I \times J$.

W-box and strip. Let $I' \subseteq I \subseteq \mathbb{N}$ and $J' \subseteq J \subseteq \mathbb{N}$ be intervals, and w be an integer. Let G be a grid graph on $I \times J$. The induced subgraph of G on $I' \times J'$ is denoted by $G(I', J')$. If $\mu(I') = w$ and $\min I' - \min I$ is divisible by w then we say that the subgraph $G' = G(I', J)$ is a *w-strip.* (Notice, w-strips do not share horizontal or diagonal edges as they are always horizontally aligned within their graphs.)

We also say that the subgraph $G' = G(I', J')$ is a *w-box* if $\mu(I') = \mu(J') = w$. For $\delta \in [0, 1]$, G' is *δ-grid-aligned* if $\min J'$ is divisible by $\max(\lfloor \delta w \rfloor, 1)$. The *$\delta$-grid-alignment* of G' in G is the w-box $G'' = G(I', J'')$ where $J'' \subseteq J$, and $\min J''$ is the multiple of $\max(\lfloor \delta w \rfloor, 1)$ closest to $\min J'$. (To break ties, pick J'' with the smaller minimum.) Note that G'' is δ-grid-aligned. Hence, we align strips horizontally and boxes vertically, and for our applications we will be interested in w-boxes contained in some w-strip (see Fig. 3).

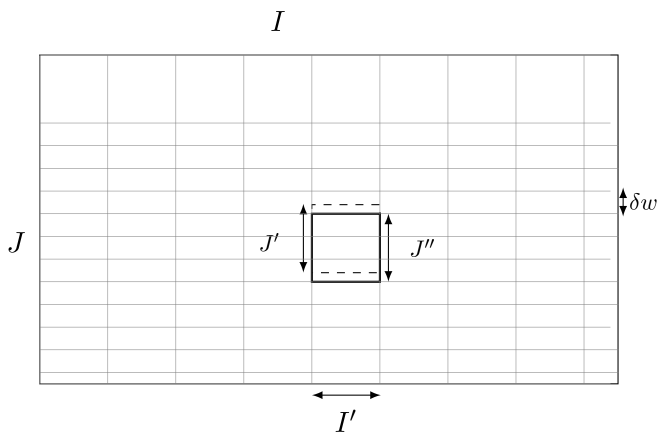


Figure 3: An illustration of δ-grid-alignment and w-strips. The δ-grid-alignment of $G(I', J')$ (dashed rectangle) is $G(I', J'')$ (thick rectangle). Note, since $\lfloor \delta w \rfloor$ is not necessarily a divisor of w, the upper border of δ-grid-aligned rectangles are not necessarily multiples of $\lfloor \delta w \rfloor$. This is not shown in the picture.

Proposition 3.2. *Let G' be a w-box in a grid graph G, and G'' be a δ-grid-alignment of G'. If G' has cost ϵ then G'' has cost at most $\epsilon + 2\delta$.*

Proof. Let $G'' = G(I', J'')$ be a δ-grid-alignment of $G' = G(I', J')$. Without loss of generality we can assume that $\max(\lfloor \delta w \rfloor, 1) = \lfloor \delta w \rfloor$, as otherwise $G'' = G'$, and the claim follows trivially. Let $\ell = |\min J' - \min J''|$. Clearly $\ell \leq \delta w$. (Except for G' near the border of G, $\ell \leq \delta w/2$.)

Let s' be the source of G' and r' be its sink. Similarly, let s'' and r'' be the source and sink of G''. As G' has cost ϵ there is a path τ' from s' to r' of cost ϵ. Consider the case when $\min J'' < \min J'$. Let v be the last vertex of τ' that belongs to G''. We construct a path τ'' from s'' to r'' by concatenating the following three subpaths: the path from s'' to s' consisting of ℓ vertical edges, the subpath of τ' from s' to v, and the path from v to r'' consisting of only horizontal edges. Let k be the number of cost one edges in the subpath of τ' from s' to v. The path from v to r' contains at most $\epsilon w - k$ horizontal edges (of cost one) so the distance from v to r'' is at most $\epsilon w - k + \ell$. Thus the total number of cost one edges in τ'' is at most $\ell + k + \epsilon w - k + \ell = \epsilon w + 2\ell$. Hence, the cost of G'' is at most $\epsilon + 2\delta$. The case of $\min J'' > \min J'$ is symmetric. □

τ-cost of a graph. A *path τ crosses* a grid graph G on $I \times J$ if it starts in some vertex $(\min I, j)$ and ends in some vertex $(\max I, j')$ where $j, j' \in J$. Let G' be a w-strip in G and τ be a path that crosses G. Then the *τ-cost of the strip* G', denoted by τ-cost(G'), is the cost of the shortest subpath of τ that crosses G'. So the *τ-cost*(G') does not account for vertical edges of τ between border vertices of G'. Observe, that τ-cost(G) might be strictly less than cost(τ).

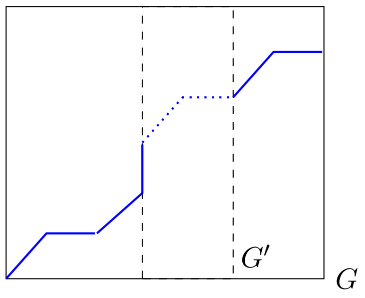


Figure 4: Illustration of a path that crosses a grid: Solid blue edges are the edges of a given path τ. Dotted blue edges are the edges crossing the dashed strip G'.

Proposition 3.3. *Let τ be a path that crosses a grid graph G on $I \times J$. Let w be a divisor of $\mu(I)$. Then the τ-cost of G is at least the average of the τ-costs of distinct w-strips of G.*

Proof. Let $k = \mu(I)/w$, let $G_1, \ldots, G_k$ be the w-strips of G, and for each $i \in [k]$, let τ_i be the shortest subpath of τ that crosses G_i. Notice, that the edges between border vertices shared by G_i and G_{i+1} are part of neither τ_i nor τ_{i+1}. Hence τ_i's do not share any edge. So $\text{cost}(\tau)\mu(\tau) \geq \sum_{i\in[k]} \text{cost}(\tau_i)\mu(\tau_i)$. Since each $\mu(\tau_i) = w$ and $\mu(\tau) = kw$,

$$\tau\text{-cost}(G) \geq \frac{1}{k} \sum_{i\in[k]} \tau\text{-cost}(G_i) = \mathbb{E}_{i\in[k]} \tau\text{-cost}(G_i).$$

□

Definition 1 ($(1-\delta)$-cover)**.** *Let τ be a path that crosses a grid graph G on $I \times J$ where $I, J \subseteq \mathbb{N}$. Let $I' \subseteq I$, $J' \subseteq J$ and w be an integer. Suppose $G' = G(I' \times J')$ is a w-box in G. For $\delta \in [0,1]$ we say that G'* $(1-\delta)$-covers *the path τ if τ passes through a border vertex of G' at horizontal or vertical distance of at most δw from the source of G', and it also passes through a border vertex of G' at horizontal or vertical distance of at most δw from the sink of G'.*

Proposition 3.4. *Let τ be a path that crosses a grid graph G. Let $w < |J|/2$ be a positive integer and $\delta \in [0,1]$. If G' is w-strip in G of the τ-cost at most ϵ then there is a δ-grid-aligned w-box in G' of cost at most $2(\epsilon + \delta)$ that $(1 - \epsilon - \delta)$-covers τ.*

Proof. Let τ' be the shortest subpath of τ crossing G'. Let s' be the starting vertex of τ' and r' be its end vertex. By the assumption the cost of τ' is at most ϵ. Let G'' be the w-box whose source is s'. (Note, if s' is at vertical distance less than w from the top horizontal border of G' then such G'' will not exist. Then apply a similar argument for a box G'' with $\text{sink}(G'') = r'$.) We first show that G'' is of cost at most 2ϵ.

Observe that since the cost of τ' is at most ϵ, then it uses at most ϵw horizontal or vertical edges of cost one, and at least $(1-\epsilon)w$ diagonal edges of cost zero. Let v be the last vertex of τ' that belongs to G''. Clearly, v is a border vertex of G'' that lies on a diagonal at distance at most ϵw from the main diagonal of G'', i.e, v lies on a diagonal d of G'' where $|d| \le \epsilon w$. Hence, we can reach the sink of G'' from v using at most ϵw horizontal or vertical edges. Thus, the cost of G'' is at most 2ϵ.

Now consider $\tilde{G}$ the δ-grid alignment of G'' in G'. By Proposition 3.2 we get that the cost of $\tilde{G}$ is at most $2\epsilon + 2\delta$. By definition, the vertical distance between s' and the source of $\tilde{G}$ is at most δw. By an argument similar to the above, τ' passes a border vertex of $\tilde{G}$ within distance at most $(\delta + \epsilon)w$ of the sink of $\tilde{G}$. Hence, $\tilde{G}$ $(1 - \epsilon - \delta)$-covers τ. □

Note, for simplicity we allow only square boxes, if we were to allow also rectangular boxes of few prescribed dimensions we would get a slightly better approximation factor for the cost of the path. This would increase the running time of the algorithm only by a constant factor.

Definition 2 ((d, δ, ϵ)-sparse)**.** *Let $\epsilon, \delta \in [0,1]$ be reals and d be an integer. Let G be a grid graph on $I \times J$ for $I, J \subseteq \mathbb{N}$. We say that a w-strip G' in G is* (d, δ, ϵ)-sparse *if the number of δ-grid-aligned w-boxes in G' of cost at most ϵ is at most d. If G' is not (d, δ, ϵ)-sparse then it is* (d, δ, ϵ)-dense.

Diagonal extension. Let G be a grid graph on $I \times J$ for $I, J \subseteq \mathbb{N}$. For a w-box $G' = G(I' \times J')$, where $I' \subseteq I$ and $J' \subseteq J$, the *diagonal extension* of G' in G is the $\mu(I)$-box $G'' = G(I \times J'')$ for some $J'' \subseteq J$, whose main diagonal extends the main diagonal of G' (if possible), i.e.,

$$\min J'' = \begin{cases} \min J & \text{if } \min J' - \min J < \min I' - \min I \\ \max J - \mu(I) & \text{if } \max J - \max J' < \max I - \max I' \\ \min J' - (\min I' - \min I) & \text{otherwise} \end{cases}$$

Proposition 3.5. *Let τ be a path that crosses a grid graph G and assume that the τ-cost of G is at most ϵ. Let $w = \mu(I_G)$. Let G' be a w'-box in G that $(1-\delta)$-covers τ. Then the diagonal extension of G' is of cost at most $3\epsilon + 2\delta w'/w$ and $(1 - (\epsilon + \delta w'/w))$-covers τ.*

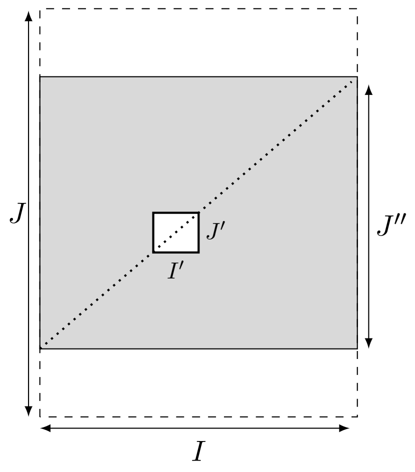


Figure 5: Illustration of diagonal extension: Given a w-box $G(I', J')$ its diagonal extension is the grey box $G(I, J'')$.

