Almost Linear Size Edit Distance Sketch

Michal Koucký[*1,2] and Michael Saks[†3]

[1]Computer Science Institute of Charles University, Prague, Czech Republic
[2]DIMACS, Rutgers University, Piscataway, NJ, USA
[3]Department of Mathematics, Rutgers University, Piscataway, NJ, USA

June 15, 2024

Abstract

Edit distance is an important measure of string similarity. It counts the number of insertions, deletions and substitutions one has to make to a string x to get a string y. In this paper we design an almost linear-size sketching scheme for computing edit distance up to a given threshold k. The scheme consists of two algorithms, a sketching algorithm and a recovery algorithm. The sketching algorithm depends on the parameter k and takes as input a string x and a public random string ρ and computes a sketch $sk_\rho(x; k)$, which is a digested version of x. The recovery algorithm is given two sketches $sk_\rho(x; k)$ and $sk_\rho(y; k)$ as well as the public random string ρ used to create the two sketches, and (with high probability) if the edit distance $\mathbf{ED}(x, y)$ between x and y is at most k, will output $\mathbf{ED}(x, y)$ together with an optimal sequence of edit operations that transforms x to y, and if $\mathbf{ED}(x, y) > k$ will output LARGE. The size of the sketch output by the sketching algorithm on input x is $k2^{O(\sqrt{\log(n)\log\log(n)})}$ (where n is an upper bound on length of x). The sketching and recovery algorithms both run in time polynomial in n. The dependence of sketch size on k is information theoretically optimal and improves over the quadratic dependence on k in schemes of Kociumaka, Porat and Starikovskaya (FOCS'2021), and Bhattacharya and Koucký (STOC'2023).

1 Introduction

The *edit distance* of two strings x and y measures how many edit operations (removing a symbol, inserting a symbol or substituting a symbol by another) are needed to transform x to y. Computing edit distance is a classical algorithmic problem. For input strings of length at most n, edit distance can be computed in time $O(n^2)$ using dynamic programming [WF74, MP80, Gra16]. Assuming the Strong Exponential Time Hypothesis (SETH), this cannot be improved to truly sub-quadratic time

$O(n^{2-\epsilon})$ [BI18]. When parameterized by the edit distance $k = \mathbf{ED}(x, y)$, the running time has been improved to $O(n + k^2)$ [LMS98]. The edit distance of two strings can be approximated within a constant factor in time $O(n^{1+\epsilon})$ [AN20, KS20, BR20, CDG$^+$20].

This paper concerns *sketching schemes* for edit distance, which consist of a *sketching algorithm*, parameterized by an integer k, that takes a string x and (using a public random string ρ) maps it to a short *sketch* $\mathrm{SK}_\rho(x; k)$, and a *recovery algorithm* that takes as input two sketches $\mathrm{SK}_\rho(x; k)$ and $\mathrm{SK}_\rho(y; k)$ and the public random string ρ and, with high probability (with respect to ρ), outputs $\mathbf{ED}(x, y)$ when $\mathbf{ED}(x, y) \le k$ and outputs LARGE otherwise.

The goal is to get polynomial time sketch and recovery algorithms that achieve the smallest possible sketch length. Jin, Nelson and Wu [JNW21] proved that sketches must have length $\Omega(k)$. For edit distance it is not apriori clear whether sketches of size $k^{O(1)} n^{o(1)}$ exist, even non-uniformly. The first sketching scheme with $poly(k)$ sketch size was found by Belazzougui and Zhang [BZ16] who attained sketch size $\widetilde{O}(k^8)$. (Here, and throughout the paper, $\widetilde{O}(t)$ means $t \log^{O(1)} n$, where n is an upper bound on the length of the strings.) This was improved to $\widetilde{O}(k^3)$ by Jin, Nelson and Wu [JNW21], and then to $\widetilde{O}(k^2)$ by Kociumaka, Porat and Starikovskaya [KPS21]. The above sketches used *CGK random walks* on strings [CGK16] to embed the edit distance metric into the Hamming distance metric with distortion $O(k)$, plus additional techniques. A different approach, based on a string decomposition technique, was used by Bhattacharya and Koucký [BK23]. Here we will also use this technique.

The quadratic dependence on k in the (very different) sketches of [KPS21] and [BK23] and also in the exact computation algorithm of [LMS98] is suggestive that there may be something intrinsic to the problem that requires quadratic dependence on k for both sketching and evaluation. In this paper, we show that this is not the case, by presenting an efficiently computable sketch of size $O(k 2^{O(\sqrt{\log(n) \log \log(n)})})$ which is k times a "slowly" growing function of n, that is intermediate between $\log^{\omega(1)} n$ and $n^{o(1)}$. Our main result is:

Theorem 1.1 (Sketch for edit distance)**.** *There is a randomized sketching algorithm* ED-SKETCH *that on an input string x of length at most n with parameter $k < n$ and using a public random string ρ produces a sketch $\mathrm{SK}_\rho(x)$ of size $O(k 2^{O(\sqrt{\log(n) \log \log(n)})})$, and recovery algorithm* ED-RECOVER *such that given two sketches $\mathrm{SK}_\rho(x)$ and $\mathrm{SK}_\rho(y)$ for two strings x and y and ρ, with probability at least $1 - 1/n$ (with respect to ρ), outputs $\mathbf{ED}(x, y)$ if $\mathbf{ED}(x, y) \le k$ and* LARGE *otherwise. The running time of the* ED-SKETCH *is $n^{O(1)}$ and the running time of* ED-RECOVER *is $\widetilde{O}(\min(n^2, k^3 2^{O(\sqrt{\log(n) \log \log(n)})}))$.*

We remark that we did not attempt to optimize the running time, or poly-log factors in the sketch sizes. The running time of ED-SKETCH is largely determined by the running time of the Ostrovsky-Rabani embedding [OR07] (see below) which runs in polynomial time, but we don't know the exponent. The amount of randomness the algorithm uses can be reduced to poly-logarithmic in n.

