Constant factor approximations to edit distance on far input pairs in nearly linear time

Michal Koucký*[1] and Michael Saks†[2]

[1]Computer Science Institute of Charles University, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
[2]Department of Mathematics, Rutgers University, Piscataway, NJ, USA

Abstract

For any $T \geq 1$, there are constants $R = R(T) \geq 1$ and $\zeta = \zeta(T) > 0$ and a randomized algorithm that takes as input an integer n and two strings x, y of length at most n, and runs in time $O(n^{1+\frac{1}{T}})$ and outputs an upper bound U on the edit distance of $d_{\text{edit}}(x, y)$ that with high probability, satisfies $U \leq R(d_{\text{edit}}(x, y) + n^{1-\zeta})$. In particular, on any input with $d_{\text{edit}}(x, y) \geq n^{1-\zeta}$ the algorithm outputs a constant factor approximation with high probability. A similar result has been proven independently by Brakensiek and Rubinstein [13].

*Email: koucky@iuuk.mff.cuni.cz. 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. Partially supported by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X.

†Email: msaks30@gmail.com. Supported in part by Simons Foundation under award 332622.

1 Introduction

The *edit distance* (or *Levenshtein distance*) [21] between strings x, y, denoted by $d_{\text{edit}}(x, y)$, is the minimum number of character insertions, deletions, and substitutions needed to convert x into y. It has numerous applications in various fields from text processing to bioinformatics so algorithms for edit distance computation attract lot of attention. It was recently shown independently that edit distance can be approximated within a constant factor in truly subquadratic time in the quantum computation model [11, 12], and in the classical model [15, 16]. The running time for a classical algorithm obtained in [15, 16] is $\widetilde{O}(n^{12/7})$, which was improved by Andoni [4] to $O(n^{3/2+\epsilon})$.

This raises the natural question: what is the best possible running time of a constant factor approximation classical algorithm. We make progress on this problem by developing a nearly linear time algorithm that gives a constant factor approximation when restricted to inputs whose edit distance is *not too small*:

Theorem 1.1. *For every $T \geq 1$ there are constants $\zeta = \zeta(T)$ and $R = R(T)$ and a randomized algorithm* **FAST-ED-UB**^{T} *that takes as input an integer n and two strings x and y, with $|x|, |y| \leq n$, over an (arbitrary) alphabet Σ, and runs in time $\widetilde{O}(n^{1+\frac{1}{T}})$ and outputs an upper bound U on $d_{edit}(x, y)$, such that with probability at least $1 - 1/n$, $U \leq R \cdot (d_{edit}(x, y) + n^{1-\zeta})$.*

In particular, on any input x, y with $d_{\text{edit}}(x, y) \geq n^{1-\zeta}$ the algorithm gives a constant factor approximation. This is arguably the hardest case of edit distance. For example, it is known that one can exactly compute $d_{\text{edit}}(x, y)$ in subquadratic time on instances of small edit distance [20, 23]; in particular when $d_{\text{edit}}(x, y) \leq \sqrt{n}$ there is a linear time algorithm. The additive $n^{1-\zeta}$ term in our algorithm arises from some technical limitations in our algorithm and analysis. We expect that these technical limitations will eventually be overcome (but this may require a substantially different algorithm.)

Brakensiek and Rubinstein [13] independently obtained essentially the same theorem. While both our work and theirs builds on the techniques of [11, 12, 15, 16], the algorithms have quite different structure.

Other prior work. Because of the importance of the edit distance for various applications there is a large body of literature on computing it. Wagner and Fischer [24] gave the well-known quadratic time algorithm for computing it *exactly* using dynamic programming. Masek and Paterson [22] improved the running time slightly to $O(n^2/\log n)$, and the current asymptotically fastest algorithm (Grabowski [19]) runs in time $O(n^2 \log\log n/\log^2 n)$. Backurs and Indyk [7] showed that a *truly sub-quadratic algorithm* ($O(n^{2-\delta})$ for some $\delta > 0$) would imply a $2^{(1-\gamma)n}$ time algorithm for CNF-satisfiabilty, contradicting the Strong Exponential Time Hypothesis (SETH). Their proof proceeds by reducing the CNF-satisfiabilty to high range edit distance problem (on exponentially long strings). Abboud et al. [3] showed that even shaving an arbitrarily large polylog factor from the $O(n^2)$ running time would have the plausible, but apparently hard-to-prove, consequence that NEXP does not have non-uniform NC^1 circuits. Other "barrier" results are covered in [2, 14].

There is a long line of work on *approximating* edit distance. The exact $O(n + d_{\text{edit}}(x, y)^2)$ time algorithm of Landau *et al.* [20] yields a linear time $\sqrt{n}$-factor approximation. This approximation factor was improved, first to $n^{3/7}$ [8], then to $n^{1/3+o(1)}$ [10] and later to $2^{\widetilde{O}(\sqrt{\log n})}$ [6], all with slightly superlinear runtime. The strongest result of this type is the $(\log n)^{O(1/\epsilon)}$ factor approximation (for every $\epsilon > 0$) with running time $n^{1+\epsilon}$ of Andoni *et al.* [5]. In a somewhat different direction, Batu *et al.* [9] obtained a non-trivial $O(n^{1-\alpha})$-factor approximation algorithm running in *sublinear time*

$O(n^{\max\{\frac{\alpha}{2},2\alpha-1\}})$. Most recently, Goldenberg *et al.* [18] give an algorithm for distinguishing whether the edit distance is at most t or at least t^2 in time $\widetilde{O}(\frac{n}{t}+t^3)$. Abboud and Backurs [1] showed that a truly sub-quadratic deterministic time $(1+o(1))$-factor approximation algorithm for edit distance would imply new circuit lower bounds.

Lots of algorithmic work focuses also on other variants of edit distance such as Ulam distance, edit distance with block moves, etc. Often algorithmic properties of these measures are not quite comparable to the standard edit distance.

There are several open problems that remain to be resolved. The foremost question is whether there is a near-linear time (or perhaps even quasi-linear time) algorithm for a constant factor approximation of edit distance for the whole range of edit distance. Although this paper makes a step in that direction for the seemingly hardest case of the high edit distance we do not know how to extend the result to the whole range. Perhaps one needs a completely different technique to do that. The other intriguing question we would like to mention is the question of improving the approximation factor. The approximation factor $R(T)$ of our algorithm grows with T as the running time $O(n^{1+1/T})$ decreases. Indeed, we do not know of any truly sub-quadratic algorithm which would give approximation factor better than 2. Is there a truly sub-quadratic algorithm with approximation factor $(1+\epsilon)$, for any $\epsilon > 0$?

Reduction to a Gap-Algorithm. We will review now our basic approach to designing our approximation algorithm. For simplicity we will assume that the bound n on the length $\max(|x|,|y|)$ is a power of 2 and $|x|=|y|=n$. It is easy to reduce the general case to this case: on input x',y', let n be the least power of 2 that is at least $\max(|x'|,|y'|)$ and pad both x' and y' using a single new symbol to obtain strings x, y of length n. It is easy to verify that $d_{\text{edit}}(x',y') \le d_{\text{edit}}(x,y) \le 2d_{\text{edit}}(x',y')$, and so it suffices to approximate $d_{\text{edit}}(x,y)$.

Following a common paradigm for approximation algorithms, our approximation algorithm is built by reducing to a *gap algorithm.* In this paper, we consider randomized gap algorithms for edit distance. These algorithms take as input $(n,\theta,\delta;x,y)$ where n is an integral power of 2, x and y are strings of length n, $\theta \in (0,1]$ is a nonnegative power of $1/2$ and $\delta \in (0,1)$. The triple (n,θ,δ) are referred to as the *input parameters* and x,y as the *input strings.* We say that the algorithm has *quality Q with respect to* (n,θ,δ) provided that for all strings x,y of length n:

Gap Algorithm Soundness. If $d_{\text{edit}}(x,y) > Q\theta n$ then the algorithm returns **REJECT**.

Gap Algorithm Completeness. If $d_{\text{edit}}(x,y) \le \theta n$ then the algorithm returns **ACCEPT** with probability at least $1-\delta$.

We say that the algorithm *satisfies gap-condition*(T,ζ,Q), where $T,Q \ge 1$ and $\zeta \ge 0$ provided that for n a power of 2, and for all $\theta \ge n^{-\zeta}$

- The algorithm has quality Q with respect to (n,θ,δ),
- The running time of the algorithm on any input $(n,\theta,\delta;x,y)$ is $\widetilde{O}(n^{1+1/T}\log(1/\delta))$ with probability 1. Here $\widetilde{O}$ hides powers of $\log(n)$ whose exponent may depend on T.

We will prove:

Theorem 1.2. *For every $T \ge 1$ there are constants $\zeta = \zeta(T) > 0$ and $Q = Q(T) \ge 1$, and a gap-algorithm* **GAP-ED**^{T} *that satisfies gap-condition(T,ζ,Q).*

In Section 5 we present the (routine) construction of the algorithm **FAST-ED-UB**^{T} from **GAP-ED**^{T}, which proves Theorem 1.1. The focus of the paper is on proving Theorem 1.2.

1.1 Speed-up routines

Our algorithm, like that of [15] is built from a *core speed-up algorithm* having access to an existing "slow" gap algorithm. The speed-up algorithm produces a faster gap algorithm, with worse (but still constant) approximation quality, while making queries to the slow algorithm on pairs of "short" substrings. Given such a speed-up algorithm, one can build up a sequence of increasingly faster gap algorithms $A_0, A_1, \ldots$, where A_0 is just the quadratic exact edit distance algorithm, and A_j is obtained by using the core speed-up algorithm with A_{j-1} playing the role of the "slow" algorithm. If the core speed-up algorithm involves some free parameters that may be optimized for best performance, this optimization can be done separately for each A_j

The core speed-up algorithm designed in [15], gives an algorithm A_1 that has running time $\widetilde{O}(n^{12/7})$. The algorithms A_j are successively faster, but do not get below n^{ϕ} where $\phi = 1.61\cdots$. The core speed-up algorithm we design in this paper gives a sequence of gap-algorithms where the exponent of n in the run-time converges to 1.

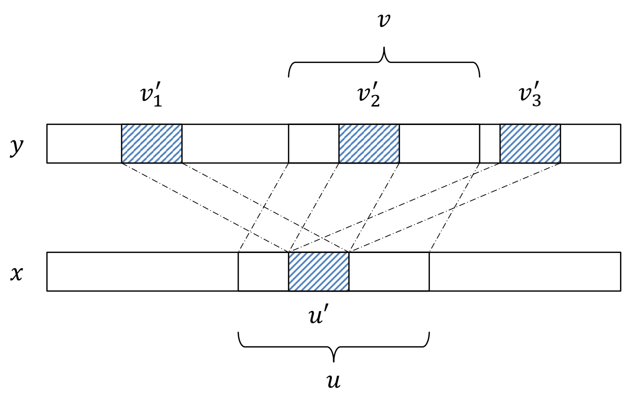
1.2 High-level overview

We explain here the basic ideas behind the core speed-up algorithm for the gap problem. This section is meant as a high-level intuitive overview where a more detailed technical overview is in Sections 3 and 4. Our new algorithm builds on ideas from [15] so we review that algorithm first and explain how we extend it further.

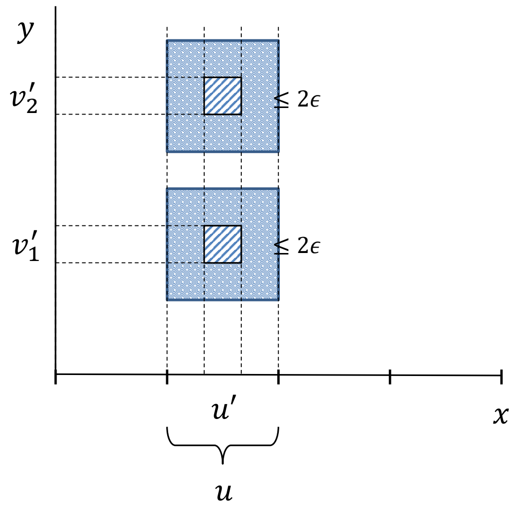
The algorithm of [15] gets two input strings x and y, it assumes that x and y are at edit distance at most θn, and it attempts to verify this assumption by constructing a good *alignment* (match) between the two strings. If x and y are indeed at edit distance at most θn then the algorithm finds an alignment of cost at most $Q\theta n$, for some fixed constant $Q > 1$. If the algorithm fails to find any such alignment then it declares x and y to be at distance larger than θn. So we focus now on how we build the match between x and y. We will divide x into blocks of size w, for suitably chosen parameter $w = n^{\Omega(1)}$. For each of the blocks we will find a matching block in y (Fig. 1a) and we will assemble a good overall match from those matches (Fig. 2b). If the overall cost of the alignment between x and y is at most θn, then on average a block u of size w in x is matched to a block v in y which is at edit distance at most θw from u. Hence, such a v is of size between $(1-\theta)w$ and $(1+\theta)w$. Thus, for a small loss in approximation factor we can assume that each block of x of size w is matched to a block of size exactly w in y. (These blocks in y might be overlapping by up-to θw symbols.) Furthermore, we can assume that the matching blocks in y start at positions that are multiples of say $\theta w/8$. (This worsens the approximation factor further but allows for a smaller set of blocks to focus on.) There is a bit of asymmetry between blocks in x which are non-overlapping and block in y which might overlap.

Let us assume for a moment that in the optimal alignment between x and y, each block of x is matched to a block of size w in y for the cost of θn. Our algorithm could find for each block u in x of size w all blocks in y that are at edit distance at most θw from u, and then from among these pieces select ones which together give the best overall match between x and y. The latter step of assembling the right pieces can be done in quasilinear time in the total number of pieces using standard techniques. There are $\frac{n}{w}$ blocks in x so if finding all the matching blocks in y could be done quickly, we could assemble the overall match in time $\widetilde{O}(n^2/w)$. For w polynomial in n this represents a substantial speed-up.

Thus the problem reduces to the question how to find quickly for each (non-overlapping) block



(a) One-dimensional representation.



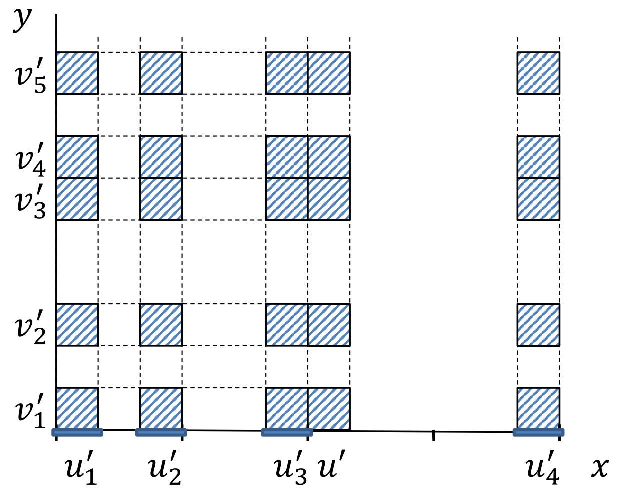
(b) Two-dimensional representation.

Figure 1: Searching for a good match for u of size w in y using a short sub-string u' of length w'.

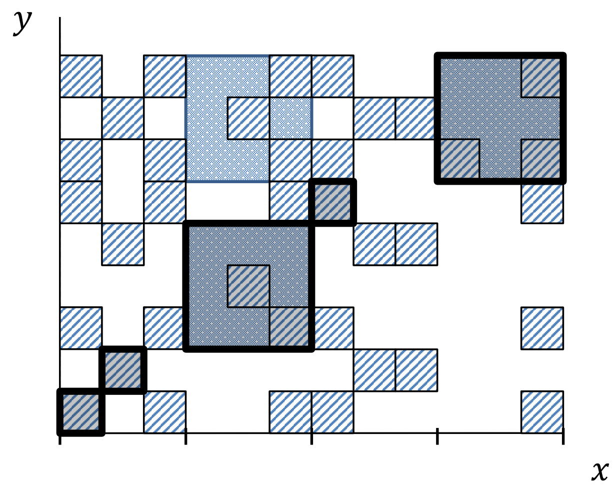
u of size w in x all blocks in y (perhaps overlapping) that are at edit distance ϵw from u, for any $\epsilon \in [\theta, 1]$. A standard solution to this problem, which is known as the *approximate pattern matching*, takes times $O(nw)$. Hence, identifying all the matches in y to all the blocks in x would take time $O(\frac{n}{w} \cdot nw) = O(n^2)$.

Easy (sparse) case. Consider a block u of size w in x. We want to find blocks of size w in y that are at edit distance at most ϵw from u. Let v be such a block. The main idea of [15] is to find v as follows: Partition u into sub-blocks of size $w' \ll w$, pick one of them at random, call it u', and find all blocks in y of size w' that are at distance at most $O(\epsilon w')$ from u'. With high probability over the choice of u', one of these blocks from y is the sub-block of v to which u' is matched in the optimal alignment of u and v. Finding these matches of size w' takes time $O(nw')$ instead of $O(nw)$. So if there are not that many matches in y for u' (u' is *sparse*) we can afford to expand each of the matches of size w' into size w by adding its neighborhood and verify by edit distance calculation which of those expanded strings is a good match to u. (See Fig. 1a and 1b.)

Dense case. We call u' of size w' *dense* (with respect to ϵ) if its number of matches in y of distance at most $\epsilon w'$ is more than some designated threshold d, and we call it *sparse* otherwise. In the sparse case we will need to check at most d possible matches for u which is relatively cheap assuming d is small. However, in the dense case we would have too many expanded blocks to check individually for a possible match. But in the dense case we have learned a lot of information about y, we found that many parts of y look similar to u'. To leverage this information we find also all blocks of size w' in x that are similar to u'. By the triangle inequality for edit distance we conclude that all pairs of blocks from x and y that are similar to u' are similar to each other (Fig. 2a). By tweaking the similarity threshold appropriately, for each block u'' in u (and x) of size w' that is similar to u' we obtain for free all blocks in y of the same size that are at distance at most $\epsilon w'$ from u''. In particular, we don't need to process any such u'' ever again and we can declare it dense. Hence, we can be picking blocks u' of size w' in u at random, removing the dense ones, until we sample a sparse block u'. If we exhaust all blocks u' in u without finding a sparse one, then we collected



(a) Substring u' is dense, and it is similar to $v_1', \ldots, v_5'$ in y and $u_1', \ldots, u_4'$ in x. These substrings are all similar to each other.



(b) Assembling a good match between x and y from partial matches to substrings of x. This can be done using one sweep from left to right.

Figure 2: Matching x to y.

information on matches in y for all blocks u' in u. From these matches we can assemble all matches for u in y in almost linear time.

The actual implementation of this idea proceeds by enumerating blocks u' of u systematically one by one, classifying each whether it is sparse or dense, removing the dense ones together with similar blocks in x from further consideration, and only then sampling from the remaining sparse blocks in u. One needs to argue that in this way we will find all good matches for u if there are some. This is non-obvious and will be shown in later sections.