Proof. If $\epsilon \geq 1$, the claim follows trivially, so we can assume $\epsilon < 1$. Let G'' be the diagonal extension of G' and τ' be the shortest subpath of τ that crosses G. Let first(τ') and last(τ') be the first and the last vertex of τ' respectively in G. Let h and v be the number of horizontal and vertical edges taken by τ' respectively. As τ-cost of G is at most ϵ, clearly $h + v \leq \epsilon w$. Let $k, k', k'' \in \mathbb{Z}$ be such that, τ' starts on diagonal k of G, source(G') lies on diagonal k' of G and source(G'') lies on diagonal k'' of G. Clearly τ' is confined to the diagonals $\{k - h, \ldots, k + v\}$ of G. As G' $(1 - \delta)$-covers τ so we have $k - h - \delta w' \leq k' \leq k + v + \delta w'$. Hence τ' is confined to the diagonals $\{k' - h - v - \delta w', \ldots, k' + h + v + \delta w'\}$ of G. We first claim that G'' $(1 - (\epsilon + \delta w'/w))$-covers τ' and therefore τ. This is trivially true if $k' = k''$. Next consider the case when $k' < k''$: in this case $\min J_{G''} = \min J_G$ and therefore $k'' = 0$. So τ' is confined to the diagonals $\{k - h, \ldots, k' + h + v + \delta w'\}$. Now as $k \geq 0$, $k - h \geq -h$ and as $k' < 0$, $k' + h + v + \delta w' < h + v + \delta w'$. So τ' is confined to the diagonals $\{-h, \ldots, h + v + \delta w'\}$ and our claim follows. Otherwise consider the case $k' > k''$: this implies $\max J_{G''} = \max J_G$. Thus sink(G'') is at or above last(τ'). Since the number of horizontal edges taken by τ' is at most h, observe that τ' is confined to the diagonals $\{k' - h - v - \delta w', \ldots, k'' + h\}$, i.e., diagonals $\{k'' + (k' - k'') - h - v - \delta w', \ldots, k'' + h\}$. As $k' > k''$, $k' - k'' - h - v - \delta w' > -h - v - \delta w'$ and hence the claim follows.

Next we build a path τ'' from the source to the sink of G'', of cost at most $3\epsilon + 2\delta w/w'$ consisting the following three parts: a path from source(G'') to first(τ'), then from first(τ') to last(τ') along τ', and then from last(τ') to sink(G''). The first and the third part each contributes by at most $\epsilon + \delta w/w'$ to the cost of τ'' and the second part by the cost of τ'. Hence, the cost of τ'' is at most $3\epsilon + 2\delta w/w'$ and the proposition follows. □

Approximation of a path. We conclude the section by introducing the notion of $(1 - \delta, k, \theta)$-*approximation* of a path. Let us define a *certified box* of G to be a triple (I', J', ϵ) where $|I'| = |J'|$ and the cost of $G(I', J')$ is at most ϵ. The first phase of our algorithm will discover a collection of certified boxes, and the second phase will combine the information given by this collection to try to prove that the upper bound on the cost of G is at most $k\theta$. The crucial point will be to show that if the overall cost of G is at most θ, then the collection of certified boxes discovered in the first phase provides enough information for the second phase to discover that the overall cost is at most $k\theta$.

The following definition defines a condition on a collection of certified boxes relative to a particular path τ that crosses G. Lemma 6.5 will show that if this condition holds for some path τ of cost at most ϵ, then the collection of certified boxes is sufficient for the Phase II algorithm to declare that the total cost is at most $O(k\epsilon + \theta + \delta)$.

Let $I, J \subseteq \mathbb{N}$ be intervals, and G be a grid graph on $I \times J$. Let $\theta, \epsilon \in [0, 1]$. Let τ be a path of cost ϵ that crosses G. Let $R = \{(I_1, J_1, \epsilon_1), (I_2, J_2, \epsilon_2), \dots, (I_\ell, J_\ell, \epsilon_\ell)\}$ where for all $i \in [\ell]$, interval $I_i \subseteq I$, interval $J_i \subseteq J, \epsilon_i \in [0, 1]$. Let $\delta \in [0, 1]$ and $k \in \mathbb{N}$. We say that R $(1 - \delta, k, \theta)$-*approximates* τ if all the following hold:

1. For all $i \neq j$, $|I_i \cap I_j| \leq 1$.
2. $\sum_{i \in [\ell]} \mu(I_i) \geq (1 - \delta)\mu(I)$.
3. For each $i \in [\ell]$, $G(I_i, J_i)$ $(1 - \epsilon_i)$-covers τ and is of cost at most ϵ_i.
4. $\sum_{i \in [\ell]} \epsilon_i \mu(I_i) \leq (k\epsilon + \theta)\mu(I)$.

4 Covering Algorithm

In this section, we present the first phase of our algorithm which will produce a collection of certified boxes that will suitably approximate (cover) the optimal path τ. We first provide an informal description of the algorithm. Then we provide the actual pseudo-code.

The algorithm has several parameters $\theta \in [0, 1]$, $d < n$ and $w_1 < w_2 < n$ satisfying $w_1 | w_2$, $w_2 | n$. Parameters w_1, w_2 and d will be set to be a small polynomial in n. First, the algorithm CoveringAlgorithm() calculates $m < n$ such that $w_2 | m$ and $\frac{\theta n}{4} \leq m < \frac{\theta n}{2}$. Then for $k = 0, \dots, \lfloor \frac{n}{m} \rfloor$, it enumerates over $I = J = km, \dots, (k + 8)m$. For each fixed value of I, J the algorithm proceeds in phases $i = 0, ..., \lceil \log 1/\theta \rceil$ associated with $\epsilon_i = 2^{-i}$. In each phase i, the algorithm first invokes the procedure DenseStripRemoval(), followed by SparseStripExtensionSampling() with parameter ϵ_i.

The procedure DenseStripRemoval() on input G, divides I_G into w_1-strips. For each w_1-strip I, it first checks whether the strip is dense or sparse, i.e. whether the number of δ-grid-aligned w_1-boxes of cost at most ϵ_i residing in I exceeds some parameter d, where d, δ and ϵ_i are provided as input to DenseStripRemoval()). This verification is done by sampling. If the strip is sparse, then the algorithm adds the strip into the list S. Otherwise, the algorithm computes two sets: Y the set of all w_1-sized intervals J (in the vertical support of the δ-grid-aligned boxes) such that $\Delta_{\text{edit}}(x_I, y_J) \leq 3\epsilon_i$ and X the set of intervals I' such that $\Delta_{\text{edit}}(x_I, x_{I'}) \leq 2\epsilon_i$. For each pair in $X \times Y$ insert a certified box with cost $5\epsilon_i$ into the list R_1.

The procedure SparseStripExtensionSampling() is then invoked and gets the set S produced by DenseStripRemoval() as an input. The algorithm enumerates over all possible w_2-strip I'. For each I' the algorithm samples sub-intervals $I \in S \cap I'$. For each sampled I, the algorithm enumerates over all possible δ-aligned boxes $I \times J$. For each such box if its cost is at most ϵ_i, then the algorithm checks whether its diagonal extension is of cost at most ϵ_i. If it is then the procedure includes a certified box for the diagonal extension in its final output list R_2. (For each diagonal extension the procedure includes a collection of certified boxes with increasing costs.)

The final output of the algorithm is the set of all certified boxes of R_1 and R_2 over all possible phases i which we denote by R. We now give a pseudo-code for the algorithm.

The algorithm uses two global constants $c_0 \geq 0$ and $c_1 \geq 120$, where the former one is set by Chernoff bound in Proposition 4.7.

```
Algorithm: CoveringAlgorithm($x, y, n, w_1, w_2, d, \theta$)
Input: Strings $x, y$ of length $n$, $w_1, w_2, d \in [n]$, $w_1 | w_2$, $w_2 < \theta n/4$, and $\theta \in [0, 1]$
Output: A set $R$ of certified boxes in $G$.
Initialization: $G = G_{x,y}$, $R_D = R_E = \emptyset$.
Fix an integer $m$ divisible by $w_2$ such that $\frac{\theta n}{4} \leq m < \frac{\theta n}{2}$.
for $k = 0, \ldots, \lfloor \frac{n}{m} \rfloor$ do
    Let $I = J = \{km, km + 1, \ldots, (k + 8)m\}$.
    for $i = \lceil \log 1/\theta \rceil, \ldots, 0$ do
        Set $\epsilon_i = 2^{-i}$.
        Invoke DenseStripRemoval($G(I, J), n, w_1, d/\epsilon_i, \epsilon_i/8, \epsilon_i$) to obtain $S$ and $R_1$.
        Invoke SparseStripExtensionSampling($G(I, J), S, n, w_1, w_2, d/\epsilon_i, \epsilon_i/8, \epsilon_i, \theta$) to obtain
        $R_2$.
        Add items from $R_1$ to $R_D$ and from $R_2$ to $R_E$.
    end
end
Output $R = R_D \cup R_E$.
```

4.1 Proof of correctness

In this section we prove the correctness of our algorithm. In particular we establish the following theorem.

Theorem 4.1. *Let x, y be strings of length n, $1/n \leq \theta \leq 1$ be a real. Let $w_1 \leq w_2 \leq \theta n/4$, $1 \leq d \leq n$ be integers, where $w_1 | w_2$, $w_2 | n$, and $w_1 \leq \theta w_2$. Let R be the set of certified boxes obtained by running CoveringAlgorithm$(x, y, n, w_1, w_2, d, \theta)$ with $c_1 > 120$. The following hold:*

1. *For every certified box $(I, J, \epsilon) \in R$, $\Delta_{edit}(x_I, y_J) \leq \epsilon$.*
2. *For every path τ from the source to the sink in $G = G_{x,y}$ of cost $\eta \leq \theta$, with probability at least $1 - 1/n^9$, there is a subset of R that $(1, 35, 15\theta)$-approximates τ.*

The first part of the theorem is straightforward given the logic of the algorithm. The main part of the theorem is the second part. We first give a brief overview of the proof of that part.

We prove that for any path τ from the source to sink in G, with high probability, there exist a subset of R that approximates the cost of τ upto a constant factor. Using averaging argument, the claim holds if it is true restricted to every w_2-strip of G. So let us consider a single w_2-strip and let T be the set of all its w_1-strips. We know that for every w_1-strip there exists a δ-aligned w_1-box of cost three times the τ-cost of the w_1-strip that tightly covers τ. Our goal is to show that the CoveringAlgorithm either finds *all* such δ-aligned certified w_1-boxes corresponding to each w_1-strip (with the right cost) or finds at least one such δ-aligned w_1-box such that its corresponding w_1-strip is in list S. In the latter case the diagonal extension of this w_1-box indeed tightly covers τ over the entire w_2-strip and has cost roughly three times the τ-cost of the w_2-strip, so our algorithm indeed identifies this certified w_2-box.

Procedure: DenseStripRemoval($G, n, w, d, \delta, \epsilon$)

Input: $G = G_{x,y}(I_G, J_G)$ for some $I_G, J_G \subseteq \{0, 1, \ldots, n\}$, $w, d \in [n]$, $w | \min I_G$, and $\delta, \epsilon \in [0, 1]$

Output: A set S of (d, δ, ϵ)-sparse w-strips in G, and a set R of δ-grid-aligned certified w-boxes of cost at most 5ϵ contained in w-strips of G that are not in S so that all such certified boxes of cost at most ϵ are in R.

```
Initialization: $S = R = \emptyset$.
Define horizontal support of $w$-strips: $T = \{I \subseteq I_G;\ I = \{iw, iw + 1, \ldots, (i + 1)w\}$ for some $i \in \mathbb{N}\}$.
Define vertical support of $\delta$-grid-aligned boxes: $B = \{J \subseteq J_G;\ J = \{j, j + 1, \ldots, j + w\}$ for some $j \in J_G$ divisible by $\max(\lfloor \delta w \rfloor, 1)\}$.
**while** *$T$ is non-empty* **do**
    Pick $I \in T$
    Sample $c_0 |B| \frac{1}{d} \log n$ intervals $J \in B$ uniformly at random and compute $\Delta_{\text{edit}}(x_I, y_J)$.
    **if** *for at most $\frac{c_0}{2} \log n$ sampled $J$'s, $\Delta_{edit}(x_I, y_J) \leq \epsilon$* **then**
        Add $I$ to $S$, and remove $I$ from $T$.
    **end**
    **else**
        Compute:
        $Y = \{J \in B;\ \Delta_{\text{edit}}(x_I, y_J) \leq 3\epsilon\}$.
        $X = \{I' \in T;\ \Delta_{\text{edit}}(x_I, x_{I'}) \leq 2\epsilon\}$.
        Add $(I', J', 5\epsilon)$ to $R$ for all pairs $(I', J') \in X \times Y$.
        Set: $T = T \setminus X$.
    **end**
**end**
Output $S$ and $R$.
```