To test whether a given block u' is dense or sparse we sample a $\widetilde{O}(1/d)$-fraction of blocks in y and see if there are any similar to u'. This is by a factor roughly $1/d$ cheaper then trying all the blocks of y. Similarly, when we process a dense block u' and remove all the similar blocks from x we save a factor of roughly $1/d$ of edit distance operations on strings of length w' that would need to be performed. Setting the parameters $d = n^{2/7}$, $w = n^{3/7}$ and $w' = n^{1/7}$ leads to the running time $O(n^{12/7})$ obtained in [15]. Different parts of the algorithm depend on those parameters differently and this particular setting gives the best possible trade-off.

Improving the running time. One can speed-up the algorithm of [15] by implementing the edit distance checks on strings of length w and w' by the approximation algorithm we just designed. Doing this recursively leads to faster and faster algorithms with worse and worse approximation guarantee. Nevertheless, this recursion does not give an algorithm that would run in time better than $O(n^{1.61\cdots})$. Subsequently, Andoni [4] gave for each $\epsilon > 0$ an algorithm that runs in time $O(n^{3/2+\epsilon})$. This algorithm uses the same technique as [15], but for the recursion it uses a stronger primitive. Andoni's algorithm works for any edit distance.

Our algorithm. To achieve a faster algorithm we will perform a *non-black-box* recursion of the technique from [15] to more block lengths, and implement the recursion iteratively. To find all blocks v' of length w' in y that are similar to some block u' in x we will recursively sub-divide u'

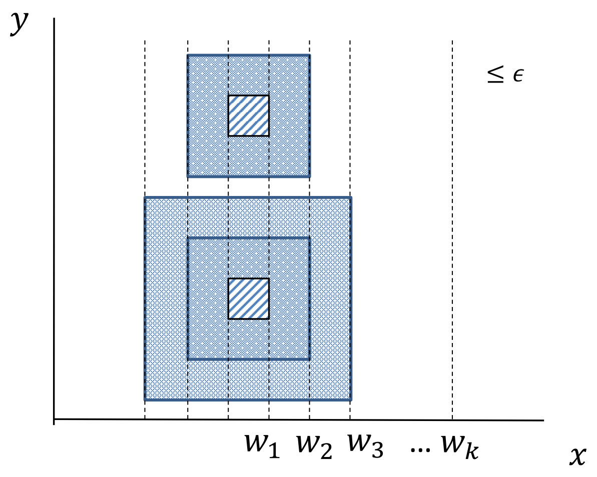


Figure 3: Idealized multi-level search for matches between x and y. (Not to scale.)

into blocks of length w'' and search for matching strings in y that will help to identify candidate v'. We iterate this for k levels.

To implement this, we will pick block lengths $w_1 = \sqrt{n} < w_2 < \cdots < w_k < n$ and density thresholds $\sqrt{n} > d_1 > d2 > \cdots > d_k = 1$. The block length w_i and density d_i is picked so that the amount of work for each level i would be roughly the same. Multiple levels provide more freedom to set the parameters, yet, to get an algorithm with running time close to linear one still needs to recurse the resulting algorithms as done in [15]. We will elaborate on that at the end of this section.

The main idea which we want to implement in the new algorithm is when looking for matches of a block u_i of size w_i, we sub-divide the block into sub-blocks of size w_{i-1}, we recursively find all good matches to a randomly selected (presumably sparse) sub-block of u_i of size w_{i-1}, and we use those matches to identify small set of candidates for the matches of u_i which we prune by checking their actual edit distance from u_i. At the bottom of the recursion we look for matches to blocks of size w_1 naïvely. Then we repeatedly expand and prune those matches as we go up the recursion levels (Fig. 3). The challenge is that as we progress up in the recursion at some level i we might get more than d_i candidates to check their edit distance since d_i's are decreasing. On the other hand since we are pruning the candidate strings at some later level the number of the candidates could fall back below the corresponding density threshold. So being dense is not a monotone property with respect to the recursion level. It is not clear how to define being dense, how to test it, and how to deal with it in the multi-level context.

Data structure approach. Our approach to implement the above idea is to "slice the recursion horizontally" and proceed iteratively. For each level $i = 1, \ldots, k$, we will build a data structure that allows to quickly enumerate matches in y (and x) to any block of length w_i in x. We query this data structure via the procedure EnumerateClose which is the central component of our algorithm. EnumerateClose gets as its input a level index i, a substring u of x of length w_i and a similarity threshold ϵ, and it must enumerate *all* matches to u_i in y (and x) at distance at most ϵw_i. The data structure for level i will consist of information on which blocks of size w_{i-1} in x are dense and which are sparse with respect to d_{i-1}. For the dense blocks we will also keep the complete list of similar blocks in y. This will be maintained for all similarity thresholds $\epsilon \in [\theta, 1]$, where ϵ is a power of $1/2$.

When building the data structure for level i, for each block u in x of size w_i, we will first find all blocks in y at distance at most ϵw_i from u that can be inferred from the information on dense blocks of size w_{i-1}. That is we use the stored information on matches between (dense) blocks of size w_{i-1} to identify some matches between blocks of size w_i, by assembling together the known matches of size w_{i-1}. This can be done efficiently in time almost linear in the total number of stored matches. This identifies some of the matches in y for each u. All the other matches to u can be identified with high probability by sampling a sparse sub-block of u of length w_{i-1}, enumerating all its matches in y, expanding them and pruning them as usually. (The fact that this works is a non-trivial technical fact which requires a careful argument.) However, the other matches are calculated only during each query to the data structure. Thus our procedure EnumerateClose returns a union of blocks inferred from the dense sub-blocks together with blocks identified from the sampled sparse sub-block. Calling the procedure EnumerateClose leads to recursive calls on sub-blocks from lower levels that we already know are sparse. This eliminates the problem with non-monotonicity of sparseness.

Hence, the key parts of the data structure for level i are the classification of blocks of level $i-1$ into sparse and dense, complete information on matches for the dense blocks of level $i-1$, and a list $\mathcal{B}^{\text{below}}$ of matches for strings at level i that can be inferred from the dense blocks at level $i-1$.

During the initialization of the data structure for level i, we compute the list $\mathcal{B}^{\text{below}}$. Once that is computed, the procedure EnumerateClose can be safely called with any block of size w_i. We use the procedure EnumerateClose to determine which blocks of size w_i are dense and which are sparse in order to prepare the data structure for next level $i+1$. However, enumerating all blocks in y that are similar to each of the blocks u in x of size w_i to determine the density of u would be too expensive. So we allow the procedure EnumerateClose to get as an extra input a list of blocks in y on which to focus. When checking the density of u we sample $\widetilde{O}(1/d_i)$ blocks from y from which EnumerateClose selects the close matches. If there are $\widetilde{O}(1)$ matches, the block u is sparse otherwise it is dense and we run EnumerateClose again to identify all the matches to u, and also to identify all blocks in x similar to u. Similarly to [15] we take all pairs of such blocks and store them as the relevant information for the dense blocks of size w_i. The $\widetilde{O}(1/d_i)$ savings in running the density tests is matched by $\widetilde{O}(1/d_i)$ savings in processing the dense blocks (as we do not have to process blocks in x deemed similar to some already processed block u). These savings are key to the overall savings of the algorithm.

Although the overall idea looks quite simple substantial care has to be taken to fit all the parts of the algorithm together. By a careful choice of parameters w_i and d_i we can speed-up a base (approximate) edit distance algorithm running in time $O(n^{1+1/T})$ into an algorithm that runs in time $O(n^{1+1/(T+1/6)})$. The approximation factor gets worse by a constant factor. We can repeat this any number of times to get an algorithm running in time $O(n^{1+\epsilon})$ for an arbitrary $\epsilon > 0$.

One unfortunate feature of the algorithms is that they work only in the high edit distance regime $\geq n^{1-\zeta}$. The problem is that the parameter θ affects the running time of the algorithm. For small edit distance, [15] actually uses an exact edit distance algorithm running in time $O(n + (\theta n)^2)$. The approximation algorithm is run only for large enough θ. As we recursively apply the approximation algorithm to build the faster one, the working range of the algorithm actually shrinks. As we do not have near linear time algorithm for gap edit distance for medium range θ such as $\theta = n^{-1/3}$ we cannot fall back on standard algorithms for this range. We do not know how to extend the range of our near linear time algorithm.

In what follows we provide a more specific description of the algorithm. We will look at the problem as the problem of finding a shortest path in the *edit distance graph* (*grid graph*) associated with x and y. As we need to look for similarities between blocks of x and y and also between blocks of x, it is convenient to define $z = xy$ and build the edit distance graph for x and z. We will identify blocks in x by intervals of positions denoted by I and blocks in z by J. We will introduce this terminology in next section. In Section 3 we provide a more quantitative overview of the algorithm from [15], and in Section 4 we describe the new algorithm.

2 Preliminaries

Many definitions and routine claims are adapted (with some modifications) from [16]. The edit distance of strings u, v is denoted $d_{\text{edit}}(u, v)$ and the *normalized edit distance* of u, v, denoted $\Delta_{\text{edit}}(u, v)$ is defined to be $d_{\text{edit}}(u, v)/|u|$.

Throughout the paper x, y denote two input strings of length n, where n is a power of 2, and z denotes the concatenation xy.

Intervals, Decompositions, aligned intervals, and δ-aligned intervals. We consider intervals in $\{0, \ldots, 2n\}$ which are as usual, subsets consisting of consecutive integers. The *width* of interval I, $\mu(I)$ is equal to $\max(I) - \min(I) = |I| - 1$. Most intervals we consider have width a power of 2. An interval of width w is a w-interval. Intervals index substrings of z, where z_I denotes the substring indexed by the set $I \setminus \{\min(I)\}$. (Note that $z_{\min(I)}$ is not part of z_I. In particular, $z = z_{\{0,\ldots,2n\}}$, and $x = z_{\{0,\ldots,n\}}$ and $y = z_{\{n,\ldots,2n\}}$.)

A *decomposition* of an interval I is a sequence $I_1, \ldots, I_k$ of intervals with $\min(I_1) = \min(I)$, $\max(I_k) = \max(I)$ and $\min(I_{j+1}) = \max(I_j)$ for $j \in \{1, \ldots, k-1\}$. Note that $z_{I_1}, \ldots, z_{I_k}$ partitions the string z_I.

Let w be a power of 2 that is at most n, and let δ be a power of 2 that is at most 1. An interval of width w is *aligned* if $\min(I)$ is a multiple of w (and consequently $\max(I)$ is also a multiple of w). The interval is *δ-aligned* if $\min(I)$ is a multiple of $\max(\delta w, 1)$ (and consequently so is $\max(I)$). In particular a 1-aligned interval is aligned. We define:

- $\text{Intervals}(w)$ is the set of aligned intervals of width w, subsets of $\{0, \ldots, n\}$.
- $\text{Intervals}(w, \delta)$ to be the set of δ-aligned intervals of width w, subsets of $\{0, \ldots, 2n\}$.
- For an interval I, $\text{Intervals}(w; I) = \{I' \in \text{Intervals}(w) : I' \subseteq I\}$, and $\text{Intervals}(w, \delta; I) = \{I' \in \text{Intervals}(w, \delta) : I' \subseteq I\}$.

Since n and w are powers of 2, $\text{Intervals}(w)$ is a decomposition of $\{0, \ldots, n\}$. When we use the notation $\text{Intervals}(w; I)$, I will be an aligned interval of width a power of 2, so that $\text{Intervals}(w; I)$ is a decomposition of I.

The grid $\{0, \ldots, n\} \times \{0, \ldots, 2n\}$, boxes and stacks. Consider the grid $\{0, \ldots, n\} \times \{0, \ldots, 2n\}$ lying in the coordinate plane. For $S \subseteq \{0, \ldots, n\} \times \{0, \ldots, n\}$, the horizontal projection $\pi_H(S)$ is the set of first coordinates of elements of S, and the vertical projection of S, $\pi_V(S)$ is the set of second coordinates.

A *box* is a set $I \times J \subseteq \{0, \ldots, n\} \times \{0, \ldots, 2n\}$ for intervals I, J, and it represents the pair x_I, z_J of substrings. Since $I \subseteq \{0, \ldots, n\}$, $z_I = x_I$. Note that if $J \subseteq \{0, \ldots, n\}$ then z_I, z_J is a pair of substrings of x and if $J \subseteq \{n, \ldots, 2n\}$, it is a pair (substring of x, substring of y). $I \times J$ is a *w-box* if

$\mu(I) = \mu(J) = w$. The lower left hand corner is $(\min(I), \min(J))$ and the upper right hand corner is $(\max(I), \max(J))$. Note that $\pi_H(I \times J) = I$ and $\pi_V(I \times J) = J$. Box $I \times J$ is *horizontally aligned* if I is aligned, and it is *vertically δ-aligned* or simply *δ-aligned* if J is δ-aligned; we have no need to refer to *horizontally δ-aligned* boxes. Box $I \times J$ is *square* if $\mu(I) = \mu(J)$.

A *stack* is a set of boxes all having the same horizontal projection. For interval I and set of intervals $\mathcal{J}$, $I \times \mathcal{J}$ is the stack $\{I \times J : J \in \mathcal{J}\}$.

Grid graphs. The *grid graph of z*, G_z, is a directed graph with edge costs, having vertex set $\{0, \ldots, n\} \times \{0, \ldots, 2n\}$ and all edges of the form $(i-1, j) \to (i, j)$ (H-edges), $(i, j-1) \to (i, j)$ (V-edges) and $(i-1, j-1) \to (i, j)$ (D-edges). Every H-edge and V-edge costs 1, and a D-edge has cost 1 if $z_i \neq z_j$ and 0 otherwise. G_z is acyclic, with edges moving "up and to the right". A directed path τ joins a pair of vertices source(τ) and sink(τ) with source(τ) $\leq$ sink(τ). The *box spanned by τ* is the unique minimal box $I \times J$ that contains τ; this is equal to $\pi_H(\tau) \times \pi_V(\tau)$. We say τ *traverses* $I \times J$ if $I \times J$ is the box spanned by τ, which is equivalent to source(τ) $= (\min(I), \min(J))$ and sink(τ) $= (\max(I), \max(J))$. A *traversal* of $I \times J$ is any path that traverses $I \times J$.

For $I \subseteq \pi_H(\tau)$, let τ_I denote the minimal subpath of τ whose horizontal projection is I.

Cost and normalized cost. The *cost* of a directed path τ, cost(τ) is the sum of the edge costs, and the *normalized cost* is $\text{ncost}(\tau) = \frac{\text{cost}(\tau)}{\mu(\pi_H(\tau))}$. The cost of box $I \times J$, cost($I \times J$), is the min-cost of a traversal of $I \times J$ and $\text{ncost}(I \times J) = \frac{1}{\mu(I)}\text{cost}(I \times J)$.

It is well known (and easy to see) that for any box $I \times J$, a traversal of $I \times J$ corresponds to an *alignment* from $a = z_I$ to $b = z_J$, i.e. a set of character deletions, insertions and substitutions that changes a to b, where an H-edge $(i-1, j) \to (i, j)$ corresponds to "delete a_i", a V-edge $(i, j-1) \to (i, j)$ corresponds to "insert b_j between a_i and a_{i+1}" and a D-edge $(i-1, j-1) \to (i, j)$ corresponds to replace a_i by b_j, unless they are already equal. Thus:

Proposition 2.1. *The cost of an alignment corresponding to path τ is cost(τ). Thus for any $I, J \subseteq \{0, \ldots, 2n\}$, $d_{edit}(z_I, z_J) = $ cost$(I \times J)$. In particular $d_{edit}(x, y) = $ cost$(\{0, \ldots, n\} \times \{n, \ldots, 2n\})$.*

Displacement of a box relative to a path or box. The following easy fact (noted in [15]) relates the cost of two boxes having the same horizontal projection:

Proposition 2.2. *For intervals $I, J, J' \subseteq \{0, \ldots, n\}$, $|\text{cost}(I \times J) - \text{cost}(I \times J')| \leq |J \Delta J'|$, where Δ denotes symmetric difference.*

Let τ be a path whose horizontal projection includes I. The *displacement* of the square box $I \times J$ with respect to τ, disp$(I \times J, \tau)$ is the smallest K such that $(\min(I), \min(J))$ is within K vertical units of source(τ_I) and $(\max(I), \max(J))$ is within K vertical units of sink(τ_I).

We make a few easy observations.

Proposition 2.3. *Let τ be a path whose horizontal projection includes I and let $I \times J$ be a box. Then cost$(I \times J) \leq$ cost$(\tau_I) + 2$disp$(I \times J, \tau)$.*

Proof. Let J' be the vertical projection of τ_I. Then: $\text{cost}(I \times J) \leq \text{cost}(I \times J') + |J \Delta J'| \leq \text{cost}(\tau_I) + |J \Delta J'| \leq \text{cost}(\tau_I) + 2\text{disp}(I \times J, \tau)$. □

The following fact (which is essentially the same as Proposition 3.4 of [16]) says that every path τ with projection I' can be approximately covered by a δ-aligned box whose cost is close to cost(τ) and whose displacement from τ is small:

Proposition 2.4. *Let I' and J be intervals and suppose $\delta \in (0,1]$. Let τ be a path lying inside of $I' \times J$ whose horizontal projection is I'. There is a δ-aligned interval J' of width $\mu(I')$ so that $disp(I' \times J', \tau_{I'}) \leq \delta\mu(I') + cost(\tau_{I'})$ and $ncost(I' \times J') \leq 2ncost(\tau_{I'}) + \delta$.*

Proof. Let J be the vertical projection of $\tau_{I'}$. If $\mu(J) \geq \mu(I')$ then let $\hat{J}$ be the interval of width $\mu(I')$ with $\min(\hat{J}) = \min(J)$. Otherwise let $\hat{J}$ be any interval of width $\mu(I')$ that contains J.

The box $I' \times \hat{J}$ has displacement at most $\text{cost}(\tau_{I'})$ from $\tau_{I'}$, and has cost at most $2\text{cost}(\tau_{I'})$. Finally, let J' be obtained by shifting $\hat{J}$ up or down to the closest δ-aligned interval. This shift is at most $\delta/2$ units. This increases both the displacement and the cost by at most $\delta\mu(I')$. □

The *diagonal* of a square box $I \times J$ is the diagonal path joining $(\min(I), \min(J))$ to $(\max(I), \max(J))$. Let $I \times J$ and $I' \times J'$ be square boxes with $I' \subseteq I$. The *displacement* of $I' \times J'$ with respect to $I \times J$, $\text{disp}(I' \times J', I \times J)$ is the displacement of $I' \times J'$ with respect to the diagonal of $I \times J$, which is just the number of vertical units one needs to shift $I' \times J'$ so that its diagonal is a subpath of the diagonal of $I \times J$.

Proposition 2.5. *Suppose τ traverses the square box $I \times J$ of width w. Then every point of τ is within vertical distance $cost(\tau)/2$ of the diagonal of $I \times J$.*

Proof. Consider a point of τ expressed as $P = (\min(I) + u, \min(J) + v)$. Then τ can be split into two parts τ_1, ending at P and τ_2 starting at P. Then $\text{cost}(\tau) = \text{cost}(\tau_1) + \text{cost}(\tau_2) \geq 2|v - u|$ which is twice the vertical distance of P to the diagonal of $I \times J$. □

Weighted boxes and stacks, certified boxes and stacks, shortcut graphs.

A *weighted box* is a pair $(I \times J, \kappa)$ where $\kappa \geq 0$. If $\text{ncost}(I \times J) \leq \kappa$ we say that $(I \times J, \kappa)$ is a *certified box.* A *weighted stack* $(I \times \mathcal{J}, \kappa)$ is a pair where $I \times \mathcal{J}$ is a stack and $\kappa \geq 0$. We associate $(I \times \mathcal{J}, \kappa)$ with the set $\{(I \times J, \kappa) : J \in \mathcal{J}\}$. If every box in $(I \times \mathcal{J}, \kappa)$ is certified, we call it a *certified stack.*

Let $\widetilde{G}$ be the digraph on $\{0, \ldots, n\} \times \{0, \ldots, 2n\}$ with arc set $\{(i,j) \to (i',j') : i \leq i', j \leq j', (i,j) \neq (i',j')\}$ The edges with $i < i'$ and $j < j'$ are called *shortcuts.* Associated to any weighted box $(I \times J, \kappa)$ there is a weighted shortcut edge $(\min(I), \min(J)) \to (\max(I), \max(J))$ with weight $\kappa\mu(I)$. Given a set $\mathcal{R}$ of weighted boxes, we define the *weighted shortcut graph* $\widetilde{G}(\mathcal{R})$ to be the weighted directed graph consisting of all H-edges and V-edges with weight 1, and all of the shortcut edges corresponding to the boxes in $\mathcal{R}$. For a box $I \times J$, let $\text{cost}_{\mathcal{R}}(I \times J)$ denote the minimum cost of a traversal of $I \times J$ in $\widetilde{G}(\mathcal{R})$.

If every box in $\mathcal{R}$ is certified we say that $\widetilde{G}(\mathcal{R})$ is a *certified shortcut graph.* A certified shortcut graph $\bar{G}(\mathcal{R})$ provides upper bounds on the edit distance. We omit the proof of the following easy fact:

Proposition 2.6. *Let $\mathcal{R}$ be a set of certified boxes. For any box $I \times J$, $d_{edit}(z_I, z_J) \leq cost_{\mathcal{R}}(I \times J)$.*

3 The core speed-up algorithm of [15]

As discussed in Sections 1.1 and 1.2, the main ingredient in [15] is a core speed-up algorithm that has access to a slow edit distance approximation algorithm and uses it to build a faster approximation algorithm. We give here a more quantitative review of the core speed-up algorithm in [15], which provides the starting point for ours. To simplify the description we assume that the slow edit distance algorithm is just the quadratic exact edit distance algorithm. In their work, they reduce to the case $\theta > n^{-1/5}$ and build a subquadratic time algorithm for the gap-problem where $\theta \geq n^{-1/5}$. The algorithm operates in two phases. The *discovery phase* generates a set $\mathcal{Q}$ of certified boxes which represent known matches between x and y. In the *shortest path phase* the algorithm assembles these boxes together to evaluate the cost of $(\{0, \ldots, n\} \times \{n, \ldots, 2n\})$ in the shortcut graph $\widetilde{G}(\mathcal{R})$ where $\mathcal{R}$ is a set of certified boxes obtained by a minor modification of $\mathcal{Q}$. Proposition 2.6 implies that this is an upper bound on $d_{\text{edit}}(x, y)$. The main work is to define the discovery phase to ensure that this upper bound is not too much bigger than the true value. The shortest path phase is implemented by a straightforward variant of dynamic programming.

The discovery phase is defined in terms of parameters $w_1 < d < w_2$, which are powers of 2 that are, respectively, approximately $n^{1/7}$, $n^{2/7}$ and $n^{3/7}$. The set $\mathcal{Q}$ consists of certified w_1-boxes and certified w_2-boxes, and satisfies with high probability: for every horizontally aligned w_2-box $I \times J$, $\text{cost}_{\mathcal{R}}(I \times J) \leq C \cdot [\text{cost}(I \times J) + \theta w_2]$ for some constant C. It is not difficult to show that this implies that the upper bound on $d_{\text{edit}}(x, y)$ output by the shortest path inference phase will be at most $C \cdot [d_{\text{edit}}(x, y) + \theta n]$, which is enough to solve the gap-problem.

The algorithm generates boxes of width w_1 iteratively for i from $0, \ldots, \log(1/\theta)$ and $\varepsilon(i) = 2^{-i}$. For each horizontally aligned I, let $\mathcal{N}_{\varepsilon(i)}(I)$ be the set of J that are $\varepsilon(i+3)$-aligned and satisfy $\text{ncost}(I \times J) \leq \varepsilon(i)$. Iteration i starts by classifying each of the n/w_1-aligned w_1-intervals, as *dense* or *sparse* subject to the requirement that every I with $N_{\varepsilon(i)}(I) \geq 2d$ is classified as dense, and every I with $N_{\varepsilon(i)}(I) \leq d/2$ is classified as sparse; this classification of I is done with high probability by sampling J at a rate $\log(n)/d$ and calling I dense (resp. sparse) if at least (resp. at most) $\log(n)$ of the sample are within distance $\varepsilon(i)$ of I. Next for each dense interval I a set $\mathcal{J}(I)$ of $\varepsilon(i+3)$-aligned w_1-intervals J is constructed such that $\text{ncost}(I \times J) \leq 5\varepsilon(i)$ and $\mathcal{N}_{\varepsilon(i)}(I) \subseteq \mathcal{J}(I)$. For any given I we can construct $\mathcal{J}(I)$ by computing its edit distance with every $\varepsilon(i)/8$-aligned interval, in time $O(nw_1/\varepsilon(i))$. If we do this for all n/w_1-aligned intervals the time is $\Theta(n^2/\varepsilon(i))$, but the restriction to dense intervals allows a savings of a factor of $\varepsilon(i)d$: Initialize $\mathcal{D}$ to be the set of dense aligned w_1-intervals. While $\mathcal{D} \neq \emptyset$ choose $I \in \mathcal{D}$ (the *pivot* for the current round) and construct $\mathcal{X} = N_{2\varepsilon(i)}(I)$ and $\mathcal{Y} = N_{3\varepsilon(i)}(I)$ and certify all boxes $(I' \times J', 5\varepsilon(i))$ for $I' \in \mathcal{X}$ and $J' \in \mathcal{Y}$. Delete $\mathcal{X}$ from $\mathcal{D}$ and continue. The number of pivots is thus only $O(n/w_1\varepsilon(i)d)$ since the sets $N_{\varepsilon(i)}(I)$ are of size at least d and are disjoint for different pivots.

The rest of the discovery phase constructs a (relatively small) set of w_2-boxes. For each horizontally aligned w_2-interval I', the w_1-subintervals of I' that were declared sparse (over all iterations of i) are used to select a small subset $\mathcal{J}'(I')$ of the w_2-intervals, and we certify each box $I' \times J'$ for $J' \in \mathcal{J}'(I')$ by computing their edit distance exactly. The set $\mathcal{J}'(I')$ is obtained as follows: For each $i \in \{0, \ldots, \log(1/\theta)\}$, select a polylog$(n)$ size subset $\mathcal{S}_i(I')$ of the subintervals of I' that were declared sparse in iteration i, and for each $I'' \in \mathcal{S}_i(I')$ exactly compute $\text{cost}(I'', J)$ for all $\varepsilon(i+3)$-aligned intervals J to determine $\mathcal{N}_{\varepsilon(i)}(I'')$ (which has size at most $2d$). For each box $I'' \times J$, let J' be the unique w_2-interval such that the diagonal of $I'' \times J$ is a subset of the diagonal of $I' \times J'$ and add J' to $\mathcal{J}'(I')$. The size of $\mathcal{J}'(I')$ is $\widetilde{O}(d)$ and so the total cost of evaluating the edit distance of boxes $I' \times J'$ for $I' \in \text{Intervals}(w_2; \{0, \ldots, n\})$ and $J' \in \mathcal{J}'(I')$ is $\widetilde{O}(ndw_2)$.

The parameters w_1, d, w_2 are adjusted to minimize the run time at $\widetilde{O}(n^{12/7})$. The key claim in [15] is that for every horizontally aligned w_2-box $I \times J$, the boxes from the discovery phase imply an upper bound $\text{ncost}(I \times J)$ that is at most $C \cdot \text{ncost}(I \times J) + C'\theta$ which is sufficient for the shortcut phase to succeed. The claim is proved by showing that if the set of certified w_1-boxes does not imply a sufficiently good upper bound on $\text{ncost}(I \times J)$, then with high probability, one of the w_2-boxes

$I \times J'$ constructed in the second part of the discovery phase is within a small vertical shift of $I \times J$, and therefore can be used in the inference phase to imply a good upper bound on $\mathrm{cost}(I \times J)$.

4 The new core speed-up algorithm

The main new ingredient of the new core speed-up algorithm presented here is the replacement of the pair $w_1 < w_2$ of widths from [15] by a hierarchy $w_1 < \cdots < w_k$ of widths. While the idea of such an extension is natural, it is not a priori clear how to extend the ideas of [15] to such a hierarchy. Our new algorithm proceeds in k iterations. During iteration j the algorithm builds a data structure that supports approximate distance queries between substrings of width w_j. Each successive data structure recursively uses the data structure from the previous iterations. Iteration j is accomplished by a suitable variant of the algorithm from [15].

The algorithm of [15] splits neatly into a discovery phase and an inference phase. In the new algorithm, each iteration begins with an inference phase (using boxes discovered in the previous phase) followed by a discovery phase.

Here is our main speed-up theorem.

Theorem 4.1. *Suppose that SLOW-GED is a gap algorithm for edit distance satisfying gap-condition*(T', ζ', Q') *where* $T' \geq 1$, $\zeta' > 0$ *and* $Q' \geq 1$. *There is an algorithm FAST-GED (using SLOW-GED as a subroutine) that satisfies gap-condition*(T, ζ, Q) *with* $T = T' + 1/6$ *where* $\zeta > 0$ *and* $Q \geq 1$ *are suitably chosen (depending only on* T',ζ' *and* Q'*).*

Applying this theorem inductively with A_0 being the exact edit distance algorithm, we get a sequence of algorithms A_j where A_j satisfies gap-condition$(1 + j/6, \zeta_j, Q_k)$ for suitable constants $\zeta_j > 0$ and Q_j, and taking $j = 6(T - 1)$ gives Theorem 1.2.

The proof of Theorem 4.1 is the heart of the paper. We describe the algorithm in the following order:

1. The parameters used by the algorithm (Section 4.1).
2. The overall architecture, including data objects, of the algorithm (Section 4.2).
3. Some basic functions used in the algorithm (Section 4.3).
4. The mechanics of the algorithm. (Section 4.4).
5. The use of randomness in the algorithm (Section 4.5).
6. The properties enforced by the algorithm (Section 4.6 and 4.7).
7. The proof that FAST-GED satisifes the gap-algorithm Soundness and Completeness requirements (Section 4.8).
8. The running time analysis in terms of the parameters (Section 4.9).
9. The choice of parameters that attain the run time claims for FAST-GED (Section 4.10).
10. Tying up the proof of Theorem 4.1 (Section 4.11).

4.1 The algorithm parameters

Recall that a gap-algorithm takes as input $(n, \theta, \delta; x, y)$ where n is a power of 2 and $|x| = |y| = n$, and $\theta \in (0, 1]$ is a power of $1/2$.

In our description of the algorithm, we fix the input parameter δ in the algorithm FAST-GED to $\delta = 1/2$. For $\delta < 1/2$, we execute the algorihm with $\delta = 1/2$ independently for $r = \lceil \log_2(1/\delta) \rceil$ times, and **REJECT** only if every run returns **REJECT**. This compound algorithm will **REJECT** every input x, y such that $d_{\text{edit}}(x, y) \geq Q\theta n$, since every run will **REJECT**. The probability that the compound algorithm incorrectly returns **REJECT** on input with $d_{\text{edit}}(x, y) \leq \theta n$ is at most $(1/2)^r \leq \delta$, as required.

Second, we fix the value of δ for all calls of SLOW-GED within FAST-GED, to $\delta = n^{-12}$ where n is the length of the global input to FAST-GED. Since the number of calls to SLOW-GED will be bounded above (easily) by $\widetilde{O}(n^2)$, a union bound implies that the probability that every call to SLOW-GED is correct is at least $1 - n^{-8}$.

The algorithm FAST-GED takes as input $n, \theta; x, y$ where n is a power of 2, x and y are strings of length n and the gap parameter $\theta \in (0, 1]$ is a power of $1/2$. The algorithm sets z to be the concatenation of xy and treats z as a global variable.

The number of iterations (levels) of FAST-GED is a parameter k. For each $j \in 1, \ldots, k+1$, there is a *width parameter* w_j and for each $j \in \{0, \ldots, k\}$, there is a *density parameter* d_j. These parameters are integer powers of 2 satisfying:[1]

$$w_1 = \lfloor \sqrt{n} \rfloor_2 < w_2 < \cdots < w_k < w_{k+1} = n.$$

$$d_0 = \lfloor \sqrt{n} \rfloor_2 > d_1 > \cdots > d_k = 1.$$

Furthermore, for $1 \leq j \leq k$:

$$\frac{n}{w_j} \quad \geq \quad d_j. \tag{1}$$

These parameters will be chosen in Section 4.10 to optimize the time analysis. For now we note a technical assumption, that will be verified in Section 4.10, that is needed in the analysis. For $1 \leq j \leq k$:

$$\frac{w_j}{w_{j+1}} \quad \leq \quad \theta/2. \tag{2}$$

For each $j \in \{0, \ldots, k\}$, there are quality parameters q_j that satisfy the recurrence:

$$\begin{aligned} q_0 &= \log(Q') && \text{(where } Q' \text{ is the quality of SLOW-GED)} \\ q_j &= 3q_{j-1} + 21 && \text{for } j > 1. \end{aligned}$$

The quality of the final approximation is $Q = 2^{q_k + 6}$

We also define, for integers i, $\varepsilon(i) = 2^{-i}$. In most cases, $i \in \{0, \ldots, \log(1/\theta)\}$ so $1 \geq \varepsilon(i) \geq \theta$.

There is a constant c_0 used in the definition of the procedure ProcessDense. (See Section 4.5.)

[1]We denote by $\lfloor \cdot \rfloor_2$ the closest power of two of size smaller or equal.

4.2 The architecture of the algorithm, and the neighborhood data structure

FAST-GED consists of k iterations (*levels*), and a final post-processing step. During iteration j, the algorithm examines pairs $\langle i; I \times J\rangle$, called *candidates*, where $i \in \{0, \ldots, \log(1/\theta)\}$, $I \in \text{Intervals}(w_j)$ and $J \in \text{Intervals}(w_j, \varepsilon(i+3))$. (Hence, a candidate is any $\langle i; I \times J\rangle$ that satisfies some weak consistency requirements.) The pair $I \times J$ is called a *level j box* and $\langle i; I \times J\rangle$ is a *level j candidate.* Iteration j implicitly classifies all level j-candidates as **CLOSE** or **FAR**. This classification satisfies:

- If $\text{ncost}(I \times J) \le \varepsilon(i)$ then $\langle i; I \times J\rangle$ is classified as **CLOSE**.
- If $\text{ncost}(I \times J) > \varepsilon(i - q_{j-1} - 6)$ then $\langle i; I \times J\rangle$ is classified as **FAR**.

If $\varepsilon(i) < \text{ncost}(I \times J) \le \varepsilon(i - q_{j-1} - 6)$ then $\langle i; I \times J\rangle$ may be classified as either **CLOSE** or **FAR**.

This implicit classification is accomplished by a data structure, called the *neighborhood data structure.* The data structure implements a query EnumerateClose which takes as input $(j, I \times \mathcal{J}, i)$ where:

- $j \in \{1, \ldots, k\}$ is the level,
- $I \times \mathcal{J}$ is a stack satisfying $I \in \text{Intervals}(w_j)$ and $\mathcal{J} \subseteq \text{Intervals}(w_j, \varepsilon(i+3))$,
- $i \in \{0, \ldots, \log(1/\theta)\}$,

and returns the set of $J \in \mathcal{J}$ for which $\langle i; I \times J\rangle$ is **CLOSE**. In particular, $\text{EnumerateClose}(j, I \times \{J\}, i)$ returns $\{J\}$ if $\langle i; I \times J\rangle$ is **CLOSE** and returns $\emptyset$ otherwise. The pair $\langle i; I \times \mathcal{J}\rangle$ is called a *level j candidate stack.*

The queries with level parameter j are the *level j queries.* Initially the data structure is unable to answer any queries. During iteration j the algorithm constructs the part of the data structure that determines the classification of level j candidates as **CLOSE** or **FAR**, and thereby enabling level j queries.

At the start of iteration j, queries up to level $j-1$ have been enabled. To enable $\text{EnumerateClose}(j, \cdot)$ the algorithm constructs families of sets for each $I \in \text{Intervals}(w_j)$ and each $i \in \{0, \ldots, \log(1/\theta)\}$ as follows:

- A subset of $\text{Intervals}(w_j, \varepsilon(i+3))$ denoted $\mathcal{B}^{\text{below}}(j, I, i)$.
- A subset of $\text{Intervals}(w_{j-1}; I)$ denoted $\text{SparseSample}(j, I, i)$.

The query $\text{EnumerateClose}(j, \cdot)$ uses these sets, as well as calls to $\text{EnumerateClose}(j-1, \cdot)$. Thus the level j neighborhood data structure consists of all of the sets $\mathcal{B}^{\text{below}}(j', \cdot)$ and $\text{SparseSample}(j', \cdot)$ for $1 \le j' \le j$.

During iteration j, subroutines Preprocess and ProcessDense are called with parameter j. The purpose of $\text{Preprocess}(j)$ is to create the sets $\mathcal{B}^{\text{below}}(j, \cdot)$ and $\text{SparseSample}(j, \cdot)$. The construction of these sets involves some random choices, which affect the **CLOSE**/**FAR** classification; but once the choices are made the **CLOSE**/**FAR** classification is fixed. The creation of these sets activates $\text{EnumerateClose}(j, \cdot)$. While the data structure grows during each iteration to enable higher level queries, once $\text{EnumerateClose}(j, \cdot)$ is enabled, the portion of the data structure used to handle level j queries is static.

The other procedure in iteration j of FAST-GED() is $\text{ProcessDense}(j)$. $\text{ProcessDense}(j)$ creates the following sets for each $i \in \{0, \ldots, \log(1/\theta)\}$:

- Sparse$(j,i) \subseteq$ Intervals(w_j).
- For each $I \notin$ Sparse(j,i), a subset of Intervals$(w_j, \varepsilon(i+3))$ denoted $\mathcal{B}^{\mathbf{dense}}(j,I,i)$.
- A set $\mathcal{R}(j)$ of weighted boxes (which we will prove are all certified).