Procedure: SparseStripExtensionSampling$(G, S, n, w_1, w_2, d, \delta, \epsilon, \theta)$

Input: $G = G_{x,y}(I_G, J_G)$ for some $I_G, J_G \subseteq \{0, 1, \ldots, n\}$, $w_1, w_2, d \in [n]$, $w_1 | w_2$ and $w_2 | \min I_G$, S is a set of sparse w_1-strips of G and $\delta, \epsilon, \theta \in [0, 1]$
Output: A set R of certified w_2-boxes in G.

```
Initialization: $R = \emptyset$.
Define vertical support of $\delta$-grid-aligned boxes: $B = \{J \subseteq J_G;\ J = \{j, j+1, \ldots, j+w_1\}$ for
some $j \in J_G$ divisible by $\max(\lfloor \delta w_1 \rfloor, 1)\}$.
for $i = 0, \ldots, \lfloor \frac{\mu(I_G)}{w_2} \rfloor - 1$ do
    Define a $w_2$-strip: $I' = \{\min I_G + i w_2, \ldots, \min I_G + (i+1) w_2\}$.
    Sample $c_1 \log^2 n$ intervals $I \in S$ uniformly at random subject to $I \subseteq I'$.
    for each sampled $I$ and each $J \in B$ do
        if $\Delta_{edit}(x_I, y_J) \leq \epsilon$ then
            Let $J'$ be such that $G(I', J')$ is the diagonal extension of $G(I, J)$ in $G(I', J_G)$.
            if $\Delta_{edit}(x_{I'}, y_{J'}) \leq \epsilon$ then
                For $k = 0, \ldots, \lceil \log n \rceil$, add $(I', J', \Delta_{\mathrm{edit}}(x_{I'}, y_{J'}) + \theta + 2^{-k})$ to $R$.
            end
        end
    end
end
Output $R$.
```

The algorithm proceeds in $t + 1$ phases, $t = \lceil \log 1/\theta \rceil$. For each phase $i = t, \ldots, 0$ and $\epsilon_i = 2^{-i}$, it calculates a set S_i of sparse w_1-strips in DenseStripRemoval(). We show that the latter case occurs if there exists at least one phase i such that at least $(1/12t)$-fraction of w_1-strips from S_i have τ-cost at most ϵ_i. These are the good strips in S_i that SparseStripExtensionSampling() tries to find by sampling and expand their w_1-boxes into w_2-boxes that tightly cover τ. If there are only few good strips in S_i then our algorithm detects their w_1-boxes that tightly cover τ only at some later phase k when their corresponding w_1-strips are not present in S_k. (At that point those strips are either dense or similar to dense strips, and their w_1-boxes of small cost are reported by DenseStripRemoval().)

For a w_1-strip let ϵ_j be the smallest value such that the τ-cost of the strip is at most ϵ_j and $\epsilon_k \geq \epsilon_j$ be the smallest such that the strip is not in S_k. So instead of finding the w_1-box that tightly covers τ in the strip during phase j we will find it only during phase k. So although the right cost of the box is ϵ_j we associate it with a lager cost of ϵ_k. As we assume that for each phase i, the number of good strips is small so is the number of w_1-boxes that are associated with excessively large cost. Therefore the sum of the detected costs of such certified w_1-boxes is bounded by a small multiple of their true cost (using Lemma 4.4), and the claim follows.

We now proceed with the actual proof of correctness of the algorithm.

Proof of Theorem 4.1. Let R_E and R_D be the sets obtained during the main loop of CoveringAlgorithm, i.e., $R = R_E \cup R_D$. Each certified box in R_E satisfies the first part of the claim trivially as can be easily seen from the algorithm for SparseStripExtensionSampling(). So we need to argue only about certified boxes from R_D. As can be seen from the code of DenseStripRemoval(), for each certified box $(I, J, \epsilon) \in R_D$, there is some interval I' such that $\Delta_{\mathrm{edit}}(x_{I'}, y_J) \leq \frac{3}{5} \cdot \epsilon$ and

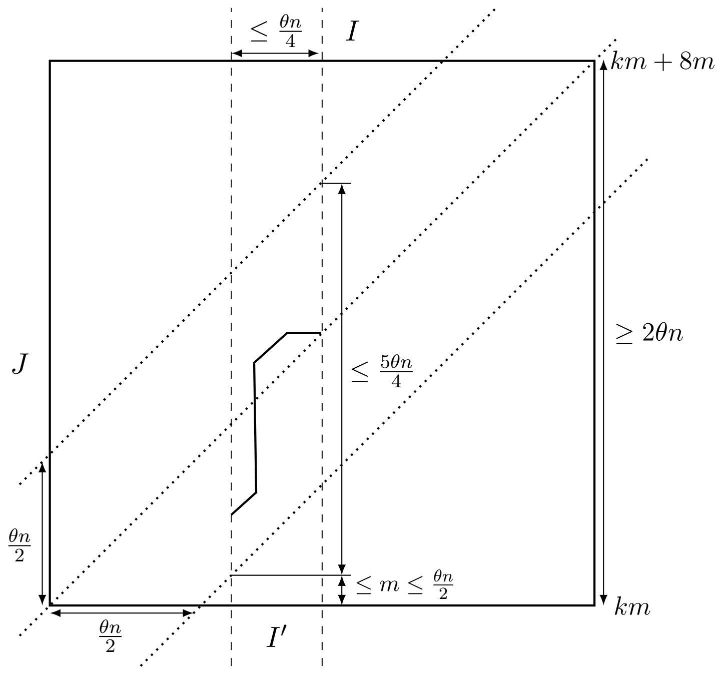
$\Delta_{\text{edit}}(x_{I'}, x_I) \leq \frac{2}{5} \cdot \epsilon$. So by the triangle inequality, $\Delta_{\text{edit}}(x_I, y_J) \leq \epsilon$, which we wanted to prove.

So we only need to establish the second part. Fix a path τ from source to sink in $G = G_{x,y}$ of cost θ. For a w_2-strip G' of G let $\tau_{I_{G'}}$ be the shortest subpath of τ that crosses G' and $\epsilon_{I_{G'}}$ be its cost. By Proposition 3.3, $\sum(15\theta + 35\epsilon_{I_{G'}})\mu(I_{G'}) \leq (15\theta + 35\eta)\mu(I_G)$, where the sum is over all w_2-strip G' of G. So it suffices to show that for each w_2-strip G' of G there is a subset of R that $(1, 35, 15\theta)$-approximates $\tau_{I_{G'}}$.

The main loop of CoveringAlgorithm() processes G in overlapping pieces which cover the relevant parts of G. In particular, each $\tau_{I_{G'}}$ is entirely covered by one of the pieces.

Claim 4.2. *Let G' be a w_2-strip of G, and let $I' = I_{G'}$. There exist intervals $I, J \subseteq \mathbb{N}$, $I = J$ which are enumerated in the main loop of CoveringAlgorithm() such that $I' \subseteq I$ and τ crosses $G(I', J)$.*

Proof. Since τ is of cost at most θ, it cannot use more than $\theta n/2$ horizontal edges as for each horizontal edge of cost 1, it must use one vertical edge of cost 1. Similarly for vertical edges. So τ is confined to diagonals $\{-\theta n/2, \ldots, 0, \ldots, \theta n/2\}$ of G. By the choice of m in CoveringAlgorithm, there will be I and J considered in the main loop of the algorithm such that $I' \subseteq I$ and τ crosses $G(I, J)$. In particular, $I = J = \{km, km+1, \ldots, (k+8)m\}$, where k is the largest integer such that $km \leq \min I' - \frac{\theta n}{2}$ has the desired property. See Fig. 6.



$\leq \frac{\theta n}{4}$
I
$km + 8m$
$\geq 2\theta n$
J
$\leq \frac{5\theta n}{4}$
$\frac{\theta n}{2}$
$\leq m \leq \frac{\theta n}{2}$
km
$\frac{\theta n}{2}$
I'

Figure 6: For a path τ from the source to sink of G and w_2-strip G' on $I' \times J_G$ there are suitable I and J such that τ restricted to G' is contained in $G(I, J)$.

□

Let us fix a w_2-strip G' of G, let $I' = I_{G'}$, $\tau' = \tau_{I_{G'}}$, and let I, J be as provided by the claim. Let $t = \lceil \log 1/\theta \rceil$. For $i = 0, \ldots, t$, let $\epsilon_i = 2^{i-t}$ and let S_i be the set S obtained by calling DenseStripRemoval$(G(I,J), n, w_1, d/\epsilon_i, \epsilon_i/8, \epsilon_i)$. ($S_i$'s depend on the random choices made by DenseStripRemoval() but for the rest of the proof we fix this randomness arbitrarily.) We need the following claim about low-cost certified w_1-boxes that reside in w_1-strips not in S_i.

Claim 4.3. *Let $i \in \{0, \dots, t\}$. Let $I'' \subseteq I$ and $J'' \subseteq J$ be intervals such that $G'' = G(I'', J'')$ is an $\epsilon_i/8$-grid-aligned w_1-box of cost at most ϵ_i that resides in some w_1-strip of $G(I,J)$. If $I'' \notin S_i$ then $(I'', J'', 5\epsilon_i) \in R_D$.*

Proof. If $I'' \notin S_i$ then there is an interval $\tilde{I}$ picked from T during some iteration of the main loop of DenseStripRemoval$(G(I,J), n, w_1, d/\epsilon_i, \epsilon_i/8, \epsilon_i)$ such that $\Delta_{\text{edit}}(x_{\tilde{I}}, x_{I''}) \leq 2\epsilon_i$. Since $\Delta_{\text{edit}}(x_{I''}, y_{J''}) \leq \epsilon_i$, $\Delta_{\text{edit}}(x_{\tilde{I}}, y_{J''}) \leq 3\epsilon_i$. Hence, $I'' \notin S_i$ implies, that $I'' \in X$ and $J'' \in Y$ during the iteration for $\tilde{I}$ of the main loop of DenseStripRemoval(). Thus, DenseStripRemoval() adds the certified box $(I'', J'', 5\epsilon_i)$ into R that becomes part of R_D. □

We need the following lemma about pairs of reals.

Lemma 4.4. *Let $t \geq 2$ be an integer, $c > 1$ and $\gamma, \epsilon_1, \dots, \epsilon_t \in (0,1]$ be real numbers, where $\epsilon_t = 1$ and for all $i < t$, $\epsilon_i = \epsilon_{i+1}/(1+\gamma)$. Let $(a_1, b_1), \dots, (a_l, b_l) \in \{\epsilon_1, \dots, \epsilon_t\}^2$ be ℓ pairs such that, for all $j \in [\ell]$, $a_j \leq b_j$. For every $i \in [t]$, define sets $S_i = \{j \in [\ell];\ a_j \leq \epsilon_i < b_j\}$, and $B_i = \{j \in [\ell];\ \epsilon_i < a_j\}$. If for all $i \in [t]$, $|S_i| \leq \frac{|B_i|}{c(t-1)}$, then*

$$\sum_{j \in [\ell]} b_j \leq \left(1 + \frac{1+\gamma}{c}\right) \sum_{j \in [\ell]} a_j.$$

Proof. First we argue that for all $i \in [t]$, $|B_i| \leq \frac{\sum_{j \in [\ell]} a_j}{\epsilon_i}$. Consider any $i \in [t]$. $\sum_{j \in [\ell]} a_j \geq \sum_{j \in B_i} a_j > \epsilon_i \cdot |B_i|$. But this implies $|B_i| \leq \frac{\sum_{j \in [\ell]} a_j}{\epsilon_i}$.