The sets $\mathcal{B}^{\text{dense}}(j,\cdot)$ are local variables within ProcessDense(j), used to create $\mathcal{R}(j)$.

The set $\mathcal{R}(j)$ and Sparse$(j,\cdot)$ are global variables but, with the exception of the final iteration $j=k$, they are used only in Preprocess$(j+1)$, and then never used again. Following iteration k, the set $\mathcal{R}(k)$ is used in the post-processing step to generate the final output which is $\text{cost}_{R(k)}(\{0,\dots,n\} \times \{n,\dots,2n\})$.

4.3 Elementary primitives

We describe some elementary functions used within the algorithm.

The function Round. Round(J,ϵ) where J is an interval and $\epsilon \le 1$ is a power of 2, is equal to the ϵ-aligned interval J' of width $\mu(J)$ obtained by shifting J down (decreasing its two endpoints) at most $\epsilon\mu(J) - 1$ units.

The function ZoomIn. Recall the definition of *displacement* in Section 2. The function ZoomIn takes as input a box $I \times J$, and a subinterval I' of I and some additional parameters, and outputs a set of suitably aligned intervals J' of width $\mu(I')$ so that each box $I' \times J'$ has small displacement from $I \times J$. More precisely, for a box $I \times J$, a subinterval $I' \subseteq I$, and $0 \le i' \le i \le \log(1/\theta)$, ZoomIn$(j, I \times J, i, I', i')$ is the set of all $\varepsilon(i'+3)$-aligned intervals $J' \subseteq J$ of width $\mu(I')$, for which the displacement of $I' \times J'$ from $I \times J$ is at most $2\varepsilon(i)\mu(I)$.

Proposition 4.2. *Let I be an interval of width w and $I' \subseteq I$ of width w' a divisor of w. Let $i' \le i \in \{0,\dots,\log(1/\theta)\}$.*

1. *For J of width w, $|ZoomIn(j, I \times J, I', i')|$ has size at most $1 + 32\varepsilon(i-i')w/w'$.*
2. *Let $I' \times J'$ be a box. The number of $\varepsilon(i+3)$-aligned width-w intervals J such that $J' \in ZoomIn(j, I \times J, i, I', i')$ is at most 33.*

Proof. Set $\Delta = \min(I') - \min(I)$. If $J' \in$ ZoomIn$(j, I \times J, i, I', i')$ then $|\min(J') - \Delta - \min(J)| \le 2\varepsilon(i)w$.

Proof of (1). Holding J fixed, we have $\min(J') \in [\min(J) + \Delta - 2\varepsilon(i)w, \min(J) + \Delta + 2\varepsilon(i)w]$. This is an interval of width $4\varepsilon(i)w$, and the number of $\varepsilon(i'+3)$-aligned intervals of width w' that start in this interval is at most $1 + 32\varepsilon(i-i')w/w'$.

Proof of (2). Holding J' fixed, we have $\min(J) \in [\min(J') - \Delta - 2\varepsilon(i)w, \min(J') - \Delta + 2\varepsilon(i)w]$. This is an interval of width $4\varepsilon(i)w$, and the number of $\varepsilon(i+3)$-aligned intervals of width w that start in this interval is at most 33. □

Calling ZoomIn$(j, I \times \mathcal{J}, i, I', i')$ with a stack $I \times \mathcal{J}$ returns the union of results $\bigcup_{J \in \mathcal{J}}$ ZoomIn$(j, I \times J, i, I', i')$.

The function InducedBoxes. This is a function that takes as input a set of weighted square boxes $\mathcal{Q}$ and outputs a collection of weighted boxes *induced by* $\mathcal{Q}$. For an interval J, and $t \le \mu(J)/2$ let $J/[t]$ denote the interval $[\min(J) + t, \max(J) - t]$. For each $(I \times J, \kappa)$ in $\mathcal{Q}$, InducedBoxes$(\mathcal{Q})$ includes $(I \times J, \kappa)$ together with boxes of the form $(I \times J/[2^i], \kappa + \frac{2^{i+1}}{\mu(I)})$ for $i \in \{0,\dots,\log(\mu(J)) - 1\}$.

Proposition 4.3. *If all boxes of $\mathcal{Q}$ are certified boxes then so are all boxes of InducedBoxes($\mathcal{Q}$).*

Proof. Note that $|J\Delta(J/[2^i])| = 2^{i+1}$ and apply Proposition 2.2. □

The function APM (Approximate pattern match). Recall from Section 2 that $\text{cost}_{\mathcal{R}}(I \times J)$ is the length of the min-cost traversal of $I \times J$ in the shortcut graph $\widetilde{G}(\mathcal{R})$. APM takes as input a stack $I \times \mathcal{J}$, $\kappa > 0$ and a set $\mathcal{R}$ of certified boxes, and outputs a subset $\mathcal{S}$ of $\mathcal{J}$ that satisfies:

Completeness of APM. For all $J \in \mathcal{J}$ satisfying $\text{cost}_{\mathcal{R}}(I \times J) \leq \kappa\mu(I)$, $J \in \mathcal{S}$

Soundness of APM. For all $J \in \mathcal{J}$ satisfying $\text{cost}(I \times J) > 2\kappa\mu(I)$, $J \notin \mathcal{S}$.

The running time is $\widetilde{O}(\mu(I) + |\mathcal{J}| + |\mathcal{R}|)$. (Notice, the subtle distinction between $\text{cost}_{\mathcal{R}}$ and cost in Soundness and Completeness.) The implementation, described in Section 6, is a customized variant of dynamic programming that closely follows [16, 17].

4.4 The mechanics of the algorithm

We are now ready to present the pseudocode for FAST-GED and the three main subroutines: Preprocess and ProcessDense, and EnumerateClose.

Algorithm 1 FAST-GED($n, \theta; x, y$)

Input: n is a power of 2. $|x| = |y| = n$. $\theta \in (0, 1]$ is a power of 1/2.
Output: If $\Delta_{\text{edit}}(x, y) \geq Q\theta$ then return REJECT. If $\Delta_{\text{edit}}(x, y) \leq \theta$ then return ACCEPT with probability at least 1/2.

```
1: $z \longleftarrow xy$.
2: for $j \in \{1, \ldots, k\}$ do
3:     Preprocess($j$).
4:     ProcessDense($j$).
5: end for
6: Return ACCEPT if APM($\{0, \ldots, n\} \times \{\{n, \ldots, 2n\}\}, \theta 2^{q_k+5}, \mathcal{R}(k)$) is non-empty, otherwise re-
   turn REJECT.
```

The algorithm FAST-GED. This algorithm inputs an integer n which is a power of 2, $\theta \in (0, 1]$ a power of 1/2, and two strings x, y of length n, and returns ACCEPT or REJECT. (Recall that the error parameter δ is fixed to 1/2.) The algorithm consists of iterations indexed by $j \in \{1, \ldots, k\}$. Preprocess(j) creates the sets $\mathcal{B}^{\text{below}}(j, I, i)$ and SparseSample(j, I, i) that enable the level j queries EnumerateClose($j, \cdot$). ProcessDense(j) creates sets $\mathcal{R}(j)$ and Sparse(j, i) needed for Preprocess(j + 1).

Algorithm 2 Preprocess(j)

Input: $j \in [k]$. Levels $j = 1, \dots, j-1$ were already processed. Uses sets Sparse$(j-1, i)$ and $\mathcal{R}(j-1)$ constructed by ProcessDense$(j-1)$, where $\mathcal{R}(0) = \emptyset$.
Output: $\mathcal{B}^{\text{below}}(j, I, i)$, and SparseSample$(j, I, i)$ for $(I, i) \in \text{Intervals}(w_j) \times \{0, \dots, \log(1/\theta)\}$.

```
1:  Partition $\mathcal{R}(j-1)$ into $\{\mathcal{R}(j-1, I) : I \in \text{Intervals}(w_j)\}$ where $I' \times J'$ is in $\mathcal{R}(j-1, I)$ if $I' \subseteq I$.

2:  for $i \in \{0, \dots, \log 1/\theta\}$ do
3:    for $I \in \text{Intervals}(w_j)$ do
4:      if $j = 1$ then
5:        SparseSample$(j, I, i) \longleftarrow \emptyset$; $\mathcal{B}^{\text{below}}(j, I, i) \longleftarrow \emptyset$
6:      else
7:        if $\text{Intervals}(w_{j-1}; I) \cap \text{Sparse}(j-1, i) = \emptyset$ then
8:          SparseSample$(j, I, i) \longleftarrow \emptyset$
9:        else
10:         Make $30 \log n$ independent uniform selections from $\text{Intervals}(w_{j-1}; I) \cap \text{Sparse}(j-1, i)$
            to obtain SparseSample$(j, 1, i)$
11:       end if
12:       $\mathcal{B}^{\text{below}}(j, I, i) \longleftarrow \text{APM}(I \times \text{Intervals}(w_j, \varepsilon(i+3)), \varepsilon(i - q_{j-1} - 5), \mathcal{R}(j-1, I))$
13:     end if
14:   end for
15: end for
```

The subroutine Preprocess. On input j, the sets Sparse$(j-1, i)$ and $\mathcal{R}(j-1)$ created by ProcessDense$(j-1)$ are used to produce the sets $\mathcal{B}^{\text{below}}(j, I, i)$ and SparseSample(j, I, i) for $I \in$ Intervals(w_j) and $i \in \{0, \dots, \log(1/\theta)\}$. To begin, the set of weighted w_{j-1}-boxes $\mathcal{R}(j-1)$ is partitioned into sets $\mathcal{R}(j-1, I)$, with $I' \times J'$ assigned to $\mathcal{R}(j-1, I)$ for $I' \subseteq I$. For each i and I:

1. The set Sparse$(j-1, i) \subseteq$ Intervals(w_{j-1}) was produced by ProcessDense$(j-1)$. SparseSample$(j, I, i) = \emptyset$ if Sparse$(j-1, i)$ contains no subintervals of I, and otherwise is an independent random sample (multiset) of size $\log(n)^{\theta(1)}$ selected from the subsets of I belonging to Sparse$(j-1, i)$.

2. Run APM with input stack $I \times$ Intervals$(w_j, \varepsilon(i+3))$ and $\mathcal{R}(j-1, I)$ to determine the set of intervals $J \in$ Intervals$(w_j, \varepsilon(i+3))$ that are suitably close to I in the shortcut graph $\widetilde{G}(\mathcal{R}(j-1, I))$.

The subroutine EnumerateClose. The creation of $\mathcal{B}^{\text{below}}(j, I, i)$ and SparseSample(j, I, i) by Preprocess enables the query EnumerateClose$(j, \cdot)$, which implicitly classifies all level j candidates $\langle i; I \times J \rangle$ as CLOSE or FAR subject to:

Completeness of EnumerateClose. If ncost$(I \times J) \le \varepsilon(i)$ then with high probability $\langle i; I \times J \rangle$ is CLOSE.

Soundness of EnumerateClose. If ncost$(I \times J) > \varepsilon(i - q_{j-1} - 6)$ then $\langle i; I \times J \rangle$ is FAR.

Algorithm 3 EnumerateClose($j, I \times \mathcal{J}, i$)

Input: $j \in [k]$, $I \in \text{Intervals}(w_j)$, $\mathcal{J} \subseteq \text{Intervals}(w_j, \varepsilon(i+3))$, $i \in \{0, \dots, \log(1/\theta)\}$, levels $1, \dots, j-1$ were already processed, and level j was preprocessed.
Output: Returns $\mathcal{S} = \{J \subseteq \mathcal{J} : \langle i; I \times J \rangle \text{ is CLOSE}\}$

```
if $j = 1$ then
    Initialization: $\mathcal{S} \longleftarrow \emptyset$.
    for $J \in \mathcal{J}$ do
        if SLOW-GED($z_I, z_J, \varepsilon(i)$) returns ACCEPT then
            Add $J$ to $\mathcal{S}$.  // $\langle i; I \times J \rangle$ is classified as CLOSE //
        end if
    end for
else
    // $j > 1$ //
    Initialization: $\mathcal{S} \longleftarrow \mathcal{B}^{\text{below}}(j, I, i) \cap \mathcal{J}$, $\mathcal{J} \longleftarrow \mathcal{J} \setminus \mathcal{S}$, $\mathcal{K} \longleftarrow \emptyset$.
    for $i' \in \{0, \dots, i\}$ do
        for $I' \in \text{SparseSample}(j, I, i')$ do
            $\mathcal{J}' \longleftarrow \text{ZoomIn}(j, I \times \mathcal{J}, i, I', i')$.
            $\mathcal{S}' \longleftarrow \text{EnumerateClose}(j-1, I' \times \mathcal{J}', i')$.
            for $J \in \mathcal{J}$ do
                if $\text{ZoomIn}(j, I \times J, i, I', i') \cap \mathcal{S}' \neq \emptyset$ then
                    Add $J$ to $\mathcal{K}$
                end if
            end for
        end for
    end for
    for $J \in \mathcal{K}$ do
        if SLOW-GED($z_I, z_J, \varepsilon(i)$) returns ACCEPT then
            Add $J$ to $\mathcal{S}$.  // $\langle i; I \times J \rangle$ is classified as CLOSE //
        end if
    end for
end if
Return $\mathcal{S}$.
```

EnumerateClose($j, \cdot$) takes a stack $I \times \mathcal{J}$ and $i \in \{0, \dots, \log(1/\theta)\}$ with $I \in \text{Intervals}(w_j)$ and $\mathcal{J} \subseteq \text{Intervals}(w_j, \varepsilon(i+3))$ and returns $\{J \in \mathcal{J} : \langle i; I \times J \rangle$ is **CLOSE**$\}$. $\mathcal{S}$ accumulates the set of intervals to be output. For $j = 1$, SLOW-GED(z_I, z_J, ϵ) is run for each $J \in \mathcal{J}$ and $\mathcal{S}$ is the set of accepted J. For $j > 1$, $\mathcal{S}$ is the union of two sets. The first is $\mathcal{B}^{\text{below}}(j, I, i) \cap \mathcal{J}$ found by Preprocess(j). The second is obtained by identifying (as described below) a small subset $\mathcal{K} \subseteq \mathcal{J}$, testing each $J \in \mathcal{K}$ using SLOW-GED, and adding J to $\mathcal{S}$ if z_J is suitably close to z_I. To identify $\mathcal{K}$, for each $(I', i') \in \text{SparseSample}(j, I, i) \times \{0, \dots, i\}$ use ZoomIn to identify the set $\mathcal{J}'$ of $J' \in \text{Intervals}(w_{j-1}, \varepsilon(i'+3))$ such that $I' \times J'$ has displacement at most $2\varepsilon(i)\mu(I)$ from $I \times J$. Recursively use EnumerateClose($j-1, I' \times \mathcal{J}', i'$) to select $\mathcal{S}' = \{J' \in \mathcal{J}' : \langle i'; I' \times J' \rangle$ is CLOSE$\}$. $\mathcal{K}$ consists of those J for which $I \times J$ has small displacement from $I' \times J'$ for some $J' \in \mathcal{S}'$.

The loops on i', I' (line 11-21) produce $\mathcal{K} \subseteq \mathcal{J}$. For each $J \in \mathcal{K}$, SLOW-GED is run on z_I, z_J. The loop on I' is over SparseSample(j, I, i'). The subset $\mathcal{K}$ of $\mathcal{J}$ depends on the random sample

SparseSample(j, I, i') of Sparse$(j-1, i') \cap$ Intervals$(w_{j-1}; I)$. The following definitions highlight this dependence.

- For $\langle i; I \times J \rangle$, let $I' \in$ Intervals$(w_{j-1}; I)$ and $i' \in \{0, \ldots, i\}$. The pair (I', i') is a *marker*[2] *for the candidate* $\langle i; I \times J \rangle$ if $I' \in$ Sparse$(j-1, i')$ and there is some $J' \in$ ZoomIn$(j, I \times J, i, I', i')$ such that $\langle i'; I' \times J' \rangle$ is classified as **CLOSE**. When lines (13-18) are executed for a marker (I', i'), J is added to $\mathcal{K}$ in line 17. Ideally, $\mathcal{K}$ will consist of all intervals J identifiable by their markers.
- $\mathcal{M}(j, I \times J, i, i') = \{I' \in \text{Sparse}(j-1, i') \cap \text{Intervals}(w_{j-1}; I) : (I', i') \text{ is a marker for } \langle i; I \times J \rangle\}$. We will be interested in situations when for some $i' \leq i$ there will be many markers, namely, $|\mathcal{M}(j, I \times J, i, i')| \geq \frac{1}{3}|\text{Sparse}(j-1, i') \cap \text{Intervals}(w_{j-1}; I)|$, so that with high probability SparseSample$(j-1, I, i')$ will contain a marker that will identify J.

The procedure ProcessDense. This takes as input a level number j. The procedure corresponds closely to the procedure Dense Strip Removal in [16].

For each $i \in \{0, \ldots, \log(1/\theta)\}$ the procedure builds a set Sparse$(j, i) \subseteq$ Intervals(w_j) and also builds sets $\mathcal{B}^{\text{dense}}(j, I, i) \subseteq$ Intervals$(w_j, \varepsilon(i+3))$ for every $I \in$ Intervals$(w_j) \setminus$ Sparse(j, i). This is done by processing the intervals of Intervals(w_j); when interval I is processed it is either assigned to Sparse(j, i) or the set $\mathcal{B}^{\text{dense}}(j, I, i)$ is constructed. We keep track of a subset $\mathcal{T} \subseteq$ Intervals(w_j) of *unprocessed* intervals. This set is initialized to Intervals(w_j) and the iteration ends when $\mathcal{T} = \emptyset$. We proceed in rounds. In a round we select an arbitrary I from $\mathcal{T}$. We perform a test (see "Testing potential pivots in ProcessDense" in Section 4.5) to decide whether to put it in Sparse(j, i). If I is not placed in Sparse(j, i) then I is designated the *pivot* for that round. We then call EnumerateClose on the stack $I \times \mathcal{T}$ (with suitable parameters) to determine the subset $\mathcal{X}$ of Intervals(w_j), we call EnumerateClose on the stack $I \times$ Intervals(w_j, κ) (for a suitable $\kappa \geq \varepsilon(i)$) to determine $\mathcal{Y}' \subseteq$ Intervals(w_j, κ) and we let $\mathcal{Y}$ be the set of intervals from Intervals$(w_j, \varepsilon(i+3))$ which round to an interval in $\mathcal{Y}'$. We then define $\mathcal{B}^{\text{dense}}(j, I', i) = \mathcal{Y}$ for all $I' \in \mathcal{X}$, and remove $\mathcal{X}$ from $\mathcal{T}$, to complete the round.

The parameters used in the above calls are expressed in terms of h_1 and h_2 introduced in the pseudocode. The particular choice h_1 and h_2 is motivated by both the correctness analysis and the time analysis (Section 4.9).

In the sequel, we will need the following definition and observation.

Approved Candidate. A candidate $\langle i; I \times J \rangle$ is said to be *approved* if $I \notin$ Sparse(j, i) and $J \in \mathcal{B}^{\text{dense}}(j, I, i)$. Note that the boxes in $\mathcal{Q}(j)$ are in one-to-one correspondence with the approved candidates, with $(I \times J, \varepsilon(i - q_j)) \in \mathcal{Q}(j)$ if and only if $\langle i; I \times J \rangle$ is approved. All candidates of the form $\langle i; I \times J \rangle$ are approved for $i \leq q_j$.

Proposition 4.4. *At level k, the sets Sparse(k, i) are empty for all $i \in \{0, \ldots, \log(1/\theta)\}$.*

Proof. Since $d_k = 1$, the set $\mathcal{S}$ created in line (10) is all of Intervals$(w_j, \varepsilon(i+3))$ which, in particular includes I. The set returned by EnumerateClose in line (11) includes I and so the if condition fails, and I is not added to Sparse(k, i). □

[2]We call it *marker* as in genomics, where a short DNA sequence identifies a gene. Similarly here, a marker for z_I is its substring $z_{I'}$ which is relatively rare in z, i.e., I' belongs to Sparse$(j-1, i')$.

Algorithm 4 ProcessDense(j)

Input: $j \in [k]$. Levels $1, \dots, j-1$ were already processed, and level j was preprocessed.
Output: For each $i \in \{0, \dots, \log(1/\theta)\}$, specify the set $\text{Sparse}(j,i) \subseteq \text{Intervals}(w_j)$ and specify the sets $\mathcal{B}^{\text{dense}}(j, I, i)$ for all intervals $I \in \text{Intervals}(w_j) \setminus \text{Sparse}(j,i)$.

```
1: for $i = 0, \dots, \log 1/\theta$ do
2:   Initialization: $\mathcal{T} = \text{Intervals}(w_j)$.
3:   Initialization: $\text{Sparse}(j,i) = \emptyset$.
4:   if $i \le q_j$ then
5:     Set $\mathcal{B}^{\text{dense}}(j,I,i) = \text{Intervals}(w_j, \varepsilon(i+3))$ for every $I \in \mathcal{T}$.
6:   else
7:     // $i > q_j$ //
8:     while $\mathcal{T}$ is non-empty do
9:       Pick $I \in \mathcal{T}$.
10:      Let $\mathcal{S}$ be the subset of $\text{Intervals}(w_j, \varepsilon(i+3))$ obtained by including each element inde-
         pendently with probability $p := \min(1, (c_0 \log n)/d_j)$.
11:      if $\text{EnumerateClose}(j, I \times \mathcal{S}, i)$ has less than $p \cdot d_j$ elements then
12:        Add $I$ to $\text{Sparse}(j,i)$, and $\mathcal{T} = \mathcal{T} \setminus \{I\}$. // $I$ is declared sparse. //
13:      else
14:        // $I$ is declared dense and used as a pivot. //
15:        $h_1 \longleftarrow i - q_{j-1} - 7$; $h_2 \longleftarrow i - 2q_{j-1} - 14$. // Since $i > q_j$, $h_1, h_2 > 0$. //
16:        $\mathcal{X} \longleftarrow \text{EnumerateClose}(j, I \times \text{Intervals}(w_j), h_1)$.
17:        $\mathcal{Y}' \longleftarrow \text{EnumerateClose}(j, I \times \text{Intervals}(w_j, \varepsilon(h_2+3)), h_2)$.
18:        $\mathcal{Y} \longleftarrow \{J \in \text{Intervals}(w_j, \varepsilon(i+3)) : \text{Round}(J, \varepsilon(h_2+3)) \in \mathcal{Y}'\}$.
19:        $\mathcal{B}^{\text{dense}}(j, I', i) \longleftarrow \mathcal{Y}$ for each $I' \in \mathcal{X}$.
20:        $\mathcal{T} \longleftarrow \mathcal{T} \setminus \mathcal{X}$.
21:      end if
22:    end while
23:  end if
24: end for
25: // Convert stack of CLOSE boxes into weighted boxes. //
26: $\mathcal{Q}(j) \longleftarrow \{(I \times J, \varepsilon(i - q_j)) : i \in \{0, \dots, \log(1/\theta)\}, I \in \text{Intervals}(w_j), J \in \mathcal{B}^{\text{dense}}(j,I,i)\}$.
27: $\mathcal{R}(j) \longleftarrow \text{InducedBoxes}(\mathcal{Q}(j))$.
```

4.5 The use of randomization

Randomization is used in three parts of the algorithm: the subroutine SLOW-GED, the construction of SparseSample during Preprocess and in ProcessDense, each time we test a selected $I \in \mathcal{T}$ to decide whether it is a pivot. We discuss each of these uses below.

The subroutine SLOW-GED. SLOW-GED takes calling parameters $(n', \theta', \delta'; x', y')$. By our assumption δ' is fixed to n^{-12} for all calls. The gap-soundness and completeness conditions for SLOW-GED guarantee that if $\Delta_{\text{edit}}(x', y') > Q'\theta' n'$ then SLOW-GED returns REJECT, and if $\Delta_{\text{edit}}(x', y') \le \theta' n'$ then SLOW-GED returns ACCEPT with probability at least $1 - n^{-12}$. Say that an execution of SLOW-GED$(n', \theta', n^{-12}; x', y')$ fails if $\Delta_{\text{edit}}(x', y') \le \theta' n'$ and SLOW-GED returns REJECT. We will introduce an event **SG** that no call to SLOW-GED fails.

To simplify the analysis, we make the following assumption: when we run FAST-GED we pregenerate a single string B_{SG} of b random bits where b is an upper bound on the number of random bits used in any call to SLOW-GED. In every call to SLOW-GED we use (a prefix of) B_{SG} to provide the random bits for the call. This makes all calls to SLOW-GED deterministic, and also ensures that if the algorithm makes multiple calls to SLOW-GED with the same input parameters then all such calls yield the same output.

Reusing random bits for different calls of SLOW-GED makes these calls dependent, but this is irrelevant to the analysis. The proof of correctness relies only on the fact that the event **SG** holds.

We now upper bound the probability that there is a call that does not succeed. Every possible input tuple $(n', \theta', n^{-12}; x', y')$ for SLOW-GED satisfies that n' is a power of 2 with $n' < n$, x', y' are substrings of $z = xy$ of length n', and θ' is an integral power of $1/2$. We may assume that $\theta' \geq 1/n$ since for $\theta' < 1/n$ we may assume that SLOW-GED is the deterministic algorithm that returns **ACCEPT** if $x = y$ and **REJECT** otherwise. Let **SG** denote the event that for all possible choices of input parameters (n', x', y', θ') with $\theta' \geq 1/n$, the choice of random bits succeeds.

The number of possible choices of input parameters for which randomness is used is at most $4n^2 \log^2(n)$. (There are at most $\log(n)$ ways to choose n', and to choose θ', and at most $2n$ ways to choose the starting location of x' and of y'.) Thus by a union bound, the probability that **SG** does not hold is at most n^{-8}.

The construction of SparseSample. SparseSample(j, I, i) is a random sample of Sparse$(j-1, i)$ generated during Preprocess(j). What we want from this sample is that for each $i' \in \{0, \ldots, i\}$, if a nontrivial fraction of Sparse$(j-1, i)$ belongs to the set of markers $\mathcal{M}(j, I \times J, i, i')$ then SparseSample(j, I, i) should include a member of $\mathcal{M}(j, I \times J, i, i')$. (Note: for the purposes of this discussion, the exact technical definition of $\mathcal{M}(j, I \times J, i, i')$ is unimportant, we only need that for each j, I, J, i, i', $\mathcal{M}(j, I \times J, i, i')$ and Sparse$(j-1, i)$ are completely determined after iteration $j-1$, and $\mathcal{M}(j, I \times J, i, i') \subseteq$ Sparse$(j-1, i)$.) Formally, we say that SparseSample(j, I, i) *fails* for $J \in$ Intervals$(w_j, \varepsilon(i+3))$ and $i' \in \{0, \ldots, i\}$ if $|$Sparse$(j-1, i) \cap$ Intervals$(w_{j-1}; I)| > 0$, $|\mathcal{M}(j, I \times J, i, i')| \geq |$Sparse$(j-1, i) \cap$ Intervals$(w_{j-1}; I)|/3$ and SparseSample$(j, I, i) \cap \mathcal{M}(j, I \times J, i, i') = \emptyset$. Since $\mathcal{M}(j, I \times J, i, i')$ is completely determined by the end of iteration $j-1$, and SparseSample(j, I, i) is an independent sample of $30 \log n$ elements from Sparse(j, I, i) selected during iterations j, the probability that SparseSample(j, I, i) fails for J, i' is at most $(1 - 1/3)^{30 \log n} \leq n^{-10}$. There are at most n pairs J, i' so the probability that SparseSample(j, I, i) fails for some J, i' is at most n^{-9}. There are at most n triples j, I, i so the probability that some SparseSample(j, I, i) fails is at most n^{-8}. We denote by B^j_{SS} the random bits that are used at iteration j to generate the samples from Sparse(j, I, i) for all I and i.

Testing potential pivots in ProcessDense. During the while loop for $I \in \mathcal{T}$ of ProcessDense, we make a random selection of a set $\mathcal{S}$, and this choice affects whether I is assigned to Sparse(j, i) or becomes a pivot. The constant c_0 in line (10) is chosen below to satisfy certain technical conditions. We denote by B^j_{PD} the random bits used at iteration j to generate sets $\mathcal{S}$ where we make the simplifying assumption that there is a designated block of bits for each possible $I \in$ Intervals(w_j) and i to select the corresponding $\mathcal{S}$. (Some of the blocks might be unused.)

There are two bad events that depend on the choice of $\mathcal{S}$:

1. $|$EnumerateClose$(j, I \times$ Intervals$(w_j, \varepsilon(i+3)), i)| < d_j/2$ and I is not assigned to Sparse(j, i).

2. $|$EnumerateClose$(j, I \times$ Intervals$(w_j, \varepsilon(i+3)), i)| > 2d_j$ and I is assigned to Sparse(j, i).

For both of the bad events, we observe that (i) for any input $(j, I \times \mathcal{J}, i)$, EnumerateClose$(j, I \times \mathcal{J}, i)$ returns the stack of candidates $\langle i; I \times J\rangle$ that are classified as CLOSE among $I \times \mathcal{J}$, and (ii) the classification of level j candidates as CLOSE or FAR is completely deterministic given the random bits B_{SG} for SLOW-GED, and the random bits $B_{\mathrm{SS}}^{\leq j}$ and $B_{\mathrm{PD}}^{\leq j-1}$ for the first $j-1$ iterations and Preprocess(j). Thus, for the random sample $\mathcal{S}$ of Intervals$(w_j, \varepsilon(i+3)), i)$, where each interval is placed in $\mathcal{S}$ independently with probability p, $\frac{1}{p}|\mathrm{EnumerateClose}(j, I \times \mathcal{S}, i)|$ is an estimate of $|\mathrm{EnumerateClose}(j, I \times \mathrm{Intervals}(w_j, \varepsilon(i+3)), i)|$, and the bad events can only occur if this estimate is sufficiently inaccurate. For suitably large c_0, a simple Chernoff-Hoeffding bound shows that for each (I, i) the probability of a bad event is at most n^{-10}, and summing over the at most $O(n)$ such pairs, the probability of a bad event is at most n^{-9}. We say ProcessDense has successful sampling if no such bad event occurs.

Successful randomization. An execution of FAST-GED has *successful randomization* if all calls to SLOW-GED are correct, all calls to SparseSample are successful, and ProcessDense has successful sampling. We denote the event of successful randomization by **SR**. By the above, $\Pr[\mathbf{SR}] \geq 1 - 1/n^7$.

4.6 The properties enforced by FAST-GED.

In this section we state and prove a theorem that states the main properties enforced by FAST-GED. By hypothesis, SLOW-GED is a gap algorithm for edit distance satisfying gap-condition(T', ζ', Q'). We want to show thatFAST-GED satisfies gap-condition(T, ζ, Q) with $T = T' + 1/6$ and suitably chosen $\zeta > 0$ and $Q \geq 1$ (depending only on T',ζ' and Q'). As in the discussion in Section 4.2, we say that the level j candidate $\langle i; I \times J\rangle$ is classified as **CLOSE** if EnumerateClose$(j, I \times \{J\}, i)$ returns $\{J\}$ and is classified as **FAR** if EnumerateClose$(j, I \times \{J\}, i)$ returns $\emptyset$.

Theorem 4.5. *Assume that SLOW-GED is a gap algorithm for edit distance satisfying gap-condition(T', ζ', Q'). Consider a run of FAST-GED on input $(n, \theta, 1/2; x, y)$ where $n^{-\zeta'} \leq \theta \leq 1$ and $|x| = |y| = n$, that meets the conditions for successful randomization.*

For all $j \in \{1, \ldots, k\}$, $i \in \{0, \ldots, \log(1/\theta)\}$, $I \in$ Intervals(w_j), $J \in$ Intervals$(w_j, \varepsilon(i+3))$, $\mathcal{J} \subseteq$ Intervals$(w_j, \varepsilon(i+3))$:

Soundness of $\mathcal{B}^{\mathbf{below}}$. *If $J \in \mathcal{B}^{below}(j, I, i)$ then ncost$(I \times J) \leq \varepsilon(i - q_{j-1} - 6)$.*

Completeness of $\mathcal{B}^{\mathbf{below}}$. *If ncost$(I \times J) \leq \varepsilon(i)$ then (i) $J \in \mathcal{B}^{below}(j, I, i)$ or (ii) there exists an $i' \leq i$ such that $|\mathcal{M}(j, I \times J, i, i')| > \frac{1}{3}|$Intervals$(w_{j-1}; I) \cap$ Sparse$(j-1, i')|$.*

Consistency of EnumerateClose. *$J \in$ EnumerateClose$(j, I \times \mathcal{J}, i)$ if and only if $J \in$ EnumerateClose$(j, I \times \{J\}, i)$. If $J \in$ EnumerateClose$(j, I \times \{J\}, i)$ then $\langle i; I \times J\rangle$ is classified as* **CLOSE**.

Soundness of EnumerateClose. *If $\langle i; I \times J\rangle$ is classified as* **CLOSE** *then ncost$(I \times J) \leq \varepsilon(i - q_{j-1} - 6)$.*

Completeness of EnumerateClose. *If ncost$(I \times J) \leq \varepsilon(i)$ then $\langle i; I \times J\rangle$ is classified as* **CLOSE**.

Validity of Sparse. *$I \in$ Sparse(j, i) implies that EnumerateClose$(j, I \times$Intervals$(w_j, \varepsilon(i+3)))$ has size at most $2d_j$.*

Soundness of $\mathcal{B}^{\mathbf{dense}}$. *If $J \in \mathcal{B}^{dense}(j, I, i)$ then ncost$(I \times J) \leq \varepsilon(i - q_j)$.*

Completeness of $\mathcal{B}^{\mathbf{dense}}$. *If $I \notin$ Sparse(j, i) and ncost$(I \times J) \leq \varepsilon(i)$ then $J \in \mathcal{B}^{dense}(j, I, i)$.*

Soundness of $\mathcal{R}(j)$. *Every box in $\mathcal{R}(j)$ is correctly certified, i.e., $(I\times J,\kappa)\in\mathcal{R}(j)$ implies ncost$(I\times J)\leq\kappa$.*

Completeness of $\mathcal{Q}(j)$. *If $I\notin$ Sparse(j,i) and ncost$(I\times J)\leq\varepsilon(i)$ then $(I\times J,\min(1,\varepsilon(i-q_j)))\in\mathcal{Q}(j)$*

The proof of this theorem is by induction on j. For fixed j when we prove a property we assume the properties listed above it hold. With the exception of the Completeness of $\mathcal{B}^{\text{below}}$, which we defer to the next subsection, the proofs are straightforward.

Proof of Soundness of $\mathcal{B}^{\text{below}}$. For $j=1$, the requirement is vacuously satisfied. Suppose $j>1$. By Soundness of $\mathcal{R}(j-1)$, every box in $\mathcal{R}(j-1,I)$ is certified. If $J\in\mathcal{B}^{\text{below}}(j,I,i)$, then the pseudocode implies that $J\in\text{APM}(j,I\times\text{Intervals}(w_j,\varepsilon(i+3),\varepsilon(i-q_{j-1}-5),\mathcal{R}(j-1,I))$. By definition of the soundness of APM, J is included in the output to the call of APM implies that $\text{cost}(I\times J)\leq 2\varepsilon(i-q_{j-1}-5)=\varepsilon(i-q_{j-1}-6)$.

Proof of Completeness of $\mathcal{B}^{\text{below}}$. See subsection 4.7

Proof of Consistency of EnumerateClose. We must show that whether $J\in\text{EnumerateClose}(j,I\times\mathcal{J},i)$ does not depend on $\mathcal{J}\setminus\{J\}$. In the case $j=1$, $J\in\text{EnumerateClose}(i,I\times\mathcal{J},i)$ if and only SLOW-GED$(I\times J)\leq\varepsilon(i)$ returns **ACCEPT** which does not depend on $\mathcal{J}\setminus\{J\}$. Assume $j>1$. From the pseudocode of EnumerateClose, $J\in\text{EnumerateClose}(j,I\times\mathcal{J},i)$ if and only if (i) $J\in\mathcal{B}^{\text{below}}(j,I,i)$ or (ii) $J\in\mathcal{K}$ and SLOW-GED$(z_I,z_J,\varepsilon(i))$ returns **ACCEPT**. Neither condition (i) nor SLOW-GED$(z_I,z_J,\varepsilon(i))$ depend on $\mathcal{J}\setminus\{J\}$. It remains to show that whether $J\in\mathcal{K}$ is also independent of $\mathcal{J}\setminus\{J\}$. Now $J\in\mathcal{K}$ if and only if there exists $i'\in\{0,\dots,i\}$, $I'\in\text{SparseSample}(j,I,i)$ and $J'\in\text{ZoomIn}(j,I\times J,i,I',i')$ such that $J'\in\mathcal{S}'$. The set ZoomIn$(j,I\times J,i,I',i')$ obviously doesn't depend on $\mathcal{J}\setminus\{J\}$. For $J'\in\text{ZoomIn}(j,I\times J,i,I',i')$ we must have $J'\in\mathcal{J}'$, and therefore by the consistency of EnumerateClose at level $j-1$, $J'\in\text{EnumerateClose}(j-1,I'\times\mathcal{J}',i')$ if and only if $J'\in\text{EnumerateClose}(j-1,I'\times\{J'\},i')$.

Proof of Soundness of EnumerateClose. $\langle i;I\times J\rangle$ is classified as **CLOSE** means that $J\in\text{EnumerateClose}(j,I\times\{J\},i)$. Now for this to happen either (i) SLOW-GED$(z_I,z_J,\varepsilon(i))$ returns **ACCEPT**, or (ii) $J\in\mathcal{B}^{\text{below}}(j,I,i)$. If (i) holds then the guarantee on SLOW-GED implies $\text{ncost}(I\times J)\leq Q'\varepsilon(i)\leq\varepsilon(i-q_{j-1}-6)$, since $\log(Q')=q_0\leq q_{j-1}$ for all $j\geq 1$. If (ii) holds then the result follows from the Soundness of $\mathcal{B}^{\text{below}}$.