For each $i \in [t]$, define a set $M_i = \{j \in [\ell];\ a_j < \epsilon_i = b_j\}$. Notice that $M_1 = \emptyset$ and also observe that for $i > 1$, $M_i \subseteq S_{i-1}$. This is because, if $j \in M_i$ then $a_j < \epsilon_i = b_j$, which implies $a_j \leq \epsilon_{i-1} < b_j$. Hence $j \in S_{i-1}$. So, $|M_i| \leq |S_{i-1}|$. We can write,

$$\begin{aligned}
\sum_{j \in [\ell]} b_j &= \sum_{\substack{j \in [\ell] \\ b_j = a_j}} b_j + \sum_{\substack{j \in [\ell] \\ b_j > a_j}} b_j \leq \sum_{j \in [\ell]} a_j + \sum_{i \in [t]} \sum_{\substack{j \in [\ell] \\ b_j = \epsilon_i \\ b_j > a_j}} b_j \\
&= \sum_{j \in [\ell]} a_j + \sum_{i \in [t]} \sum_{j \in M_i} \epsilon_i = \sum_{j \in [\ell]} a_j + \sum_{i \in [t]} |M_i| \cdot \epsilon_i \\
&\leq \sum_{j \in [\ell]} a_j + \sum_{i \in [t-1]} |S_i| \cdot \epsilon_{i+1} \leq \sum_{j \in [\ell]} a_j + \sum_{i \in [t-1]} \frac{|B_i|}{c(t-1)} \cdot \epsilon_{i+1} \\
&\leq \sum_{j \in [\ell]} a_j + \sum_{i \in [t-1]} \frac{\epsilon_{i+1}}{c(t-1)} \cdot \frac{\sum_{j \in [\ell]} a_j}{\epsilon_i} \\
&= \sum_{j \in [\ell]} a_j + \frac{(1+\gamma)}{c(t-1)} \cdot \sum_{i \in [t-1]} \sum_{j \in [\ell]} a_j \leq \left(1 + \frac{1+\gamma}{c}\right) \sum_{j \in [\ell]} a_j
\end{aligned}$$

□

The theorem now follows from the following claim by applying a union bound on the event that for some w_2-strip I', the algorithm does not approximates τ'.

Claim 4.5. *With probability at least $1 - 1/n^{10}$ over the choices made by invocations of SparseStripExtensionSampling(), at least one of the sets R_E and R_D contains $(1, 35, 15\theta)$-approximation of τ'.*

Proof. Let ϵ be the cost of τ'. Let $T = \{I'' \subseteq I';\ I'' = \{iw_1, iw_1 + 1, \ldots, (i+1)w_1\}$ for some $i \in \mathbb{N}\}$ corresponds to all the w_1-strips of $G(I', J)$. For $I'' \in T$, let $\epsilon_{I''}$ be the τ-cost of $G(I'', J)$. By Proposition 3.4, for all $a \geq \epsilon_{I''}$ and all $\delta \leq a/2$, there is a δ-grid-aligned w_1-box in $G(I'', J)$ of cost at most $3a$ that $(1-2a)$-covers τ. Let $s_{I''}$ be the smallest integer such that $\epsilon_{s_{I''}} \geq \max\{3\epsilon_{I''}, \theta\}$. Let $t_{I''} \geq s_{I''}$ be the smallest integer such that $I'' \notin S_{t_{I''}}$. Define $a_{I''} = \epsilon_{s_{I''}}$ and $b_{I''} = \epsilon_{t_{I''}}$.

Hence, for all $a \in [a_{I''}, b_{I''}]$, there is a $a/8$-grid-aligned w_1-box in $G(I'', J)$ of cost at most a that $(1-a)$-covers τ. By the definition of $b_{I''}$ and Claim 4.3, R_D contains a certified box $(I'', J_{I''}, 5b_{I''})$, for some $J_{I''}$, where $G(I'', J_{I''})$ is of cost at most $5b_{I''}$ and $(1 - 5b_{I''})$-covers τ.

For each i, let $S'_i = \{I'' \in T;\ a_{I''} \leq \epsilon_i < b_{I''}\}$ and $B_i = \{I'' \in T;\ \epsilon_i < a_{I''}\}$. Clearly, $S'_i \subseteq S_i \cap T \subseteq S'_i \cup B_i$. There are two cases.

Case I. For all $i \in \{0, \ldots, t\}$, $|S'_i| \leq |B_i|/12t$:

By Lemma 4.4:

$$\begin{aligned}
\sum_{I'' \in T} b_{I''} &\leq (1 + \frac{2}{12}) \sum_{I'' \in T} a_{I''} \\
&\leq 2(1 + \frac{1}{6}) \sum_{I'' \in T} (3\epsilon_{I''} + \theta) \\
&\leq (7\epsilon + 3\theta)|T|.
\end{aligned}$$

The second inequality comes from $a_{I''} = \epsilon_{s_{I''}} \leq 2(3\epsilon_{I''} + \theta)$ by the choice of $s_{I''}$, and the last inequality uses $\sum_{I'' \in T} \epsilon_{I''} \leq \epsilon|T|$ by Proposition 3.3. So $\{(I'', J_{I''}, 5b_{I''});\ I'' \in T\}$ $(1, 35, 15\theta)$-approximates τ' on G' which is part of R_D by Claim 4.3.

Case II. There exists $i \in \{0, \ldots, t\}$ satisfying $|S'_i| > |B_i|/12t$:

We show that with probability at least $(1 - 1/n^{10})$ some of the diagonal extensions found by SparseStripExtensionSampling$(G(I, J), S_i, n,\ w_1, w_2, d/\epsilon_i, \epsilon_i/8, \epsilon_i, \theta)$ $(1, 8, 4\theta)$-approximates τ'. Indeed, if the procedure samples $I \in S'_i$ while sampling from $S_i \cap T$ then it will find an $\epsilon_i/8$-grid-aligned w_1-box of cost at most ϵ_i that $(1-\epsilon_i)$-covers τ. By Proposition 3.5, its diagonal extension $G'' = G(\tilde{I}, \tilde{J})$ is of cost at most $3\epsilon + 2\epsilon_i w_1/w_2 \leq 3\epsilon + 2\theta$ and $(1 - (\epsilon + \theta))$-covers τ'. If $\epsilon = 0$ then the singleton set $\{(\tilde{I}, \tilde{J}, \mathrm{cost}(G'') + \theta + 2^{-\lceil \log n \rceil})\}$ is a subset of R_E by the behavior of SparseStripExtensionSampling() and it $(1, 8, 4\theta)$-approximates τ'. Otherwise $\epsilon \geq 1/n$ so set $k = \lfloor \log 1/\epsilon \rfloor$. Thus, $k \leq \log n$ and $2^{-k} \in [\epsilon, 2\epsilon)$. Then $\{(\tilde{I}, \tilde{J}, \mathrm{cost}(G'') + \theta + 2^{-k})\} \subseteq R_E$ and it $(1, 8, 4\theta)$-approximates τ'. Since $t \leq \lceil \log n \rceil$, the probability of not sampling $I \in S'_i$ is at most $(1 - 1/(12\lceil \log n \rceil))^{c_1 \log^2 n} \leq 1/n^{10}$, for n large enough. □

□

4.2 Time complexity of Covering Algorithm

Let $t(n, \epsilon)$ denote the running time of an algorithm that decides whether two strings of length n have edit distance at most ϵ and if they do it computes their edit distance. For simplicity we assume that t is Lipschitz in ϵ so that for each $k \geq 1$, there is a constant c such that for all $\epsilon \in [0,1]$ and all $n > 1$, $t(n, k\epsilon) \leq c \cdot t(n,\epsilon) + c$. Let us also assume that $t(n,\epsilon) \geq n$.

Theorem 4.6. *Let c_0 by a constant from Proposition 4.7 below. Let n be a large enough integer. Let x, y be strings of length n, $1/n \leq \theta \leq 1$ be a real. Let $\log n \leq w_1 \leq w_2 \leq \theta n/4$, $1 \leq d \leq n$ be integers, where $w_1|w_2$ and $w_2|n$, and $w_1/w_2 \leq \theta$. With probability at least $1 - n^{-7}$ the CoveringAlgorithm$(x, y, n, w_1, w_2, d, \theta)$ runs in time at most*

$$O\left(\sum_{\substack{k=\lceil \log 1/\theta \rceil,\dots,0 \\ \epsilon = 2^{-k}}} \frac{\theta n^2 \log n}{d\epsilon w_1^2} \cdot t(w_1,\epsilon) + \frac{\theta n^2 \log^2 n}{w_1 w_2 \epsilon} \cdot t(w_1,\epsilon) + \frac{nd \log^2 n}{w_2 \epsilon} \cdot t(w_2,\epsilon) \right)$$

and outputs the set R of size at most $O((\frac{n}{w_1})^2 \log^2 n)$.

We will need the following variant of Chernoff bound to define the constant c_0.

Proposition 4.7 (Chernoff bound)**.** *There is a constant c_0 such that the following is true. Let $1 \leq d \leq n$ be integers, B be a set and $E \subseteq B$. Let us sample $c_0 \frac{|B|}{d} \log n$ samples from B independently at random with replacement.*

1. *If $|E| \geq d$ then the probability that less than $\frac{c_0}{2} \log n$ samples are from E is at most $1/n^{10}$.*
2. *If $|E| \leq d/4$ then the probability that at least $\frac{c_0}{2} \log n$ samples are from E is at most $1/n^{10}$.*

We are ready to establish the running time of the CoveringAlgorithm.

Proof of Theorem 4.6. The outer loop of the CoveringAlgorithm makes at most $\lceil \frac{4}{\theta} \rceil + 1$ iterations for different k's. The inner loop invokes DenseStripRemoval() and SparseStripExtensionSampling() on graphs $G(I,J)$ of horizontal and vertical size at most $4\theta n$. For each of the graphs it invokes the DenseStripRemoval() and SparseStripExtensionSampling() for $\lceil 1 + \log 1/\theta \rceil$ different values of ϵ.

Consider a call to DenseStripRemoval$(G, n, w_1, d/\epsilon, \epsilon/8, \epsilon)$ made by the inner loop of the algorithm, and let S and R be its output. Let T be the set maintained by the DenseStripRemoval() procedure. We say that a bad event happens if either some $I \in S$ is $(d/\epsilon, \epsilon/8, \epsilon)$-dense or if for some $I \in T$ such that $G(I, J_G)$ is $(d/4\epsilon, \epsilon/8, \epsilon)$-sparse, the else-branch of the if-statement of the main loop of the DenseStripRemoval() procedure is executed. By Proposition 4.7, the probability that the bad event happens during a particular call to DenseStripRemoval() is at most $n \cdot n^{-10} = n^{-9}$. There are at most $\lceil \frac{4}{\theta} + 1 \rceil \cdot \lceil 1 + \log \frac{1}{\theta} \rceil \in O(n \log n)$ calls to the DenseStripRemoval() procedure executed by the inner loop of the algorithm. Hence, the probability that a bad event happens during the execution of the algorithm is at most $O(n \log n) \cdot n^{-9} \leq n^{-7}$, for n large enough.

From now on we will analyze the running time of the algorithm assuming no bad event happened.

Consider a call to DenseStripRemoval$(G, n, w_1, d/\epsilon, \epsilon/8, \epsilon)$ made by the algorithm. Let B and T be as defined in the procedure. Since $|I_G| = |J_G| \leq 4\theta n$, $|B| \leq \frac{4\theta n}{\epsilon w_1/8} = \frac{32\theta n}{\epsilon w_1}$ during the call. We claim that the else-branch of the if-statement of the main loop of the DenseStripRemoval() procedure is executed at most $\frac{128\theta n}{d w_1}$ times during the call. Let $I_1, I_2, \dots, I_\ell$ be the I's for which the else-branch is executed. For $i = 1, \dots, \ell$, let $B_i = \{J \in B;\ \Delta_{\text{edit}}(x_{I_i}, y_J) \leq \epsilon\}$.

Since no bad event happened, $|B_i| \geq d/4\epsilon$. Moreover, for $i \neq j$, $B_i \cap B_j = \emptyset$. If there were $i < j$ and $J \in B_i \cap B_j$ then by the triangle inequality $\Delta_{\text{edit}}(x_{I_i}, x_{I_j}) \leq 2\epsilon$ so I_j would be added to

X when executing the else-branch for I_i, and I_j would be removed from T at that time so it would never be considered by the while-loop. Hence, the number of executions of the else-branch is at most $\frac{32\theta n}{\epsilon w_1} / \frac{d}{4\epsilon} = \frac{128\theta n}{d w_1}$.