Proof of Completeness of EnumerateClose. Suppose ncost$(I\times J)\leq\varepsilon(i)$. By the Completeness of $\mathcal{B}^{\text{below}}$, we have (i) $J\in\mathcal{B}^{\text{below}}(j,I,i)$ or (ii) $\text{Sparse}(j-1,i)\cap\text{Intervals}(w_{j-1};I)\neq\emptyset$ and there exists an $i^*\leq i$ so that $|\mathcal{M}(j,I\times J,i,i^*)|\geq\frac{1}{3}|\text{Intervals}(w_{j-1};I)\cap\text{Sparse}(j-1,i^*)|$. If (i) holds, then the definition of EnumerateClose immediately gives $J\in\text{EnumerateClose}(j,I\times\{J\},i)$. If (ii) holds, then the success condition for SparseSample(j,I,i) (from Section 4.5) implies that there is an $I^*\in\text{SparseSample}(j,I,i^*)$ such that (I^*,i^*) is a marker for $\langle i;I\times J\rangle$. During the execution of EnumerateClose$(j,I\times J,i)$, when i^* is selected in line (11) and I^* in line (12), by the definition of marker, J is added to $\mathcal{K}$ in line (17). The correctness of SLOW-GED implies that SLOW-GED$(I\times J,\varepsilon(i))$ will **ACCEPT** in line (23) and so J will be added to $\mathcal{S}$.

Proof of Validity of Sparse. This follows immediately from the assumption that ProcessDense has successful sampling.

Proof of Soundness of $\mathcal{B}^{\text{dense}}$. For $i\leq q_j$ the claim is trivial so we assume $i-q_j>0$. Suppose $J\in\mathcal{B}^{\text{dense}}(j,I,i)$. $\mathcal{B}^{\text{dense}}(j,I,i)$ was defined during iteration i of the main loop (1-34) of

ProcessDense(j), during one of the iterations of the while loop (8-23). Let I^* be the pivot during that iteration. Then $I \in \text{EnumerateClose}(j, I^* \times \text{Intervals}(w_j), h_1)$ and $J' \in \text{EnumerateClose}(j, I^* \times \text{Intervals}(w_j, \varepsilon(h_2+3)), h_2)$, for $J' = \text{Round}(J, \varepsilon(h_2+3))$. By the Soundness of EnumerateClose, $\text{ncost}(I^* \times I) \leq \varepsilon(h_1 - q_{j-1} - 6)$ and $\text{ncost}(I^* \times J') \leq \varepsilon(h_2 - q_{j-1} - 6)$. By the triangle inequality and Propositon 2.2, we have $\text{ncost}(I \times J) \leq \varepsilon(h_1 - q_{j-1} - 6) + \varepsilon(h_2 - q_{j-1} - 6) + \varepsilon(h_2 + 3) \leq 2\varepsilon(h_2 - q_{j-1} - 6) = \varepsilon(h_2 - q_{j-1} - 7) = \varepsilon(i - 3q_{j-1} - 21) = \varepsilon(i - q_j)$.

Proof of Completeness of $\mathcal{B}^{\text{dense}}$. Suppose $I \notin \text{Sparse}(j, i)$ and $\text{ncost}(I \times J) \leq \varepsilon(i)$. Since $I \notin \text{Sparse}(j, i)$ during iteration i of the main loop (1), there is an iteration of the while loop (8-22) of ProcessDense(j) where I was removed from $\mathcal{T}$. Let I^* be the pivot for that iteration. Since I was removed from $\mathcal{T}$, $I \in \mathcal{X}$ during this iteration, so $I \in \text{EnumerateClose}(j, I^* \times \text{Intervals}(w_j), h_1)$ and by the Soundness of EnumerateClose $\text{ncost}(I^* \times I) \leq \varepsilon(h_1 - q_{j-1} - 6)$. Let $J' = \text{Round}(J, \varepsilon(h_2+3))$. It suffices to show that $J \in \mathcal{Y}$ for this same iteration, which would follow from $J' \in \text{EnumerateClose}(j, I^* \times \text{Intervals}(w_j, \varepsilon(h_2+3)), h_2)$. By the Completeness of EnumerateClose it suffices to show that $\text{ncost}(I^* \times J') \leq \varepsilon(h_2)$. By the triangle inequality and Propositon 2.2, $\text{ncost}(I^* \times J') \leq \text{ncost}(I^* \times I) + \text{ncost}(I \times J') \leq \text{ncost}(I^* \times I) + \text{ncost}(I \times J) + \varepsilon(h_2 + 3) \leq \varepsilon(h_1 - q_{j-1} - 6) + \varepsilon(i) + \varepsilon(h_2 + 3) \leq \varepsilon(h_2)/2 + \varepsilon(h_2)/4 + \varepsilon(h_2)/8 \leq \varepsilon(h_2)$ as required.

Proof of Soundness of $\mathcal{R}(j)$. By Proposition 4.3, it suffices that every box in $\mathcal{Q}(j)$ is correctly certified. In line (26) of ProcessDense(j), $(I \times J, \varepsilon(i - q_j)) \in \mathcal{Q}(j)$ only if $J \in \mathcal{B}^{\text{dense}}(j, I, i)$ which is correctly certified by the Soundness of $\mathcal{B}^{\text{dense}}$.

Proof of Completeness of $\mathcal{Q}(j)$. Suppose $I \notin \text{Sparse}(j, i)$ and $\text{ncost}(I \times J) \leq \varepsilon(i)$. By the Completeness of $\mathcal{B}^{\text{dense}}$, $J \in \mathcal{B}^{\text{dense}}(j, I, i)$ and so the definition of $\mathcal{Q}(j)$ implies that $(I \times J, \varepsilon(i - q_j)) \in \mathcal{Q}(j)$.

4.7 Proof of Completeness of $\mathcal{B}^{\text{below}}$

Here we finish the proof of Theorem 4.5, by establishing the final property, whose proof is significantly more involved than that of the others. The proof is based on ideas from [16].

Consider a candidate $\langle i; I \times J \rangle$ with $\text{ncost}(I \times J) \leq \varepsilon(i)$. We assume condition (ii) fails and deduce $\text{cost}_{\mathcal{R}(j-1,I)}(I \times J) \leq \varepsilon(i - q_{j-1} - 5)w_j$. By the definition of Preprocess and the Completeness of APM, this immediately implies condition $J \in \mathcal{B}^{\text{below}}(j, I, i)$, which is condition (i).

Fix a minimum cost traversal τ of $I \times J$. The proof proceeds via the following steps.

Step 1. For each $I' \in \text{Intervals}(w_{j-1}; I)$ we specify a candidate $\langle t(I'); I' \times \hat{J}(I') \rangle$, which is approved in the sense defined in the description of ProcessDense in Section 4.4. (The collection of boxes $\{I' \times \hat{J}(I') : I' \in \text{Intervals}(w_{j-1}; I)\}$ should be thought of as approximatly covering τ.)

Step 2. We upper bound $\text{cost}_{\mathcal{R}(j-1,I)}(I \times J)$ as a constant times $\sum_{I'} \varepsilon(t(I'))w_{j-1}$ plus $8\varepsilon(i)w_j$.

Step 3. We show that if (ii) fails, then $\sum_{I'} \varepsilon(t(I'))w_{j-1}$ can be upper bounded by a constant multiple of $\varepsilon(i)w_j$

Step 4. This gives that $\text{cost}_{cR(j-1)}(I \times J)$ is at most a constant multiple of $\varepsilon(i)w_j$.

Step 1. Specifying $\langle t(I'); I \times \hat{J}(I') \rangle$ for each I'. Consider a pair (I', i') where $i' \in \{0, \dots, i\}$ and $I' \in \text{Intervals}(w_{j-1}; I)$. Proposition 2.4 implies there is a level $j-1$ candidate $\langle i'; I' \times J' \rangle$ such that $\mathrm{ncost}(I' \times J') \leq 2\mathrm{ncost}(\tau_{I'}) + \varepsilon(i'+3)$ and $\mathrm{disp}(I' \times J', \tau_{I'}) \leq \mathrm{cost}(\tau_{I'}) + \varepsilon(i'+3)w_{j-1}$. Select such an interval J' and denote it by $J_{i'}(I')$ (keeping the dependence on τ implicit.)

For each I' let us define $t(I')$ to be the largest index $h \leq i$ for which the candidate $\langle h; I' \times J_i(I')\rangle$ is approved that is $I' \notin \mathrm{Sparse}(j-1,h)$ and $J_{i'}(I') \in \mathcal{B}^{\mathrm{dense}}(j-1,I',h)$. Let $\hat{J}(I') = J_{t(I')}(I')$. We record the important properties:

Proposition 4.6. *For each $I' \in$ Intervals$(w_{j-1}; I)$:*

1. *The box $I' \times \hat{J}(I')$ satisfies ncost$(I' \times \hat{J}(I')) \leq 2$ncost$(\tau_{I'}) + \varepsilon(t(I')+3)$ and disp$(I' \times \hat{J}(I'), \tau_{I'}) \leq$ cost$(\tau_{I'}) + \varepsilon(i'+3)w_{j-1}$.*
2. *The candidate $\langle t(I'); I' \times \hat{J}(I')\rangle$ is approved, and hence $(I' \times \hat{J}(I'), \varepsilon(t(I') - q_{j-1})) \in \mathcal{Q}(j-1)$.*
3. *For any $i' \in \{t(I')+1, \ldots, i\}$ either $I' \in$ Sparse$(j-1,i')$ or $I' \notin$ Sparse$(j-1,i')$ and $J_{i'}(I') \notin \mathcal{B}^{dense}(j-1,I',i')$.*

Proof. The first two properties follow immediately from the definitions of $t(I')$ and $\hat{J}(I')$. For the third property, the maximality of $t(I')$ implies that for $i' \in \{t(I')+1, \ldots, i\}$, $\langle i'; I' \times J_i(I')\rangle$ is not approved, and the result follows from the definition of approved. □

Step 2. Upper bound on $\mathrm{cost}_{\mathcal{R}(j)}(I \times J)$.

Proposition 4.7.

$$\mathit{cost}_{\mathcal{R}(j-1,I)}(I \times J) \leq 8\varepsilon(i)w_j + \sum_{I' \in \mathit{Intervals}(w_{j-1};I)} \varepsilon(t(I') - q_{j-1} - 1)w_{j-1}.$$

(This is closely related to Lemma 4.1 of [16] and the proof is similar.)

Proof. We transform the path τ in G_z to a path τ' in the shortcut graph $\widetilde{G}(\mathcal{R}(j-1,I))$ (see Section 2) and control the increase in cost. Let $I_1, \ldots, I_m$ be the intervals of Intervals$(w_{j-1}; I)$ in order, and for $h \in [m]$, let $i_h = t(I_h)$ and $J_h = \widehat{J}(I_h)$. Let δ_h be the smallest power of 2 such that $\delta_h w_{j-1} \geq \mathrm{disp}(I_h \times J_h, \tau_{I_h})$. By Proposition 4.6, $\delta_h \leq 2\mathrm{ncost}(\tau_{I_h}) + 2\varepsilon(i_h+3)$, and $(I_h \times J_h, \varepsilon(i_h - q_{j-1})) \in \mathcal{Q}(j-1)$. Let $L = \{h \in [m] : \delta_h < 1/2\}$. For $h \in L$, let $J'_h = J_h/[\delta_h w_{j-1}]$ (the interval obtained by removing the first and last $\delta_h w_{j-1}$ indices from J_h). The certified box $(I_h \times J'_h, \varepsilon(i_h - q_{j-1}) + 2\delta_h)$ belongs to $\mathcal{R}(j-1)$, and since $I_h \subseteq I$, it also belongs to $\mathcal{R}(j-1,I)$. Let $e_h = e_{I_h,J'_h}$ be the shortcut edge with cost $(\varepsilon(i_h - q_{j-1}) + 2\delta_h)w_{j-1}$. We claim (1) there is a source-sink path τ' in $\widetilde{G}(\mathcal{R}(j-1,I))$ that consists of $\{e_i : i \in L\}$, plus a collection $\{H_i : i \in [m] \setminus L\}$ where H_i is a horizontal path whose projection to the x-axis is I_i, plus a collection of (possibly empty) vertical paths $V_0, V_1, \ldots, V_m$ where the x-coordinate of V_i for $i > 0$ is $\max(I_i)$ and 0 for V_0, and (2) $\mathrm{cost}(\tau')$ satisfies the bound of the lemma.

For the first claim, for $h \in [m]$, let $p_h = (i_h, j_h)$ be the first point in τ_{I_h} and define p_{m+1} to be the final point of τ. We will define τ' to pass through all of the p_h. Let J^*_h be the vertical projection of τ_{I_h} so that τ_{I_h} traverses $I_h \times J^*_h$. The choice of δ_h implies that for $h \in L$, $J'_h \subseteq J^*_h$. Define the portion τ'_h between p_h and p_{h+1} as follows: if $h \in L$, climb vertically from p_h to $(i_h, \min(J'_h))$ and traverse e_{I_h,J'_h} and climb vertically to p_{h+1} and if $h \notin L$ then move horizontally from p_h to (i_{h+1}, j_h) and then climb vertically to p_{h+1}.

For the second claim, we upper bound $\mathrm{cost}(\tau')$. For $h \in L$, e_{I_h,J_h} has cost at most $(\varepsilon(i_h - q_{j-1}) + 2\delta_h)w_{j-1}$, and for $h \notin L$, the horizontal path that projects to I_h costs $w_{j-1} \leq 2\delta_h w_{j-1}$; the total cost of shortcut and horizontal edges is at most $\sum_h(\varepsilon(i_h - q_{j-1}) + 2\delta_h)w_{j-1}$. The cost of vertical edges is $\sum_{h\in L}(w_{j-1} - \mu(J'_h)) + \sum_{h\notin L} w_{j-1} = \sum_{h\in L} 2\delta_h w_{j-1} + \sum_{h\notin L} w_{j-1} \leq \sum_h 2\delta_h w_{j-1}$.

The combined cost of all edges is at most

$$\begin{aligned}
\sum_h(\varepsilon(i_h - q_{j-1}) + 4\delta_h)w_{j-1} &\leq \sum_h(\varepsilon(i_h - q_{j-1}) + 8\mathrm{cost}(\tau_{I_h}) + 8\varepsilon(i_h+3))w_{j-1} \\
&\leq 8\mathrm{cost}(\tau) + \sum_h(\varepsilon(i_h - q_{j-1}) + \varepsilon(i_h))w_{j-1} \\
&\leq 8\mathrm{cost}(\tau) + \sum_h \varepsilon(i_h - q_{j-1} - 1)w_{j-1},
\end{aligned}$$

which implies the desired bound. □

Step 3. Implication of failure of condition (ii). We now use the failure of (ii) to obtain an upper bound on the righthand side of Proposition 4.7.

For $i' \leq i$, let $\mathcal{M}_{i'} = \mathcal{M}(j, I\times J, i, i')$ and $\mathcal{S}_{i'}$ represent the set $\mathrm{Sparse}(j-1, i') \cap \mathrm{Intervals}(w_{j-1}; I)$. Let $\mathcal{I}' = \mathrm{Intervals}(w_{j-1}; I)$.

The failure of condition (ii) implies:

$$|\mathcal{M}_{i'}| \leq \frac{1}{2}|\mathcal{S}_{i'} \setminus \mathcal{M}_{i'}| \tag{3}$$

.

Multiplying (3) by $\varepsilon(i')$ and summing on i' yields:

$$\sum_{i'\leq i}\sum_{I'\in\mathcal{M}_{i'}} \varepsilon(i') \leq \frac{1}{2}\sum_{i'\leq i}\sum_{I'\in\mathcal{S}_{i'}\setminus\mathcal{M}_{i'}} \varepsilon(i').$$

Switching the sums:

$$\sum_{I'\in\mathcal{I}'}\sum_{i':I'\in\mathcal{M}_{i'}} \varepsilon(i') \leq \frac{1}{2}\sum_{I'\in\mathcal{I}'}\sum_{i':I'\in\mathcal{S}_{i'}\setminus\mathcal{M}_{i'}} \varepsilon(i'). \tag{4}$$

To reduce this further, we need the following sufficient condition for $I' \in \mathcal{M}_{i'}$.

Proposition 4.8. *Suppose the candidate $\langle i; I\times J\rangle$ satisfies ncost$(I\times J) \leq \varepsilon(i)$ and τ is a min-cost traversal of $I\times J$. Let (I', i') be a pair such that $I' \in$ Intervals$(w_{j-1}; I)$ and $i' \in \{0,\ldots,i\}$.*

1. *If $\varepsilon(i') \geq 2$ncost$(\tau_{I'}) + \varepsilon(i+3)$ then ncost$(I' \times J_{i'}(I')) \leq \varepsilon(i')$.*
2. *If $\varepsilon(i') \geq 2$ncost$(\tau_{I'}) + \varepsilon(i+3)$ and $I' \in$ Sparse$(j-1, i')$ then (I', i') is a marker for $\langle i; I\times J\rangle$.*

Proof. For the first part, by the choice of $J_{i'}(I')$, we have $\mathrm{ncost}(I'\times J_{i'}(I')) \leq 2\mathrm{ncost}(\tau_{I'}) + \varepsilon(i+3)$ and by the hypothesis of the Proposition, this is at most $\varepsilon(i')$.

For the second part. By Completeness of EnumerateClose$(j-1,\cdot)$ and the first part, $\langle i'; I'\times J_{i'}(I')\rangle$ is classified as **CLOSE**. So we just have to show that $J_{i'}(I) \in \mathrm{ZoomIn}(j, I\times\{J\}, i, I', i')$. It suffices that $\mathrm{disp}(I'\times J_{i'}(I'), I\times J) \leq 2\varepsilon(i)w_j$. To bound $\mathrm{disp}(I'\times J_{i'}(I'), I\times J)$ it suffices to bound the vertical distance from the point $(\min(I'), \min(J_{i'}(I')))$ to the diagonal of $I\times J$. Let (p,q) be the initial point of $\tau_{I'}$. By the definition of $J_{i'}(I')$), the vertical distance from $(\min(I'), \min(J_{i'}(I')))$ to (p, q) is at most $\mathrm{cost}(\tau_{I'}) + \varepsilon(i' + 3)w_{j-1} \leq \mathrm{cost}(\tau) + w_{j-1}$. By Proposition 2.5 the vertical distance from (p, q) to the diagonal of $I \times J$ is at most $\mathrm{cost}(\tau)/2$. So $\mathrm{disp}(I' \times J_{i'}(I'), I \times J) \leq \frac{3}{2}\mathrm{cost}(\tau) + w_{j-1}$. By hypothesis, $\mathrm{cost}(\tau) \leq \varepsilon(i)w_j$, and by assumption (2) in Section 4.1, $w_{j-1} \leq \frac{\theta}{2}w_j$, and so $\mathrm{disp}(I' \times J_{i'}(I')), I \times J) \leq (\frac{3}{2}\varepsilon(i) + \frac{1}{2}\theta)w_j \leq 2\varepsilon(i)w_j$, as required. □

Let $\mathcal{G}(I) = \{I' \in \mathrm{Intervals}(w_{j-1}; I) : t(I') < i \ \&\ \varepsilon(t(I') + 1) \geq 2\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 3)\}$. We claim that for each $I' \in \mathcal{G}(I)$, $I' \in \mathrm{Sparse}(j - 1, t(I') + 1)$. If it were not then by Part 3 of Proposition 4.6, $J_{i'}(I') \notin \mathcal{B}^{\mathrm{dense}}(j - 1, I', t(I') + 1)$. But by Part 1 of Proposition 4.8 this would contradict completeness of $\mathcal{B}^{\mathrm{dense}}(j - 1, I', t(I') + 1)$. Hence, for each $I' \in \mathcal{G}(I)$, $(I', t(I') + 1)$ is a marker.

We will combine Proposition 4.8 with inequality (4). The sum on the lefthand side of (4) includes all pairs $(I', t(I') + 1)$ where $I' \in \mathcal{G}(I)$ and so is bounded below by $\sum_{I' \in \mathcal{G}(I)} \varepsilon(t(I') + 1)$. To upper bound the righthand sum of (4), we look at the inner sum corresponding to a given $I' \in \mathrm{Intervals}(w_{j-1}; I)$. This is a sum of $\varepsilon(i')$ over those i' such that I' in $\mathrm{Sparse}(j - 1, i')$ and I' not in $\mathcal{M}_{i'}$.

We claim that if i' contributes to this sum then

$$\varepsilon(i') < 2\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 3). \tag{5}$$

To see this note that if $\varepsilon(i') \geq 2\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 3)$ then Part 2 of Proposition 4.8 implies that $I' \notin \mathcal{S}_{i'} \setminus \mathcal{M}_{i'}$, so i' is not included in the sum.

Now in the case that $I' \in \mathcal{G}(I)$ then (5) implies that $\varepsilon(i') < \varepsilon(t(I') + 1)$ and so $\varepsilon(i') \leq \varepsilon(t(I') + 2)$. Summing over all such i', the geometric series is at most $\varepsilon(t(I') + 1)$.

For $I' \notin \mathcal{G}(I)$, let $v(I')$ be the least i' that contributes to the sum. So the sum is at most $2\varepsilon(v(I'))$, and by (5) this is at most $4\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 2)$.

Thus (4) implies:

$$\sum_{I' \in \mathcal{G}(I)} \varepsilon(t(I') + 1) \leq \frac{1}{2} \cdot \left(\sum_{I' \in \mathcal{G}(I)} \varepsilon(t(I') + 1) + \sum_{I' \notin \mathcal{G}(I)} 4\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 2) \right). \tag{6}$$

Multiplying the inequality by 2 and subtracting $\sum_{I' \in \mathcal{G}(I)} \varepsilon(t(I') + 1)$ from both sides gives:

$$\sum_{I' \in \mathcal{G}(I)} \varepsilon(t(I') + 1) \leq \sum_{I' \notin \mathcal{G}(I)} 4\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 2) \tag{7}$$

Now add $\sum_{I' \notin \mathcal{G}(I)} \varepsilon(t(I') + 1)$ to both sides:

$$\sum_{I' \in \mathrm{Intervals}(w_{j-1}; I)} \varepsilon(t(I') + 1) \leq \sum_{I' \notin \mathcal{G}(I)} \varepsilon(t(I') + 1) + \sum_{I' \notin \mathcal{G}(I)} 4\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 2) \tag{8}$$

For the first sum on the right, $I' \notin \mathcal{G}(I)$ implies either $\varepsilon(t(I') + 1) = \varepsilon(i + 1)$ or $\varepsilon(t(I') + 1) < 2\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 3)$, so you can bound this in both cases by $2\mathrm{ncost}(\tau_{I'}) + \varepsilon(i + 1)$. Thus we get:

$$\begin{aligned}
\sum_{I' \in \mathrm{Intervals}(w_{j-1}; I)} \varepsilon(t(I'))w_{j-1} &\leq 2 \cdot \sum_{I' \in \mathrm{Intervals}(w_{j-1}; I)} (6\mathrm{ncost}(\tau_{I'}) + 3\varepsilon(i + 2))w_{j-1} \\
&\leq 12\mathrm{cost}(\tau) + 2\varepsilon(i)w_j \\
&\leq 14\varepsilon(i)w_j.
\end{aligned}$$

Step 4. Combining the bounds. Combining the previous bound with the bound of Proposition 4.7 gives:

$$\begin{aligned}\mathrm{cost}_{\mathcal{R}(j-1,I)}(I\times J) &\leq 14\varepsilon(i-q_{j-1}-1)w_j+\varepsilon(i-3)w_j\\ &\leq \varepsilon(i-q_{j-1}-5)w_j.\end{aligned}$$

as required to establish the Completeness of $\mathcal{B}^{\mathrm{below}}$.

4.8 Correctness of FAST-GED

We now complete the proof that the output of FAST-GED gives a constant factor approximation to edit distance with high probability. As in Theorem 4.5 we assume that SLOW-GED is a gap algorithm for edit distance satisfying gap-condition(T',ζ',Q'). Consider a run of FAST-GED on input $(n,\theta,1/2;x,y)$ where $n^{-\zeta'}\leq\theta\leq 1$ and $|x|=|y|=n$. The conclusion of the theorem has a quality parameter Q which we set to 2^{q_k+6}. We must prove that the FAST-GED satisfies the Soundness and Completeness properties for gap algorithms from Section 1.

The final post-processing step is a call to APM$(\{0,\ldots,n\}\times\{\{n,\ldots,2n\}\},\theta 2^{q_k+5},\mathcal{R}_k)$, and the algorithm returns **ACCEPT** or **REJECT** according to the output of this call. We will apply the Soundness and Completeness of $\mathcal{B}^{\mathrm{below}}$ (with $j=k+1$) by reinterpreting this final step as asking whether $\{n,\ldots,2n\}\in\mathcal{B}^{\mathrm{below}}(k+1,\{0,\ldots,n\},\log(1/\theta)-q_k-6)$ (where $w_{k+1}=n$). The Soundness and Completeness of $\mathcal{B}^{\mathrm{below}}$ extends (with no change) to this case. Thus if the algorithm returns **ACCEPT**, then $\mathrm{ncost}(\{0,\ldots,n\},\{n,\ldots,2n\})\leq\theta 2^{q_k+6}=\theta Q$, and the gap-algorithm satisfies Soundness. For Completeness, assume $\Delta_{\mathrm{edit}}(x,y)\leq\theta$. The Completeness of $\mathcal{B}^{\mathrm{below}}$ extends (with no change) to this case. We conclude that (i) $J\in\mathcal{B}^{\mathrm{below}}(k+1,\{0,\ldots,n\},\log(1/\theta)-q_k-5)$ or (ii) there exists an $i'\leq i$ such that $|\mathcal{M}(k+1,I\times J,i,i')|>\frac{1}{3}|\mathrm{Intervals}(w_k;I)\cap\mathrm{Sparse}(k,i)|$. Since $d_k=1$, Proposition 4.4 implies all sets $\mathrm{Sparse}(k,i)$ are empty, so $\mathcal{M}(k+1,I\times J,i,i')$ are also empty but (ii) requires them to be non-empty. Hence, (ii) can not hold, and so (i) holds, which implies FAST-GED must **ACCEPT**, and so Completeness holds.

4.9 Time analysis

In this subsection, we upper bound the expected running time of FAST-GED conditioned on the event **SR** of successful randomization, in terms of the algorithm parameters $w_1,\ldots,w_k$, and $d_0,\ldots,d_k$. These parameters will be optimized in the next subsection.

Theorem 4.9. *Suppose that SLOW-GED is a gap algorithm for edit distance satisfying gap-condition*(T',ζ',Q'). *For* $\theta\geq n^{-\zeta'}$ *the expected running time of FAST-GED*$(n,\theta,1/2;x,y)$ *conditioned on* **SR** *is upper-bounded by:*

$$\widetilde{O}\left(\sum_{j=1}^{k}\frac{n}{\theta^2 w_j d_j}\left(\sum_{h=1}^{j}d_{h-1}w_h^{1+1/T'}\right)\right). \tag{9}$$

The above theorem is not quite sufficient for our purposes since it gives only an expected upper bound on the running time of the algorithm, while we want an absolute upper bound. We can replace the expected upper bound by an absolute upper bound by the following routine modification of FAST-GED. On a given input, use the above theorem to determine a number τ^* which is at least six times the expected upper bound on running time given by the above theorem. Then the probability that FAST-GED takes more than τ^* steps is at most 1/6. So we run FAST-GED but terminate with **REJECT** if it reaches τ^* steps. This converts the expected running time to an absolute bound on running time, but now the completeness error (the probability of false rejection) is increased from 1/2 to 2/3. But by running this algorithm twice and accepting if either run accepts we restore the completeness error to below 1/2.

Combining the above theorem with this modification gives an algorithm satisfying the correctness properties proved for FAST-GED and having an absolute upper bound on time given as in the above theorem.

We now proceed to the proof of Theorem 4.9.

Proof. Recall from Section 4.5 that successful randomization means: (1) All calls to SLOW-GED return correct answers, (2) All calls to SparseSample are successful and (3) ProcessDense has successful sampling.

Recall that B_{SG} is the sequence of random bits pregenerated for the calls to SLOW-GED (as described in Section 4.5). For $j \in [k]$, B^j_{SS} are the random bits generated to select SparseSample's in Preprocess(j) at iteration j of the algorithm, and B^j_{PD} are the random bits generated to select sets $\mathcal{S}$ in ProcessDense(j) at iteration j (also as described in Section 4.5). Let $B^{\leq j}$ denote the random bits $B_{\text{SG}}, B^1_{\text{SS}}, B^1_{\text{PD}}, \ldots, B^j_{\text{SS}}, B^j_{\text{PD}}$. We introduce the following events:

SG All calls to SLOW-GED return correct answers.

SS(j) All calls to SparseSample during iteration j are successful.

SS($\leq j$) All calls to SparseSample through the end of iteration j are successful.

PD(j) ProcessDense has successful sampling during iteration j.

PD($\leq j$) ProcessDense has successful sample through the end of iteration j.

SR Successful randomization, i.e. $\mathbf{SG} \wedge \mathbf{SS}(\leq k) \wedge \mathbf{PD}(\leq k)$.

We will argue that the expected running time of FAST-GED conditioned on **SR** is bounded by (9). In the bound, the outer sum on j corresponds to iterations of FAST-GED. We will show that the cost of iteration j is bounded by the inner sum. When we analyze iteration j we fix the randomness $B^{\leq j}$ in such a way that $\mathbf{SG} \wedge \mathbf{SS}(\leq j-1) \wedge \mathbf{PD}(\leq j-1)$ holds. The cost of iteration j is bounded conditioned on these fixed random bits and subject to requirement $\mathbf{SS}(j) \wedge \mathbf{PD}(j)$.

As a first step, we need a bound on the running time for EnumerateClose. Recall that fixing the random bits $B^{\leq j-1}$ and B^j_{SS} makes EnumerateClose($j, \cdot$) run deterministically. In the lemma below, we condition on $(B^{\leq j-1}, B^j_{\text{SS}}) = \beta^{*j}$ and consider the expected time of EnumerateClose($j, I \times \mathcal{J}, i$) where $\mathcal{J}$ is a set of intervals chosen according to any distribution (possibly depending on β^{*j}) in which no set appears in $\mathcal{J}$ with probability more than some fixed bound p.

Lemma 4.10. *Let $p \in [0,1]$, $j \in [k]$, $I \in$ Intervals(w_j), and $i \in \{0, \ldots, \log(1/\theta)\}$. Let β^{*j} be an assignment of the random bits $B^{\leq j-1}$ and B^j_{SS} that satisfies the success conditions* $\mathbf{SS}(\leq j)$ *and* $\mathbf{PD}(\leq j-1)$*. Let $\mathcal{J}$ be a random variable whose value is a subset of Intervals$(w_j, \varepsilon(i+3))$ with the property that given the fixed randomness $B^{\leq j-1}$ and B^j_{SS}, each $J \in$ Intervals$(w_j, \varepsilon(i+3))$ belongs to*

$\mathcal{J}$ with probability at most p. Then the expected running time of EnumerateClose$(j, I \times \mathcal{J}, i)$ over the choice of $\mathcal{J}$ is at most:

$$\widetilde{O}(\frac{p}{\theta}\sum_{h=1}^{j} d_{h-1} w_h^{1+1/T'}). \tag{10}$$

Proof. The proof is by induction on j. Suppose $j = 1$. We run SLOW-GED(z_I, z_J, κ) for each $J \in \mathcal{J}$. The expected time is $\widetilde{O}(\frac{1}{\theta}pd_0w_1^{1+1/T'})$ since the expected size of $\mathcal{J}$ is at most $8p\frac{n}{\theta w_1} \le \frac{16p}{\theta}\sqrt{n} \le \frac{32p}{\theta}d_0$ and each call of SLOW-GED costs $w_1^{1+1/T'}$.

Now suppose $j > 1$. The loops on i' and I' starting in lines (11-12) are executed $\widetilde{O}(1)$ times. The construction of $\mathcal{J}'$ in line (13) using ZoomIn takes $\widetilde{O}(|\mathcal{J}'|)$ time (sort $\mathcal{J}$ in the natural order and build $\mathcal{J}'$ "from left to right"). By Proposition 4.2, for each $J' \in$ Intervals$(w_{j-1}, \varepsilon(i'+3))$, the number of $\varepsilon(i+3)$-aligned w_j-intervals J such that $J' \in$ ZoomIn$(j, I \times J, i, I', i')$ is at most 33. Since $\mathcal{J}$ is selected according to a probability distribution so that no set J belongs to $\mathcal{J}$ with probability more than p, $\mathcal{J}'$ is sampled according to some distribution where for each $J' \in$ Intervals$(w_{j-1}, \varepsilon(i'+3))$ the probability of $J' \in \mathcal{J}'$ is at most $33p$. Hence, the expected size of $\mathcal{J}'$ is at most $33p\frac{n}{w_{j-1}\varepsilon(i'+3)} \le O(p\frac{w_j}{\theta})$, since $w_j \ge w_{j-1} \ge \lfloor\sqrt{n}\rfloor_2$ and $\varepsilon(i'+3) \ge \theta/8$. This is dominated by the summand for $h = j$ in (10), which is at least $\frac{p}{\theta}d_{j-1}w_j^{1+1/T'}$.

By induction hypothesis, the recursive call to EnumerateClose in line (14) takes expected time $\widetilde{O}(\frac{33p}{\theta}\sum_{1\le h\le j-1} d_{h-1}w_h^{1+1/T'})$ which is $\widetilde{O}(\frac{p}{\theta}\sum_{1\le h\le j-1} d_{h-1}w_h^{1+1/T'})$.

The final loop (22-26) on $J \in \mathcal{K}$ requires $O(|\mathcal{K}|w_j^{1+1/T'})$ time. So we need to bound the size of $\mathcal{K}$. $\mathcal{K}$ is created in the loop on i', I'. As noted there are $\widetilde{O}(1)$ iterations of these loops, so it suffices to bound the number of elements added to $\mathcal{K}$ for a single choice of I', i'. During lines (15-17), for each $J \in \mathcal{J}$, J is added to $\mathcal{K}$ if there is a $J' \in \mathcal{S}$ that is in ZoomIn$(i, I \times J, I', i')$. By Proposition 4.2, each $J' \in \mathcal{S}'$ is responsible for the addition of at most 33 intervals to $\mathcal{K}$, so $|\mathcal{K}| \le 33|\mathcal{S}'|$. Now, $\mathcal{S}'$ is the output of a call to EnumerateClose$(j-1, I' \times \mathcal{J}', i')$ where $I' \in$ SparseSample(j, I, i'). By the success condition for iteration $j-1$ of ProcessDense (Section 4.5) there are at most $2d_{j-1}$ intervals $J' \in$ Intervals$(w_{j-1}, \varepsilon(i'+3))$ classified as **CLOSE** for i'. As observed in the previous paragraph, each of these at most $2d_{j-1}$ intervals belongs to $\mathcal{J}'$ with probability at most $33p$. So the expected size of $|\mathcal{S}'| \le 66pd_{j-1}$. Thus the expected cost of the loop (22-26) is $\widetilde{O}(pd_{j-1}w_j^{1+1/T'})$. Combining with the other loop gives the claimed time bound for EnumerateClose. □

Now we analyze the running time of Preprocess(j). There are $\widetilde{O}(n/w_j)$ pairs (I, i) that are enumerated in the two outer loops. For each such pair, we construct Sparse(j, I, i) (which takes $\widetilde{O}(1)$ time), and $\mathcal{B}^{\text{below}}(j, I, i)$ whose running time is $\widetilde{O}(1)$ if $j = 1$ and is $\widetilde{O}(w_j + |\mathcal{R}(j-1, I)| + |\text{Intervals}(w_j, \varepsilon(i+3))|)$ for $j > 1$, which is the time to run APM. Summing over $O(\log(n))$ values of i and noting that $\varepsilon(i) \ge \theta$, we obtain the upper bound $\widetilde{O}(w_j + |\mathcal{R}(j-1, I)| + \frac{n}{\theta w_{j-1}})$. Summing over I gives $\widetilde{O}(n + |\mathcal{R}(j-1)| + \frac{n^2}{\theta w_{j-1}w_j})$. $|\mathcal{R}(j-1)|$ is at most the number of level $j-1$ candidates $\langle i; I' \times J'\rangle$ which is at most $\widetilde{O}(\frac{n^2}{\theta w_{j-1}^2})$. Since $w_h \ge \lfloor\sqrt{n}\rfloor_2$ for all h by assumption, the overall time for Preprocess(j) is $\widetilde{O}(n/\theta)$. We observe that this term is dominated by the $h = j$ term in the inner sum of (9) which is $\frac{n}{\theta^2}w_j^{1/T'}\frac{d_{j-1}}{d_j} \ge \frac{n}{\theta}$. The asymptotics of the running time does not depend on the choice of random bits B_{SS}^j.

We now analyze the time of ProcessDense(j). We will condition the analysis on fixing the random bits $(B^{\leq j-1}, B^j_{\text{SS}}) = \beta^{*j}$ so that $\mathbf{SG} \wedge \mathbf{SS}(\leq j) \wedge \mathbf{PD}(\leq j-1)$ holds. The multiplicative cost of the outer iteration on i is absorbed in the $\widetilde{O}$ term. The main part is the while loop (lines 8-22) on $I \in \mathcal{T}$. This cost is divided into two parts, the call to EnumerateClose within line (11), and the cost of (lines 14-20) which is only executed within the "else".