Hence, DenseStripRemoval$(G, n, w_1, d/\epsilon, \epsilon/8, \epsilon)$ makes at most $\frac{|I_G|}{w_1} \cdot c_0 \frac{|B|}{d} \log n \leq \frac{4\theta n}{w_1} \cdot \frac{32\theta n}{\epsilon w_1} \cdot \frac{\epsilon c_0}{d} \log n \leq \frac{128 c_0 \theta^2 n^2}{w_1^2 d} \log n$ checks for $\Delta_{\text{edit}}(x_I, y_J) \leq \epsilon$, at most $\frac{128\theta n}{d w_1} \cdot \frac{32\theta n}{\epsilon w_1} = \frac{4096\theta^2 n^2}{d\epsilon w_1^2}$ checks for $\Delta_{\text{edit}}(x_I, y_J) \leq 3\epsilon$, and at most $\frac{128\theta n}{d w_1} \cdot \frac{4\theta n}{w_1} = \frac{512\theta^2 n^2}{d w_1^2}$ checks for $\Delta_{\text{edit}}(x_I, x_{I'}) \leq 2\epsilon$.

Thus the running time of DenseStripRemoval$(G, n, w_1, d/\epsilon, \epsilon/8, \epsilon)$ is bounded by $O(\frac{\theta^2 n^2 \log n}{d\epsilon w_1^2} \cdot t(w_1, \epsilon))$.

Consider a call to SparseStripExtensionSampling$(G, S, n, w_1, w_2, d/\epsilon, \epsilon/8, \epsilon, \theta)$ made by the algorithm. Assuming that no bad event happened in DenseStripRemoval(), S contains only $(d/\epsilon, \epsilon/8, \epsilon)$-sparse intervals. Hence, the procedure executes at most $c_1 \log^2 n \cdot \frac{4\theta n}{w_2} \cdot \frac{32\theta n}{\epsilon w_1}$ checks $\Delta_{\text{edit}}(x_I, y_J) \leq \epsilon$, where $\mu(I) = \mu(J) = w_1$, and $c_1 \log^2 n \cdot \frac{4\theta n}{w_2} \cdot \frac{d}{\epsilon}$ checks $\Delta_{\text{edit}}(x_{I'}, y_{J'}) \leq \epsilon$, where $|I'| = |J'| = w_2$. For each execution of $\Delta_{\text{edit}}(x_{I'}, y_{J'}) \leq \epsilon$, the procedure outputs $\log n$ estimates on the cost of the w_2-box. Notice, $\log n \leq w_2 \leq t(w_2, \epsilon)$. Thus the total cost of the call is $O(\frac{\theta^2 n^2 \log^2 n}{w_1 w_2 \epsilon} \cdot t(w_1, \epsilon) + \frac{\theta n d \log^2 n}{w_2 \epsilon} \cdot t(w_2, \epsilon))$.

For a given value of ϵ, DenseStripRemoval() and SparseStripExtensionSampling() are executed for $\lceil \frac{4}{\theta} + 1 \rceil$ different graphs of size at most $4\theta n$. Summing over all those invocations we obtain the overall time bound.

The number of certified w_2-boxes output by the algorithm is bounded by $(\log n)$-times the number of possible certified w_1-boxes the algorithm could output. The number of possible certified w_1-boxes output by a single invocation of DenseStripRemoval$(G, n, w_1, d/\epsilon, \epsilon/8, \epsilon)$ is at most $\frac{32\theta n}{\epsilon w_1} \cdot \frac{4\theta n}{w_1}$ by the above bound on $|B|$ and the size of G. By the bounds on ϵ, this is $O(\frac{\theta n^2}{w_1^2})$. Summing over all possible G and ϵ we get the desired bound. □

5 Shortest Path in a Semi-Grid Graph with Shortcuts

In this section we consider a special "grid-type" graph and compute the shortest distance in that graph. For intervals $I, J \subseteq \mathbb{N}$, a *semi-grid graph* G on $I \times J$ is a directed weighted graph with the set of vertices $V = I \times J$, where for each $(i, j) \in V$ there is a *horizontal* edge $(i, j) \to (i+1, j)$ of cost 2 if $i + 1 \in I$, and there is a *vertical* edge $(i, j) \to (i, j+1)$ of cost zero if $j + 1 \in J$. Note that as opposed to the grid graphs, there are no diagonal edges in our semi-grid graph. The *source* of G, denoted by source(G), is $(\min I, \min J)$, and the *sink* of G, denoted by sink(G), is $(\max I, \max J)$. We use the notation $V(G)$ and $E(G)$ to denote the set of vertices and the set of edges of graph G respectively. For any vertex $p \in V$, $x(p)$ and $y(p)$ denote the x-coordinate and the y-coordinate value of p respectively. We also define a partial ordering among the vertices as follows: for any two $p, q \in V$ we say that $p \preceq q$ if and only if $x(p) \leq x(q)$ and $y(p) \leq y(q)$.

Suppose we are given the semi-grid graph G on $\{0, \ldots, n\} \times \{0, \ldots, n\}$ and a set of m extra weighted edges $F = \{e_1, \ldots, e_m\}$ where e_i is an edge $p_i \to q_i$ for $p_i, q_i \in V(G)$ and $p_i \preceq q_i$. Each edge in the set F acts as a shortcut in the semi-grid graph G. The weight or cost of edges in F is defined by a function wt : $F \to \mathbb{R}$. Next consider the graph $G' = (V(G), E(G) \cup F)$. We call G' a *semi-grid graph with shortcuts*. In this section our objective is to compute the shortest distance (defined next) from the source to the sink in the graph G'.

As opposed to the previous sections, throughout this section we focus on the absolute cost of paths. The absolute cost of a path τ in the graph G', denoted by $\mathrm{cost}^{abs}_{G'}(\tau)$, is the sum of the costs of its edges. Note that in terms of relative cost, $\mathrm{cost}^{abs}_{G'}(\tau) = \mathrm{cost}_{G'}(\tau)\mu(\tau)$. We use the notation $\mathrm{dist}_{G'}(u, v)$ to denote the absolute cost of the cheapest path[2] from a vertex u to v in the graph G', i.e.,

$$\mathrm{dist}_{G'}(u, v) = \min_{u \text{ to } v \text{ path } \tau \text{ in } G'} \mathrm{cost}^{abs}_{G'}(\tau).$$

We also refer this by distance between u and v in G'. When u is the source of G' we omit u from the notation and simply use $\mathrm{dist}_{G'}(v)$ to denote the distance from source(G') to v. Clearly, $\mathrm{dist}_{G'}(\mathrm{sink}(G')) = \mathrm{cost}(G')\mu(G')$.

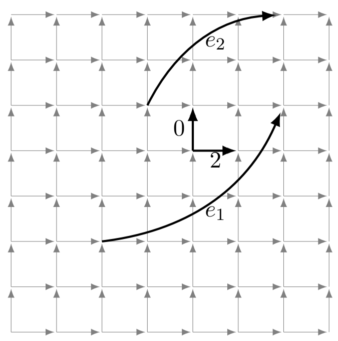


Figure 7: An example of a semi-grid graph with shortcuts given by $F = \{e_1, e_2\}$.

Theorem 5.1. *There is an algorithm that given the semi-grid graph G on $\{0, \dots, n\} \times \{0, \dots, n\}$, an edge set $F = \{e_1, \dots, e_m\}$ where each e_i is an edge $p_i \to q_i$ for $p_i, q_i \in V(G)$ and $p_i \preceq q_i$, and a cost function $wt : F \to \mathbb{R}$, outputs the distance from the source to the sink in the graph $G' = (V(G), E(G) \cup F)$. Moreover the algorithm runs in time $O((m + n)\log(m + n))$.*

Data structure. Throughout its run, the algorithm maintains a binary tree data structure such that at the end of each iteration $t = 0, 1, \dots, n$ the operation Query(t, k) outputs the shortest distance from the source of G' to the vertex (t, k). The data structure consists of a rooted binary tree T with $n+1$ leaves denoted by $\ell_0, \ell_1, \dots, \ell_n$ and numbered by $0, 1, \dots, n$ from the left to right. At any point of time, every node v of T stores a pair of values (t_v, d_v) where $t_v \in \{0, 1, \dots, n\}$ and $d_v \in \mathbb{R}$. For a leaf ℓ_i, its value (t_{ℓ_i}, d_{ℓ_i}) at iteration $t \geq t_{\ell_i}$ means that the node (t, i) is at distance $d_{\ell_i} + 2(t - t_{\ell_i})$ from the source of G'. (The value t_{ℓ_i} corresponds to the time when the leaf value was updated last time.) For an internal node v with leaves $\ell_i, \ell_{i+1}, \dots, \ell_j$ in its subtree, the values (t_v, d_v) means that at time $t \geq t_v$, $d_v + 2(t - t_v)$ is the minimum of the distances from source to the vertices $(t, i), (t, i + 1), \dots, (t, j)$. For any node v in T, left-child(v) and right-child(v) denote the left and right child of v respectively.

The data structure implements the following two operations:

[2]Note, any cheapest path with respect to the absolute cost function $\mathrm{cost}^{abs}_{G'}()$ is also a cheapest path with respect to the relative cost function $\mathrm{cost}_{G'}()$.

- Update$(i, (t, d))$: Given any $i \in \{0, \ldots, n\}$ and a pair $(t, d) \in \{0, \cdots, n\} \times \mathbb{R}$, it first finds the path P from the root to the leaf ℓ_i in T. Then it updates the value stored in every $v \in P$ to (t, d) whenever $d < d_v + 2(t - t_v)$.
- Query$((t, i))$: Given a point $(t, i) \in \{0, \ldots, n\} \times \{0, \ldots, n\}$, it first finds the path P from the root to the leaf ℓ_i in T. Let $R = \{\text{left-child}(v),\ v \in P \text{ and left-child}(v) \notin P\} \cup \{\ell_i\}$. Then it outputs the value $\min_{v \in R}(d_v + 2(t - t_v))$.

We would like to emphasize that the operation Query(p) stands for finding the value of $\text{dist}_{G'}(p)$, i.e., the distance of the point p from the source in G'.

Description of the algorithm. The algorithm maintains the data structure described above. Along with the data structure T we also maintain lists $L_0, \ldots, L_n$ which are initially empty. Each list L_t for $t \in \{0, \ldots, n\}$, contains entries that are used to update the data structure T during the time step t. Each entry (c, d) in a list L_t represents the fact that there is a path from the source to the vertex (t, c) in G' with absolute cost d.

We start the algorithm by initializing the content of each node of the binary tree data structure T to the value $(0, 0)$. Our algorithm (ApproxShortestPath()) runs in $n + 1$ steps. The algorithm first sorts the edges $e_i = (p_i \to q_i) \in F$ in the increasing order of the value of $x(p_i)$. This sorted list will help us to efficiently access the edges of F in the increasing order of the x-coordinate values of their starting vertices. At any time step t, the algorithm first performs Update() operations on T using the entries of the list L_t. Then the algorithm processes all the edges $e = (p \to q) \in F$ such that $x(p) = t$. For each such edge, the algorithm first uses the procedure Query(p) to determine the distance of the vertex p from the source. Then the algorithm adds an entry $(y(q), \text{Query}(p) + \text{wt}(e))$ to the list $L_{x(q)}$. Note that we can reach the point q from the point p while paying wt(e) extra cost and this justifies the value of the second coordinate.

Time complexity. First observe that since the depth of T is $\lceil \log n \rceil$, each of the operations Update() and Query() takes only $O(\log n)$ time. Since $|F| = m$, sorting the set F takes $O(m \log m)$ time. Next we claim that at the end of the algorithm, $\sum_{i \in [n]} |L_i| \leq m$. Note that throughout the run of the algorithm we never delete any element from any L_i. Observe that for each $e \in F$ there is exactly one entry in $\bigcup_{i \in [n]} L_i$. The claim now follows.