To bound the cost of the call to EnumerateClose in line (11), we want to apply Lemma 4.10. For the hypothesis of this lemma we need an upper bound p' on the probability of any particular w_j interval being selected for $\mathcal{S}$. According to the code of EnumerateClose, every interval is placed in $\mathcal{S}$ with probability at most $p = \min(1, c_0 \log n/d_j)$. However we need to consider the probability of a given interval being placed in $\mathcal{S}$ *conditioned on the event* $\mathbf{PD}(j)$, and this can be bounded above by $p/\Pr[\mathbf{PD}(j)]$. As noted in Section 4.5, $\mathbf{PD}(j)$ occurs with probability at least $1 - n^{-9} \geq 1/2$ so we can bound the conditional probability of any interval being placed in $\mathcal{S}$ by $2p$. Applying Lemma 4.10, the expected time for the call to EnumerateClose in line (11) is $\widetilde{O}(\frac{2p}{\theta}\sum_{1\leq h\leq j} d_{h-1} w_h^{1+1/T'})$ which is $\widetilde{O}(\frac{1}{\theta d_j}\sum_{1\leq h\leq j} d_{h-1} w_h^{1+1/T'})$. The number of times this is executed is the number of possible I, which is at most $|\text{Intervals}(w_j)| = n/w_j$, so the overall expected cost of calls to EnumerateClose in line (11) is $\widetilde{O}(\frac{n}{\theta w_j d_j}\sum_{h=1}^{j} d_{h-1} w_h^{1+1/T'})$, as claimed in the theorem.

The time for executing (14-20) is dominated by the time of the two calls to EnumerateClose, which are bounded to be at most $\widetilde{O}(\frac{1}{\theta}(\sum_{h=1}^{j} d_{h-1} w_h^{1+1/T'})$ using Lemma 4.10 with the trivial setting $p = 1$. The number of times this is executed is bounded by the number of times in the loop on I that I is declared dense and used as a pivot. We claim that if $\mathbf{PD}(j)$ holds then the number of pivots is upper bounded by $O(\frac{n}{\theta w_j d_j})$. To see this, first note that if I is chosen as a pivot then by Section 4.5, conditioning on $\mathbf{PD}(j)$ implies $|\text{EnumerateClose}(j, I \times \text{Intervals}(w_j, \varepsilon(i+3)), i)| \geq d_j/2$. Furthermore, we claim that if I and I' are both pivots then $\text{EnumerateClose}(j, I \times \text{Intervals}(w_j, \varepsilon(i+3)), i)$ is disjoint from $\text{EnumerateClose}(j, I' \times \text{Intervals}(w_j, \varepsilon(i+3)), i)$. Suppose for contradiction that both are pivots and there is a J in both sets, and that I is selected first as a pivot. Then by the Soundness of EnumerateClose, $\text{ncost}(I \times J) \leq \varepsilon(i - q_{j-1} - 6)$ and $\text{ncost}(I' \times J) \leq \varepsilon(i - q_{j-1} - 6)$ and so by the triangle inequality $\text{ncost}(I \times I') \leq \varepsilon(i - q_{j-1} - 7) = \varepsilon(h_1)$ (where h_1 is defined in the pseudocode of EnumerateClose.) But, in that case, the pseudocode of EnumerateClose ensures that I' is placed in $\mathcal{X}$ in line (16) and therefore removed from $\mathcal{T}$ in line (20), making it impossible for I' to be chosen as a pivot.

Since the sets $\text{EnumerateClose}(j, I \times \text{Intervals}(w_j, \varepsilon(i+3)), i)$ corresponding to pivots are pairwise disjoint subsets of $\text{Intervals}(w_j, \varepsilon(i+3))$ each have size at least $d_j/2$, and $|\text{Intervals}(w_j, \varepsilon(i+3))| = O(\frac{n}{\theta w_j})$, the number of pivots is at $O(\frac{n}{\theta w_j d_j})$. Multiplying this by the cost of a single loop as bounded above, the result is bounded above as claimed in the theorem. $\square$

4.10 Choosing the parameters

The time analysis is expressed in terms of the parameters $w_1, \ldots, w_k$ and $d_0, \ldots, d_k$. In this section we determine values of the parameters that achieve the claimed time bound. It is convenient to introduce parameters $\gamma_1, \ldots, \gamma_k$, $\delta_0, \ldots, \delta_k$ and τ, with $w_i = \lfloor n^{\gamma_i} \rfloor_2$ and $d_i = \lfloor n^{\delta_i} \rfloor_2$ and $\theta = \lfloor n^{-\tau} \rfloor_2$.

Recall that the parameters of gap-condition include $\zeta > 0$ and we only need our gap algorithm to work for $\tau \leq \zeta$. In the theorem we are allowed to choose ζ to be any positive constant. In the derivation below, we will see that we will need an upper bound on τ as a function of T' which will be used to determine ζ in the final proof of Theorem 4.1 in the next section.

We impose the following conditions.

- $d_0 = w_1 = \lfloor \sqrt{n} \rfloor_2$, so $\delta_0 = \gamma_1 = 1/2$
- $d_k = 1$, so $\delta_k = 0$.

The time for iteration j is:

$$\chi_j = \widetilde{O}\left(\frac{n}{\theta^2 w_j d_j}(\sum_{i=1}^{j} d_{i-1} w_i^{\frac{T'+1}{T'}})\right).$$

Define

- $\alpha_j = (1 - \gamma_j - \delta_j + 2\tau)$
- $\nu_i = \delta_{i-1} + (\frac{T'+1}{T'})\gamma_i$

Then the cost of processing level j can be rewritten as:

$$\chi_j = \widetilde{O}(\sum_{i=1}^{j} n^{\alpha_j + \nu_i}).$$

We now choose γ_i and δ_i subject to the following conditions:

- $\gamma_1 = \delta_0 = 1/2$
- α_j is the same for all j
- ν_i is the same for all i.
- $\delta_k = 0$.

It is easy to check that for any $B \geq 0$, the first three conditions are satisfied by:

$$\begin{aligned} \gamma_i &= 1/2 + B\frac{T'}{T'+1} - B\left(\frac{T'}{T'+1}\right)^i \\ \delta_i &= 1/2 - B + B\left(\frac{T'}{T'+1}\right)^i \end{aligned}$$

The condition $\delta_k = 0$ implies:

$$B = B_k(T') = \frac{(T'+1)^k}{2((T'+1)^k - T'^k)}$$

Then $\alpha_j = 1 - \gamma_j - \delta_j + 2\tau = \frac{B}{T'+1} + 2\tau$ and $\nu_i = 1 + \frac{1}{2T'}$. So the time for all iterations is:

$$\begin{aligned} \sum_{j=1}^{k} \chi_j &= \widetilde{O}\left(\sum_{j=1}^{k} j n^{1+\frac{1}{2T'}+\frac{B}{T'+1}+2\tau}\right) \\ &= \frac{k(k+1)}{2}\widetilde{O}\left(n^{1+\frac{1}{2T'}+\frac{B}{T'+1}+2\tau}\right). \end{aligned}$$

As indicated earlier, we will impose the condition $\tau \leq \frac{3T'-2}{6(6(T')^3+7(T')^2+T')}$

For fixed $T' \geq 1$, $B_k(T')$ is a decreasing function of k whose limiting value is $1/2$. So we choose $k = k(T')$ to be large enough so that $B \leq \frac{3T'+1}{6T'}$. While the value $k(T')$ is not important, it is straightforward to verify that we can choose $k(T') = \lceil (T'+1)(1+\ln(T'+1)) \rceil$.

Using the above choice for B, the exponent of n is at most $1 + \frac{1}{2T'} + \frac{3T'+1}{6T'(T'+1)} + 2\tau$ and a computation shows that setting $T = T' + 1/6$ and imposing $\tau \leq \frac{3T'-2}{6(6(T')^3+7(T')^2+T')}$ (which we can do since $T' \geq 1$) results in an upper bound on the exponent of $1 + 1/T$ as required.

Finally, we need to verify the assumptions (1) and (2) that $\frac{n}{w_j} \geq d_j$ and $\frac{w_j}{w_{j+1}} \leq \theta/2$. The former is immediate as $\gamma_j + \delta_j \leq 1$. For the latter, letting $M' = -\frac{1}{\log(n)} \log(\max_j 2w_j/w_{j+1})$, we require that $\theta \geq n^{-M'}$, which we can ensure for n large enough by choosing $\zeta < M$, where $M = \min_j \gamma_{j+1} - \gamma_j$.

4.11 Tying up the proof of Theorem 4.1

We have that SLOW-GED is a gap algorithm for edit distance satisfying gap-condition(T', ζ', Q') where $T' \geq 1$, $\zeta' > 0$ and $Q' \geq 1$. We have shown FAST-GED (using SLOW-GED as a subroutine) that satisfies gap-condition(T, ζ, Q) with $T = T' + 1/6$ and $\zeta > 0$ and $Q \geq 1$ are suitably chosen (depending only on T',ζ' and Q'. In Section 4.8 we proved that FAST-GED has quality $Q = 2^{q_k+6}$. In section 4.10 we adjusted the parameters so that the running time computed in Section 4.9 is $\widetilde{O}(n^{1+1/T})$ provided that $\theta \geq n^{-\frac{3T'-2}{6(6(T')^3+7(T')^2+T')}}$, $\theta \geq n^{-M/2}$ (where M is defined in Section 4.10) and also $\theta \geq \zeta'$. So we set $\zeta = \min(\zeta', M/2, \frac{3T'-2}{6(6(T')^3+7(T')^2+T')})$.

5 Proof of Theorem 1.1

Here we present the (routine) construction of the algorithm **FAST-ED-UB**^{T} promised by Theorem 1.1 Given T, let $\zeta(T)$ and $Q(T)$ be given by Theorem 1.2.

On input x, y, **FAST-ED-UB**^{T} defines $i_{\max} = \lfloor \zeta \log n \rfloor$ and for i from 1 to $i_{\max}$, runs FAST-GED on input $(x, y, \theta = 2^{-i}, \delta = 1/\zeta n \log(n))$. Define $i^* = 0$ if none of the runs accepts, and otherwise define i^* to be the largest index for which run i^* accepts. **FAST-ED-UB**^{T} outputs $Q2^{-i^*}n$. This is an upper bound on $d_{\text{edit}}(x, y)$ since if $i^* = 0$ then the output is $Qn \geq n$, and otherwise the first requirement of gap-condition ensures that $d_{\text{edit}}(x, y) \leq Q2^{-i^*}n$.

We claim that for $R = 2Q$, the probability that the output exceeds $R(d_{\text{edit}}(x, y) + n^{1-\zeta})$ is at most $1/n$. If $i^* = i_{\max}$ then the output is $2Qn^{1-\zeta} \leq R(d_{\text{edit}}(x, y) + n^{1-\zeta})$. So assume $i^* < i_{\max}$. Say that the ith run of **FAST-ED-UB**^{T} *fails* if $d_{\text{edit}}(x, y) \leq 2^{-i}n$ and the algorithm rejects. The probability that some iteration fails is at most $\delta\zeta \log n \leq 1/n$ so the probability that no iteration fails is at least $1 - 1/n$. If no iteration fails then in particular iteration $i^* + 1$ does not fail, and since it rejects (by the choice of i^*) we conclude that $d_{\text{edit}}(x, y) > 2^{-1-i^*}n$ and so $Q2^{-i^*}n \leq Rd_{\text{edit}}(x, y) \leq R(d_{\text{edit}}(x, y) + n^{1-\zeta})$, and so **FAST-ED-UB**^{T} has all of the required properties.

6 Approximate Pattern Matching

In this section we describe the implementation of the function APM$(I \times \mathcal{J}, \epsilon, \mathcal{R})$ from Section 4.3. This is a synthesis of algorithms from [16, 17].

We assume that $\mathcal{R}$ contains certified boxes and all $J \in \mathcal{J}$ are of the same width $\mu(I)$.

Let $\max(\mathcal{J}) = \{max(J) : J \in \mathcal{J}\}$ and $\min(\mathcal{J}) = \{min(J) : J \in \mathcal{J}\}$. Let $\mathcal{R}^+$ be $\mathcal{R}$ augmented by auxiliary shortcut edges of cost 0 from $(\min(I), 0)$ to $(\min(I), m)$ for all $m \in \min(\mathcal{J})$. Also for $J \in \mathcal{J}$ let J^0 denote the interval $\{0, \ldots, \max(J)\}$. The following was observed in [17]:

Proposition 6.1. *For all $J \in \mathcal{J}$, $cost_{\mathcal{R}}(I \times J)$ satisfies $cost_{\mathcal{R}^+}(I \times J^0) \leq cost_{\mathcal{R}}(I \times J)$ and $cost(I \times J) \leq 2cost_{\mathcal{R}^+}(I \times J^0)$.*

Proof. For the first inequality consider a min-cost traversal τ of $I \times J$ in the shortcut graph $\widetilde{G}(\mathcal{R})$. We construct a traversal τ' of $I \times J^0$ of cost at most $\text{cost}_{\mathcal{R}}(\tau)$. Consider the first shortcut edge $e = (i, j) \to (i', j')$ of τ. We may assume that prior to e, the path consists of a (possibly empty) sequence of horizontal edges followed by a (possibly empty) sequence of vertical edges. The final such horizontal edge ends at $(\min(I), j)$ and $j \in \min(\mathcal{J})$ so in $\widetilde{G}(\mathcal{R}^+)$ we can replace the horizontal path by the shortcut edge $(\min(I), 0) \to \min(I), j)$ of cost 0 to get a path that is no more costly.

For the second inequality, consider a min-cost traversal ρ of $I \times J^0$ in $\widetilde{G}(\mathcal{R}^+)$. Let $j = 0$ if the path does not use one of the auxiliary shortcut edges, and otherwise let j be such that the path starts with auxiliary shortcut edge $(\min(I), 0) \to (\min(I), j)$. Let $\hat{J} = \{j, \ldots, \max J\}$. So the remaining portion of ρ is a min-cost traversal $\hat{\rho}$ of $I \times \hat{J}$. Since $\widetilde{G}(\mathcal{R})$ is certified, $|\mu(\hat{J}) - \mu(I)| \leq \text{cost}(I \times \hat{J}) \leq \text{cost}_{\mathcal{R}}(\hat{\rho}) = \text{cost}_{\mathcal{R}^+}(\hat{\rho}) = \text{cost}_{\mathcal{R}^+}(I \times J^0)$. Also $|J \Delta \hat{J}| = |\mu(\hat{J}) - \mu(I)| = |\min(J) - j| \leq \text{cost}_{\mathcal{R}^+}(I \times J^0)$. So $\text{cost}(I \times J) \leq \text{cost}(I \times \hat{J}) + |J \Delta \hat{J}| \leq 2\text{cost}_{\mathcal{R}^+}(I \times J^0)$. $\square$

So if we compute $\text{cost}_{\mathcal{R}^+}(I \times J^0)$ for every $J \in \mathcal{J}$, and output the set of all J for which this cost is less than $\kappa\mu(I)$, we will satisfy the requirements of APM. We now describe a slightly modified version of an algorithm from [16] that accomplishes this in time $\widetilde{O}(|\mathcal{R}^+|)$.

Let $\widetilde{H}$ be the graph $\widetilde{G}(\mathcal{R}^+)$ with each cost c_e of $e = (i, j) \to (i', j')$ replaced by *benefit* $b_e = (i' - i) + (j' - j) - c_e$, (so H and V edges have benefit 0). For any interval B, the min-cost traversal of $I \times B$ in $\widetilde{G}(\mathcal{R}^+)$ is $\mu(I) + \mu(B)$ minus the max-benefit traversal of $I \times B$ in $\widetilde{H}$. So it suffices to compute the max-benefit traversal of $I \times J^0$ in $\widetilde{H}$ for all $J \in \mathcal{J}$.

To do this, let $j_1 < \cdots < j_r$ be the distinct second coordinates of the heads and tails of shortcut edges in $\widetilde{G}(\mathcal{R}^+)$. We use a binary tree data structure with leaves corresponding to the indices of I, where each tree node v stores a number a_v, and a collection of lists $L_1, \ldots, L_r$, where L_h stores pairs $(e, q(e))$ where the head of e has y-coordinate j_h and $q(e)$ is the max benefit of a path from $(\min(I), 0)$ that ends with e.

We proceed in rounds $h = 1, \ldots, r$. In round h, let A_h consist of all the shortcuts whose tail has vertical coordinate j_h. The preconditions for round h are: (1) for each leaf i, the stored value a_i is the max benefit path to (i, j_h) that includes a shortcut whose head has horizontal coordinate i (or 0 if there is no such path), (2) for each internal node v, $a_v = \max\{a_i : i \text{ is a leaf in the subtree of } v\}$, and (3) for every shortcut edge $e = (i', j_{h'}) \to (i'', j_{h''})$ with $h' < h$, the value $q(e)$ has been computed and $(e, q(e))$ is in list $L_{h''}$.

During round h, for each shortcut $e = (i, j_h) \to (i', j_{h'})$ in A_h, $q(e)$ equals the max of $a_\ell + b_e$ over tree leaves ℓ with $\ell \leq i$. This can be computed in $O(\log n)$ time as max $a_v + b_e$, where v ranges over the union of $\{i\}$ with the set of left children of vertices on the root-to-i path that are not themselves on the path. Add $(e, q(e))$ to list $L_{h'}$. After processing A_h, update the binary tree: for each $(e, q(e)) \in L_{h+1}$, let i be the horizontal coordinate of the head of e and for all vertices v on the root-to-i path, replace a_v by $\max(a_v, q(e))$. The tree then satisfies the precondition for round $h + 1$.

To obtain the output to APM, for each $J \in \mathcal{J}$, let $h(J)$ be the index of the last iteration for which $j_{h(J)} \leq \max(J)$. The benefit of $I \times J^0$ is the value, at the end of iteration of $h(J)$ of a_{v_0}

where v_0 is the root.

For the runtime analysis: It would take $\widetilde{O}(\mu(I))$ time to set up the full tree data structure so we will build it incrementally by expanding only the parts of the data structure that contain non-zero values. Hence, the set up cost of the data structure is $O(1)$. It takes $O(|\mathcal{R}^+| \log |\mathcal{R}^+|)$ time to sort the shortcuts, and $O(\log \mu(I))$ processing time per shortcut (computing $q(e)$ and later updating the data structure), overall giving runtime $\widetilde{O}(|\mathcal{R}^+| + |\mathcal{J}|)$.

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.

[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.

[4] Alex Andoni. Simpler constant-factor approximation to edit distance problems. *Manuscript*, 2018.

[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] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce (extended version of [11]). 2018.

[13] Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. *CoRR*, abs/1904.05390, 2019.

[14] 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.

[15] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 979–990, 2018. URL: `https://doi.org/10.1109/FOCS.2018.00096`, `doi:10.1109/FOCS.2018.00096`.

[16] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *CoRR*, abs/1810.03664, 2018. URL: `http://arxiv.org/abs/1810.03664`, `arXiv:1810.03664`.

[17] Diptarka Chakraborty, Debarati Das, and Michal Koucký. Approximate online pattern matching in sub-linear time. *CoRR*, abs/1810.03551, 2018. URL: `http://arxiv.org/abs/1810.03551`, `arXiv:1810.03551`.

[18] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. *FOCS*, 2019.

[19] Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016.

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

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

[22] 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.

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

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