At any time instance $t \in \{0, 1, \ldots, n\}$, we first call the Update() procedure for each element in the list L_t. So this step takes $O(|L_t| \log n)$ time. Before adding any particular element in the list L_i for some $i \in [n]$, we call the procedure Query(). Hence the total time taken by the algorithm is bounded by

$$O(m \log m) + \sum_{i=0}^{n} O(|L_i| \log n) \leq O((m + n) \log(m + n)).$$

Correctness of the algorithm. For any vertex $v \in V(G)$ we use the notation $\text{dist}_{G'}(v)$ to denote the shortest distance of the vertex v from the source in the graph G'.

Lemma 5.2. *At the end of any time step $t \in \{0, \ldots, n\}$, for any $k \in \{0, \ldots, n\}$*

$$\mathit{Query}((t, k)) = \mathit{dist}_{G'}((t, k)).$$

Algorithm: ApproxShortestPath(G, F, wt)

Input: A semi-grid graph G on $\{0,\ldots,n\} \times \{0,\ldots,n\}$, an edge set $F = \{e_1,\ldots,e_m\}$ where e_i is an edge $p_i \to q_i$ for $p_i, q_i \in V(G)$ and $p_i \preceq q_i$, and a cost function $\mathrm{wt} : F \to \mathbb{R}$
Output: The shortest distance from the source to the sink in the graph $G' = (V(G), E(G) \cup F)$.

```
Initialization: Build a complete binary tree $T$ having $n+1$ leaves numbered $0,\ldots,n$, and
store $(0,0)$ to all the nodes of $T$.
Set $L_0 = L_1 = L_2 = \cdots = L_n = \emptyset$.
Sort the elements of the set $F$ in the increasing order of the value of $x(p_i)$ and denote the
sorted list as $S$;
for $t = 0, 1, \ldots, n$ do
    forall the $(j, d) \in L_t$ do
        Update$(j, (t, d))$;
    end
    forall the $e \in S$ such that $e = p \to q$ and $x(p) = t$ do
        Add $(y(q), \mathrm{Query}(p) + \mathrm{wt}(e))$ to the list $L_{x(q)}$;
        Remove $e$ from $S$;
    end
end
Output Query$((n, n))$;
```

The correctness of the algorithm now follows from the above lemma. Before proving the lemma let us make the following simple observation.

Proposition 5.3. *At the end of any time step $t \in \{0, 1, \ldots, n\}$, for any internal node v in T, suppose (t_1, d_1) and (t_2, d_2) are the tuples stored in its two children. Then if $d_1 + 2(t - t_1) \leq d_2 + 2(t - t_2)$, v stores the pair (t_1, d_1); else v stores the pair (t_2, d_2).*

Proof. We prove the proposition using induction on t. As a base case, when $t = 0$ by the initialization step of the algorithm all the nodes store $(0,0)$ and thus the proposition is trivially true. For the induction step, suppose the proposition is true for the time step $0, \ldots, t-1$. Then we want to prove it for the time step t.

Note that for some internal node v if the value of it and its children have not been updated during the time step t, then clearly the claim is still true for that node v at the time step t. Suppose for an internal node v, the value stored at left-child(v) has been updated at t. Since the update is only possible because of the invocation of the procedure Update(), the value at v has also been updated. The proposition now follows from the definition of the Update() operation. The case when the values at right-child(v) or both at left-child(v) and right-child(v) have been updated are analogous. ☐

Proof of Lemma 5.2. We prove the lemma by using induction on t and k.

(Outer) Base case $t = 0$: due to the initialization step the lemma is trivially true.

(Outer) Induction step: Assume that the lemma is true for the time step $0, \ldots, t-1$ and we want to prove it for the time step t. Now to prove the lemma for the time step t, we use induction on

$k \in \{0, 1, \ldots, n\}$. Note that here we are using induction argument twice, where the outer induction is on t and the inner induction is on k.

(Inner) Base case $k = 0$: We want to show that Query$((t, 0)) = \text{dist}_{G'}((t, 0))$. Let τ be a path with the cheapest absolute cost, from the source to the vertex $(t, 0)$. Suppose the last vertex that appears in τ just before $(t, 0)$ is $(t', 0)$. By the definition of the graph G and the edge set F, $t' < t$. Assume that the edge $e = (t', 0) \to (t, 0)$ is in the set F; otherwise the statement is clearly true. By the description of the algorithm during step t' while processing the edge e, we add an entry in the list L_t. By the (outer) induction hypothesis we know that at time step t', Query$((t', 0)) = \text{dist}_{G'}((t', 0))$. Hence the entry added in L_t is $(0, \text{dist}_{G'}((t', 0)) + \text{wt}(e))$. Now by the definition of the procedure Update() and Query(), at the end of time step t,

$$\begin{aligned}\text{Query}((t, 0)) &= \text{dist}_{G'}((t', 0)) + \text{wt}(e)\\ &= \text{dist}_{G'}((t, 0)).\end{aligned}$$

(Inner) Induction step: Let us now assume that the hypothesis is true for $0, 1, \ldots, k-1$ and we want to prove it for k. So we want to show that Query$((t, k)) = \text{dist}_{G'}((t, k))$. Let τ be a path with the cheapest absolute cost, from the source to the vertex (t, k). We analyze the following two possibilities.

Case 1. The path τ passes through the vertex $(t, k-1)$. So, $\text{dist}_{G'}((t, k)) = \text{dist}_{G'}((t, k-1))$. By the (inner) induction hypothesis, Query$((t, k-1)) = \text{dist}_{G'}((t, k-1))$. Let P_k be the path from the root to the leaf ℓ_k and P_{k-1} be the path from the root to the leaf ℓ_{k-1} in T. Further let v be the lowest common ancestor of the leaves ℓ_{k-1} and ℓ_k. Let $u = \text{left-child}(v)$ and note that $u \in P_{k-1}$. Let us denote the set R in the definition of Query() by R_{k-1} and R_k for the queries Query$((t, k-1))$ and Query$((t, k))$ respectively. Let T_u be the subtree rooted at u. By the repeated application of Proposition 5.3,

$$d_u + 2(t - t_u) = \min_{s \in T_u}(d_s + 2(t - t_s)).$$

Hence from the definition of the procedure Query() it follows that

$$\text{Query}((t, k-1)) = \min\{d_u + 2(t - t_u), \min_{r \in R_k \cap R_{k-1}}(d_r + 2(t - t_r))\}.$$

Now by the definition of a binary tree observe that either $u = \ell_{k-1}$ or $R_k \setminus R_{k-1} = \{u, \ell_k\}$. So from the definition of the procedure Query() we can write that

$$\begin{aligned}\text{Query}((t, k)) &= \min\{\text{Query}((t, k-1)), d_{\ell_k} + 2(t - t_{\ell_k})\}\\ &\leq \text{Query}((t, k-1))\\ &= \text{dist}_{G'}((t, k-1))\\ &= \text{dist}_{G'}((t, k)).\end{aligned}$$

Next we will show that Query$((t, k-1)) \leq d_{\ell_k} + 2(t - t_{\ell_k})$, which together with the above equation implies that Query$((t, k)) = \text{dist}_{G'}((t, k))$. For the sake of contradiction let Query$((t, k-1)) > d_{\ell_k} + 2(t - t_{\ell_k})$. By the definition of Update() operation, the content of the node ℓ_k was last updated at the time step t_{ℓ_k}. Now by the (outer) induction hypothesis at the end of the time step t_{ℓ_k}

$$\begin{aligned}\text{dist}_{G'}((t_{\ell_k}, k)) &= \text{Query}((t_{\ell_k}, k))\\ &\leq d_{\ell_k}\end{aligned}$$

where the last inequality follows from the definition of the procedure Query(). Now consider a path that follows any cheapest path from the source to the vertex (t_{ℓ_k}, k) and then takes the horizontal edges till the vertex (t, k). Thus $\text{dist}_{G'}((t,k)) \leq d_{\ell_k} + 2(t - t_{\ell_k})$. Since $\text{dist}_{G'}((t,k)) = dist_{G'}((t, k-1))$, we get a contradiction on our assumption that $\text{Query}((t, k-1)) > d_{\ell_k} + 2(t - t_{\ell_k})$.

Case 2. Suppose τ takes a path from the source to a vertex p with $x(p) < t$ and then takes an edge e to a vertex (t, k). Note that the path τ till the vertex p must be a cheapest path from the source to the vertex p.

1. If e is an horizontal edge, i.e., $x(p) = t - 1$ then $\text{dist}_{G'}((t,k)) = \text{dist}_{G'}((t-1,k)) + 2$. By the (outer) induction hypothesis we know that at the end of time step $t-1$, $\text{Query}((t-1,k)) = \text{dist}_{G'}((t-1,k))$. Now at the end of time step t, by the definition of Query() operation clearly $\text{Query}((t,k)) \leq \text{dist}_{G'}((t-1,k)) + 2$. By an argument similar to that used in case 1 we can also show that $\text{Query}((t,k)) \geq \text{dist}_{G'}((t-1,k)) + 2$, which establishes the hypothesis.

2. The only other option is that $e \in F$. While processing the edge e during the time step $x(p)$, by the (outer) induction hypothesis $\text{Query}(p) = \text{dist}_{G'}(p)$. So the corresponding entry added in the list L_t is $(k, \text{dist}_{G'}(p) + \text{wt}(e))$. Further note that $\text{dist}_{G'}((t,k)) = \text{dist}_{G'}(p) + \text{wt}(e)$. Now the hypothesis follows from an argument similar to that used in case 1.

□

6 Reduction from the Cost of a Gird Graph to that of a Semi-Grid Graph with Shortcuts

In this section we present a reduction that takes the output of the CoveringAlgorithm and transforms it into an instance of determining the shortest path in a semi-grid graph with shortcuts.

Theorem 6.1. *There is an algorithm that given a set $R = \{(I_1, J_1, \epsilon_1), \dots, (I_m, J_m, \epsilon_m)\}$ where all I_i, J_i are intervals from $\{0, \dots, n\}$, $\mu(I_i) = \mu(J_i)$, and $\epsilon_i \in [0,1]$, outputs a set of shortcut edges F on the vertex set $V = \{0, \dots, n\} \times \{0, \dots, n\}$, of size $4m$ and a cost function $wt : F \to \mathbb{R}$ such that:*

- *If G is a grid graph on V so that for each $i \in [m]$, (I_i, J_i, ϵ_i) is a certified box (i.e., $cost(G(I_i, J_i)) \leq \epsilon_i$), and H' is the semi-grid graph on vertices V with shortcuts given by F and wt,*
- *Then for any path τ in G with cost at most $\epsilon \in [0,1]$ for which there is a subset $S \subseteq R$ that $(1-\delta, k, \theta)$-approximates τ:*

$$cost(G) \leq cost(H') \leq 5(k\epsilon + \theta) + 2\delta.$$

Furthermore, the algorithm runs in time $O(m)$.

We devote rest of the section to prove the above theorem. Throughout this proof we ignore rounding issues. If for some $i \in [m]$ $\epsilon_i w_i$ is not an integer then we will just replace the term $\epsilon_i w_i$ by $\lfloor \epsilon_i w_i \rfloor$ in the proof, and proceed with the argument.

Proof. The set F and the cost function $\text{wt} : F \to \mathbb{R}$ is defined as follows: for each $i \in [m]$ we set $w_i = \mu(I_i)$ and define four edges

- $e_{i,1} = ((\min I_i + \epsilon_i w_i, \min J_i) \to (\max I_i, \max J_i - \epsilon_i w_i))$, and set $\text{wt}(e_{i,1}) = 3\epsilon_i w_i$,

- $e_{i,2} = ((\min I_i + \epsilon_i w_i, \min J_i) \to (\max I_i - \epsilon_i w_i, \max J_i))$, and set $\mathrm{wt}(e_{i,2}) = \epsilon_i w_i$,
- $e_{i,3} = ((\min I_i, \min J_i + \epsilon_i w_i) \to (\max I_i, \max J_i - \epsilon_i w_i))$, and set $\mathrm{wt}(e_{i,3}) = 5\epsilon_i w_i$,
- $e_{i,4} = ((\min I_i, \min J_i + \epsilon_i w_i) \to (\max I_i - \epsilon_i w_i, \max J_i))$, and set $\mathrm{wt}(e_{i,4}) = 3\epsilon_i w_i$.

See Fig. 8 for a pictorial view of the four edges.

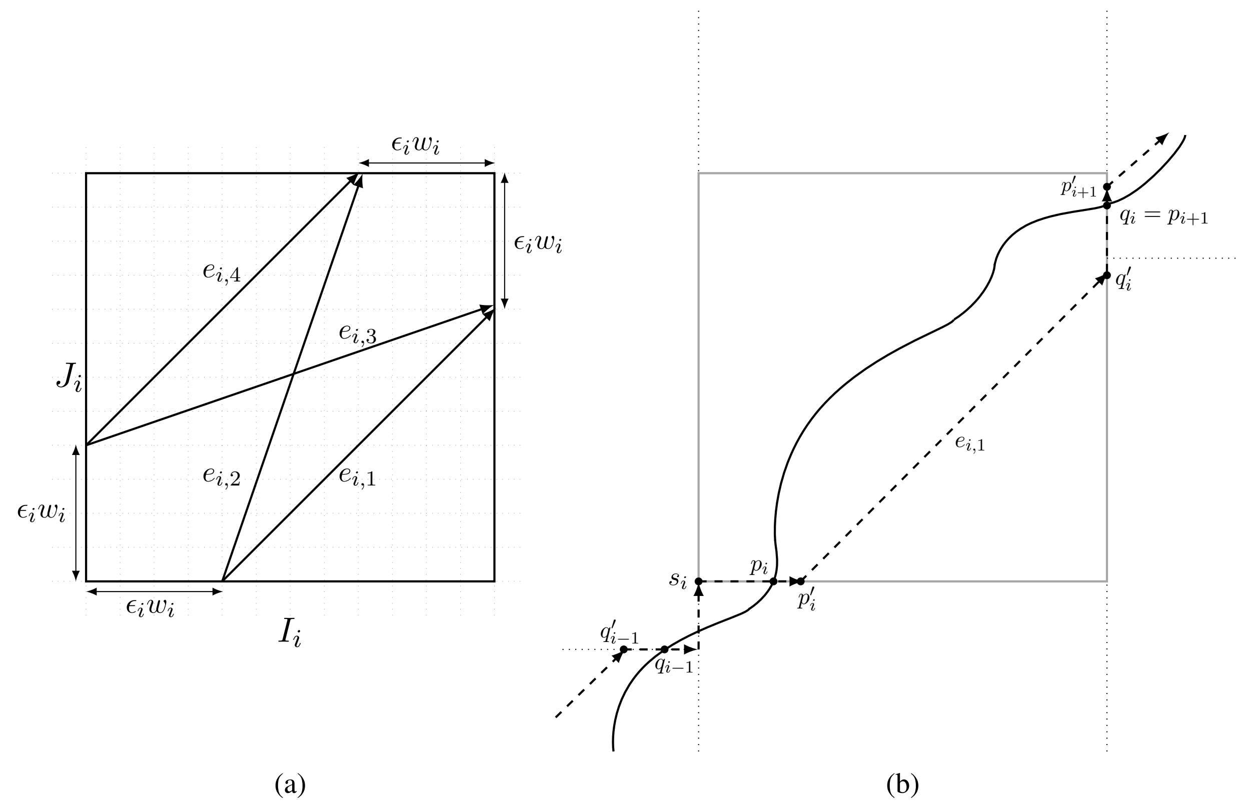


Figure 8: (a) The four edges added in F for each block G_i. (b) An example of a path τ (in solid) passing through a block G_i. The dashed path is an approximation τ' of τ in H'. Weight of the subpath of τ' from s_i to p'_i is $2\epsilon_i w_i$, $\mathrm{wt}(e_{i,1}) = 3\epsilon_i w_i$, and weight of the subpath of τ' from q'_i to p'_{i+1} is 0. So the total weight of the subpath of τ' from s_i to p'_{i+1} is $5\epsilon_i w_i$.

We let $F = \{e_{i,j},\, i \in [m], j \in [4]\}$. Now take the semi-grid graph H on $\{0,\dots,n\} \times \{0,\dots,n\}$ and consider the graph $H' = (V(H), E(H) \cup F)$ where $V(H)$ and $E(H)$ are the set of vertices and the set of edges of H respectively. Soon we will justify the cost function.

Recall that we are given a grid graph G on $\{0,\dots,n\} \times \{0,\dots,n\}$. τ is a path from the source to the sink in G with $\mathrm{cost}(\tau) \leq \epsilon$. $R = \{(I_1, J_1, \epsilon_1), \dots, (I_m, J_m, \epsilon_m)\}$ is a set of certified boxes of

G, where for each $i \in [m]$, $G_i = G(I_i, J_i)$ is a w_i-box. Let us now consider the subset $S \subseteq R$ that $(1-\delta, k, \theta)$-approximates τ. For the sake of simplicity assume that $S = \{(I_1, J_1, \epsilon_1), \dots, (I_s, J_s, \epsilon_s)\}$ for some $s \in [m]$. We further assume that $\max I_1 \leq \max I_2 \leq \dots \leq \max I_s$. Recall from the definition of $(1-\delta, k, \theta)$-approximation that S satisfies the following properties:

1. For all $i, j \in [s]$, $|I_i \cap I_j| \leq 1$.
2. $\sum_{i \in [s]} \mu(I_i) \geq (1-\delta)n$.
3. For each $i \in [s]$, $G_i = G(I_i, J_i)$ $(1-\epsilon_i)$-covers τ and is of cost at most ϵ_i.
4. $\sum_{i \in [s]} \epsilon_i \mu(I_i) \leq (k\epsilon + \theta)n$

For the sake of the analysis we build an auxiliary (weighted directed) graph K with $V(G)$ and $E(G)$ as the set of vertices and the set of edges respectively. The only difference between G and K will be the cost assigned to their edges. In K, for all the horizontal edges we assign the cost 2 and for all the vertical edges we assign the cost zero (this graph K was previously considered in [21]). For all the remaining edges, i.e., diagonal edges, we assign the cost to be as in G.

Proposition 6.2. *For any interval $I, J \subseteq \{0, \dots, n\}$ such that $\mu(I) = \mu(J)$, let σ be any source to sink path in the subgraph $G(I, J)$. Then $cost_G(\sigma) = cost_K(\sigma)$.*

In the above statement we use the notations $\text{cost}_G(\sigma)$ and $\text{cost}_K(\sigma)$ to denote the cost of the path σ in the graph G and K respectively. Note that since both the graphs G and K are defined on the same set of vertices and edges, the path σ belongs to both of them.

Proof. It must be the case that the number of horizontal edges taken by σ equals the number of vertical edges in σ otherwise σ would not have reached the sink from source. The claim follows as the cost of a single vertical edge plus a single horizontal edge is the same in both graphs. $\square$

As a corollary of the above we derive the following.

Corollary 6.3. *For any interval $I, J \subseteq \{0, \dots, n\}$ with $\mu(I) = \mu(J)$, $cost(G(I, J)) = cost(K(I, J))$.*

The following proposition justifies the cost function wt defined on the edge set F. We need this proposition to prove Claim 6.6.

Proposition 6.4. *Suppose $\tilde{G} = G(I, J)$ is a w-box. Then for any $h \in [w]$,*

1. *There is a path σ' from the vertex $(\min I + h, \min J)$ to the vertex $(\max I, \max J - h)$ such that $cost_K(\sigma') \leq cost(\tilde{G}) + 2h/w$;*
2. *There is a path σ' from the vertex $(\min I + h, \min J)$ to the vertex $(\max I - h, \max J)$ such that $cost_K(\sigma') \leq cost(\tilde{G})$;*
3. *There is a path σ' from the vertex $(\min I, \min J + h)$ to the vertex $(\max I, \max J - h)$ such that $cost_K(\sigma') \leq cost(\tilde{G}) + 4h/w$;*
4. *There is a path σ' from the vertex $(\min I, \min J + h)$ to the vertex $(\max I - h, \max J)$ such that $cost_K(\sigma') \leq cost(\tilde{G}) + 2h/w$.*

Proof. Here we provide the proof for the first item. The proofs for the remaining items are similar.

Let $\tilde{K} = K(I, J)$. Let σ be a cheapest path from the source to the sink in $\tilde{G}$. So $\mathrm{cost}_G(\sigma) = \mathrm{cost}(\tilde{G})$. Suppose the path σ passes through the vertices $(\min I + h, j)$ and $(i, \max J - h)$ in the subgraph $\tilde{G}$. We construct a new path σ' that first takes vertical edges from $(\min I + h, \min J)$ to $(\min I + h, j)$ and then follows σ till the vertex $(i, \max J - h)$, and finally takes horizontal edges till the vertex $(\max I, \max J - h)$. If $\max I - i \leq h$, then clearly

$$\begin{aligned}\mathrm{cost}_K(\sigma') &\leq \mathrm{cost}(\tilde{K}) + 2h/w \\ &= \mathrm{cost}(\tilde{G}) + 2h/w\end{aligned}$$

where the last equality follows from Corollary 6.3.

If $\max I - i > h$, then observe that after passing through the vertex $(i, \max J - h)$ the path σ must take $(\max I - i - h)$ horizontal edges; otherwise it would not have reached the sink. Hence the cost of the path σ in $\tilde{K}$ between the vertices $(\min I + h, j)$ and $(i, \max J - h)$ is at most $\mathrm{cost}(\tilde{K}) - 2(\max I - i - h)/w$. So

$$\begin{aligned}\mathrm{cost}_K(\sigma') &\leq ((\mathrm{cost}(\tilde{K})w - 2 \cdot (\max I - i - h))/w \\ &\quad + 0 \cdot (j - \min J)/w + 2 \cdot (\max I - i))/w \\ &\leq \mathrm{cost}(\tilde{K}) + 2h/w \\ &= \mathrm{cost}(\tilde{G}) + 2h/w\end{aligned}$$

where the last equality follows from Corollary 6.3. □

Claim 6.5. *There exists a source to sink path in H' of cost at most $5(k\epsilon + \theta) + 2\delta$.*

Proof. Recall that τ is a path from the source to the sink in G with $\mathrm{cost}(\tau) \leq \epsilon$. Fix any $i \in [s]$. Then consider the set S and the portion of the path τ that passes through the box G_i. Let the first and last vertex of τ in G_i be p_i and q_i respectively. Since G_i $(1-\epsilon_i)$ covers τ, both p_i and q_i are border vertices of G_i. Hence either $x(p_i) = \min I_i$ or $y(p_i) = \min J_i$ and similarly, either $x(q_i) = \max I_i$ or $y(q_i) = \max J_i$. Among the two vertices $(\min I_i + \epsilon_i w_i, \min J_i)$ and $(\min I_i, \min J_i + \epsilon_i w_i)$, let p'_i be the vertex closer to p_i, that is, if $x(p_i) = \min I_i$ then $p'_i = (\min I_i, \min J_i + \epsilon_i w_i)$, otherwise $p'_i = (\min I_i + \epsilon_i w_i, \min J_i)$. Similarly, among $(\max I_i - \epsilon_i w_i, \max J_i)$ and $(\max I_i, \max J_i - \epsilon_i w_i)$, let q'_i be the vertex closer to q_i. Note that for all $i \in [s]$, $p_i \preceq p'_i$ and $q'_i \preceq q_i$. Now we construct a path τ' in the graph H' as follows:

Starting from the source, take only horizontal and vertical edges to reach the vertex p'_1. Then take the edge $e_{1,j}$ for some $j \in [4]$ that goes from p'_1 to q'_1. Next again take only horizontal and vertical edges till p'_2. Then take the edge $e_{2,j}$ for some $j \in [4]$ that goes from p'_2 to q'_2. Proceed till we reach the sink of H' (see Fig. 8).

To show that the above procedure generates a path, we need to prove that for all $i \in [s-1]$ $q'_i \preceq p'_{i+1}$. This is trivially true because by the definition of the graph G, $q_i \preceq p_{i+1}$ and for all $i \in [s]$, $p_i \preceq p'_i$ and $q'_i \preceq q_i$.

Observe from the definition of the cost function wt() that, for any $i \in [s]$ absolute cost of the portion of the path τ' that passes through the subgraph $H'(I_i, \{0, \cdots, n\})$, is at most $5\epsilon_i w_i$ (see Fig. 8). Now since $\sum_{i \in [s]} \mu(I_i) \geq (1-\delta)n$, absolute cost of the portion of the path τ' that does not belong to $H'(I_i, \{0, \cdots, n\})$ for any $i \in [s]$, is at most $2\delta n$. So it follows that

$$\begin{aligned} \text{cost}_{H'}(\tau') &\leq \frac{1}{n}(\sum_{i\in[s]} 5\epsilon_i w_i + 2\delta n) \\ &\leq 5(k\epsilon + \theta) + 2\delta \end{aligned}$$

where the last inequality follows from the fourth property of the subset S. □

Claim 6.6. *Let σ be a cheapest path from the source to sink in H'. Then there exists a source to sink path in G of cost at most $cost_{H'}(\sigma)$.*

Proof. First observe that if the path σ does not contain any edge from the set F, then $\text{cost}(\sigma) = 2n$ and the claim is clearly true. So from now on we assume that σ contains some edge from F. Note that for each $i \in [m]$ a path in H' can contain at most one edge from the set $\{e_{i,j} | j \in [4]\}$. Now we construct a path σ' in the grid graph G as follows:

For each edge e in the path σ, if $e = (p \to q)$ is a horizontal or vertical edge then keep it as it is, i.e., include that edge in the path σ'. Otherwise if $e = e_{i,j} \in F$, we add a cheapest path from p to q (inside the subgraph G_i).

It is easy to see that the above construction produces a valid path from the source to sink in G. Note, by the definition of the graph K, σ' is also a source to sink path in K. Let $F' = \{e; e \in F \text{ and } e \text{ belongs to the path } \sigma\}$. For each edge $e \in F'$, let us denote the corresponding portion of the path in σ' as σ'_e. Note that by the definition of the cost function wt() and Proposition 6.4, for each edge $e \in F'$ if $e = e_{i,j}$ for some $i \in [m]$ and $j \in [4]$, then $\text{cost}_K(\sigma'_e) \leq \text{wt}(e)/w_i$. So it is straightforward to derive that $\text{cost}_K(\sigma') \leq \text{cost}_{H'}(\sigma)$. The claim now follows from Corollary 6.3. □

This concludes the proof of Theorem 6.1. □

7 Proof of Theorem 2.1

In this section we combine the ingredients obtained thus far to solve Gap-Edit Distance$(x, y, \theta, 251)$ as required for Theorem 2.1. We design the following algorithm:

On input strings x and y of length n, with the parameter $\theta \in [0, 1]$ proceed as follows. If $\theta \leq n^{-1/5}$, then run the $O(n + k^2)$-time edit distance algorithm of Landau *et al.* [17] on inputs x and y. (Here k denotes the edit distance of the strings.) If the algorithm finishes the computation of $\Delta_{\text{edit}}(x, y)$ in time $O(n + \theta^2 n^2)$ then output 1 iff $\Delta_{\text{edit}}(x, y) \leq \theta$, otherwise stop the algorithm and output 0.

If $\theta > n^{-1/5}$, we use our algorithm. Set w_1 and w_2 to be powers of two such that:

$$w_1 \in \left[\frac{1}{2} \cdot \theta^{-2/7} n^{1/7}, \theta^{-2/7} n^{1/7}\right] \qquad w_2 \in \left[\theta^{1/7} n^{3/7}, 2\theta^{1/7} n^{3/7}\right],$$

and set

$$d = \left\lceil \theta^{3/7} n^{2/7} \right\rceil.$$

Furthermore, shrink x and y to the nearest multiple of w_2 by cutting-off their last few symbols to obtain strings x' and y'. Set n' to be the length of x' and y'. Clearly, $n' = n(1 - O(\theta^{1/7} n^{-4/7}))$. By the choice of our parameters, $\log n' \leq w_1 \leq \theta w_2$, $w_2 \leq \theta n'/4$, $w_1 | w_2$ and $w_2 | n'$.

Thus we proceed as follows:

1. Invoke CoveringAlgorithm$(x', y', n', w_1, w_2, d, (1 + 1/500)\theta)$ to obtain a set R.
2. Apply the algorithm from Theorem 6.1 with the set R as input to construct a set of weighted edges F along with a cost function $\mathrm{wt} : F \to \mathbb{R}$.
3. Invoke ApproxShortestPath(H, F, wt) on the semi-grid graph H on $\{0, \dots, n'\} \times \{0, \dots, n'\}$ to obtain a value k.
4. If $k \le 250.5\theta n'$ then output 1 otherwise output 0.

Proof of correctness. To prove the correctness of the above algorithm, we claim that the following holds with probability at least $1 - n^{-8}$: If $\Delta_{\text{edit}}(x, y) \le \theta$, then $k \le 250.5\theta n'$, and if $\Delta_{\text{edit}}(x, y) > 251\theta$, then $k > 250.5\theta n'$.

First recall that $n' = n(1 - O(\theta^{1/7} n^{-4/7}))$. By the construction of x' and y' observe that $\Delta_{\text{edit}}(x', y')n' \le \Delta_{\text{edit}}(x, y)n$, which eventually implies that $\Delta_{\text{edit}}(x', y') \le (1 + 3n^{-4/7})\Delta_{\text{edit}}(x, y)$. Thus if $\Delta_{\text{edit}}(x, y) \le \theta$ then $\Delta_{\text{edit}}(x', y') \le (1 + 1/500)\theta$ for large enough n. Further notice that $\Delta_{\text{edit}}(x, y)n \le \Delta_{\text{edit}}(x', y')n' + (n - n')$. This implies that when $\theta > n^{-1/5}$ (which is indeed the case), if $\Delta_{\text{edit}}(x, y) > 251\theta$ then

$$\begin{aligned}\Delta_{\text{edit}}(x', y') &> 251(1 - \frac{4}{251}\theta^{-6/7} n^{-4/7})\theta \\ &> 251(1 - \frac{4}{251} n^{-2/5})\theta \\ &> 250.5\theta\end{aligned}$$

where the second inequality follows from the fact that $\theta > n^{-1/5}$ and the last inequality is true for large enough n. So to prove our claim it suffices to show that, if $\Delta_{\text{edit}}(x', y') \le (1 + 1/500)\theta$, then $k \le 250.5\theta n'$, and if $\Delta_{\text{edit}}(x', y') > 250.5\theta$, then $k > 250.5\theta n'$.

Take the edit distance graph of the two strings x' and y', denoted by $G = G_{x',y'}$. By Proposition 3.1 there is a source to sink path τ in G of cost $\Delta_{\text{edit}}(x', y')$. If $\Delta_{\text{edit}}(x', y') \le (1 + 1/500)\theta$, then by Theorem 4.1 the set R output by CoveringAlgorithm() contains a subset that $(1, 35, 15(1 + 1/500)\theta)$-approximates τ with probability at least $1 - n'^{-9}$. Now Theorem 6.1 ensures that, for the graph $H' = (V(H), E(H) \cup F)$, $\Delta_{\text{edit}}(x', y') \le \text{cost}(H')$, and if $\Delta_{\text{edit}}(x', y') \le (1+1/500)\theta$ then $\text{cost}(H') \le 250.5\theta$. Finally by Theorem 5.1 $\text{cost}(H') = k/n'$ (note, by Theorem 5.1 $\text{dist}_{H'}(\text{sink}(H')) = k$ and by the definition of distance function, $\text{cost}(H') = \text{dist}_{H'}(\text{sink}(H'))/n'$). The claim follows.

Time complexity. Next we bound the running time of our algorithm. Notice that CoveringAlgorithm() involves solving the following query: given two strings z_1 (a substring of x') and z_2 (a substring of y') each of length w for some $w \in [n']$, and $\epsilon \in (0, 1]$, "Is $\Delta_{\text{edit}}(z_1, z_2) \le \epsilon$?". Recall that we use $t(w, \epsilon)$ to denote the running time of an algorithm that solves the above query. By Theorem 4.6 CoveringAlgorithm$(x', y', n', w_1, w_2, d, (1 + 1/500)\theta)$ runs in time

$$O\left(\sum_{\substack{k=\lceil \log 250/\theta \rceil, \dots, 0 \\ \epsilon = 2^{-k}}} \frac{\theta n^2 \log n}{d\epsilon w_1^2} \cdot t(w_1, \epsilon) + \frac{\theta n^2 \log^2 n}{w_1 w_2 \epsilon} \cdot t(w_1, \epsilon) + \frac{n d \log^2 n}{w_2 \epsilon} \cdot t(w_2, \epsilon)\right)$$

with probability at least $1 - n'^{-7}$. We use the algorithm by Ukkonen [22] to solve the query "Is $\Delta_{\text{edit}}(z_1, z_2) \le \epsilon$?", and thus we get $t(w, \epsilon) = O(\epsilon w^2)$. Hence for our choice of parameters

w_1, w_2 and d, the running time of CoveringAlgorithm$(x', y', n', w_1, w_2, d, (1 + 1/500)\theta)$ is bounded by $\widetilde{O}(n^{12/7}\theta^{4/7})$ with probability at least $1 - n'^{-7}$. Furthermore, it outputs the set R of size at most $\widetilde{O}(n^{12/7}\theta^{4/7})$. By the bound on the size of the set R it follows that the algorithm referred in Theorem 6.1 runs in time $\widetilde{O}(n^{12/7}\theta^{4/7})$. Moreover, $|F| \leq \widetilde{O}(n^{12/7}\theta^{4/7})$. Hence by Theorem 5.1, the procedure ApproxShortestPath() also runs in time $\widetilde{O}(n^{12/7}\theta^{4/7})$ and this concludes the proof.

References

[1] Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 11:1–11:26, 2017.

[2] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. URL: `https://doi.org/10.1109/FOCS.2015.14`, `doi:10.1109/FOCS.2015.14`.

[3] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388, 2016. URL: `http://doi.acm.org/10.1145/2897518.2897653`, `doi:10.1145/2897518.2897653`.

[4] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: Or: a polylog shaved is a lower bound made. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 375–388, New York, NY, USA, 2016. ACM.

[5] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386, 2010.

[6] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 199–204, New York, NY, USA, 2009. ACM.

[7] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM.

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

[9] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 316–324, New York, NY, USA, 2003. ACM.

[10] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 792–801, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[11] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed 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, New Orleans, LA, USA, January 7-10, 2018*, pages 1170–1189, 2018.

[12] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015. URL: `https://doi.org/10.1109/FOCS.2015.15`, `doi:10.1109/FOCS.2015.15`.

[13] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for computing edit distance without exploiting suffix trees. *CoRR*, abs/1607.03718, 2016. `arXiv:1607.03718`.

[14] Diptarka Chakraborty, Elazar Goldenberg, and Michal 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, Cambridge, MA, USA, June 18-21, 2016*, pages 712–725, 2016.

[15] Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016. URL: `https://doi.org/10.1016/j.dam.2015.10.040`, `doi:10.1016/j.dam.2015.10.040`.

[16] Robert Krauthgamer and Yuval Rabani. Improved lower bounds for embeddings into L_1. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 1010–1017, 2006.

[17] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, April 1998.

[18] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[19] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18 – 31, 1980.

[20] Timothy Naumovitz, Michael E. Saks, and C. Seshadhri. Accurate and nearly optimal sublinear approximations to ulam distance. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2012–2031, 2017.

[21] Dimitrios P. Papamichail and Georgios P. Papamichail. Improved algorithms for approximate string matching (extended abstract). *BMC Bioinformatics*, 10:S10 – S10, 2009.

[22] Esko Ukkonen. Algorithms for approximate string matching. *Inf. Control*, 64(1-3):100–118, March 1985.

[23] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.