Our sketch has the additional property that the recovery algorithm also determines an optimal sequence of edit operations that transforms x to y.

Sketching for Hamming distance has been studied extensively and is well understood. Several approaches yield sketches of size $\widetilde{O}(k)$ that can recover the Hamming distance and can solve the harder problem of *mismatch recovery*, i.e., reconstructing the set of positions where the two strings differ, see e.g. [CKP19, PL07]; this sketch size is information theoretically optimal. Our construction for edit distance uses sketches for Hamming distance, but we need a more refined version of Hamming

distance sketches that allow for recovery of all differences in regions of the strings where the density of differences is low, even if the overall Hamming distance is large. We call this new sketch a *hierarchical mismatch recovery scheme*. This is the main new technical tool.

Not much is known about sketching edit distance when we only want to approximate edit distance from the sketches. For Hamming distance, there are known sketches of poly-logarithmic size in n that allow recovery of Hamming distance within a $(1 + \epsilon)$-factor [FKSV02]. For edit distance nothing like that is known. A more stringent notion of sketching is that of embedding edit distance metrics into ℓ_1 metrics. The best known result in this direction, by Ostrovsky and Rabani [OR07], gives an embedding of edit distance into ℓ_1 with *distortion* (approximation factor) $s_{\mathrm{OR}}(n) = 2^{O(\sqrt{\log(n) \log \log(n)})}$. Interestingly, our sketch relies on this embedding to *fingerprint* strings by their approximate edit distance. The dependence of our sketch size on n in Theorem 1.1 can be stated more precisely as $\widetilde{O}(s_{\mathrm{OR}}(n)^2)$. Since our use of their embedding is "black box", any improvement on the distortion factor for embedding edit distance into ℓ_1 would give a corresponding improvement in our sketch size.

2 Our technique

The starting point of our sketch is the string decomposition algorithm of Bhattacharya and Koucký [BK23], The algorithm BASIC-DECOMP (see Section 4.3) takes a string x and partitions it into fragments so that each fragment can be described concisely by a small context-free grammar. The size of the grammar is at most k' for some chosen parameter k'. (In [BK23] this k' is chosen to be $\widetilde{O}(k)$.) The partitioning uses randomness to select the starting point for each fragment and has the property that for any given position in the string, the probability that it will start a fragment is at most $p \approx 1/k'$.

The partitioning process is *locally consistent*, i.e., when applied to two strings x and y with probability at least $1 - \widetilde{O}(\mathbf{ED}(x, y)/k')$ the decompositions of x and y are *compatible*. Being compatible means that they have the same number of fragments and $\mathbf{ED}(x, y)$ is equal to the sum of edit distance of corresponding fragments. In particular, if x and y are compatibly split then to recover $\mathbf{ED}(x, y)$ it suffices to reconstruct each corresponding pair of fragments of x and y that are different and sum up their edit distances.

In [BK23] this decomposition procedure was used to obtain edit distance sketches of size $\widetilde{O}(k^2)$ by reducing the problem of sketching for edit distance to the easier problem of sketching for (Hamming) mismatch recovery mentioned in the introduction. This is the problem of sketching two equal length sequences so that from their sketches one can recover all the locations where the two sequences differ.

The reduction to mismatch recovery sketches proceeds as follows. The string x is decomposed into fragments. The grammar of each fragment is encoded using some fixed-size encoding of length $\widetilde{O}(k')$, the encodings are concatenated and the resulting sequence is sketched using a mismatch recovery scheme. Given the Hamming distance sketch for the sequence of fragments of x, and the corresponding sketch for y, we can (with an additional technical trick) recover all corresponding pairs of grammars that are different. Each pair of corresponding grammars represents a pair of corresponding fragments of x, y, and $\mathbf{ED}(x, y)$ is equal to the sum of the edit distances of all recovered pairs of fragments. There are at most k corresponding pairs of grammars that differ, and each of them has $k' = \widetilde{O}(k)$ bits. The size of the Hamming sketch needed to perform this recovery is $\widetilde{O}(k^2)$. This is the sketch from [BK23].

The sketch length is $\widetilde{O}(k^2)$ because the sketch must handle two different extremes. If all edit

3

operations appear in large clusters of size k/C, for $C = \widetilde{O}(1)$, and each cluster is contained in a single fragment pair, then there are at most C fragment pairs that are unequal and these could be handled by sketches of size $\widetilde{O}(k)$. On the other hand, if the edit operations are well separated, then we could choose a value of k' that is $\widetilde{O}(1)$ resulting in a partition into much smaller fragments each of which has grammar size $\widetilde{O}(1)$. In this case the edit operations may appear in $\Omega(k)$ different fragment pairs, but because the grammar size of the fragments is $\widetilde{O}(1)$ we can again manage with sketch size $\widetilde{O}(k)$. Of course, the distribution of edit operations will rarely match either of these extremes, there may be clusters of edit operations of varying size and density.

Decomposition tree. A natural approach to handling this is to build decompositions for many different values of k'. We start with a decomposition obtained from parameter $k' = k_0$ which is larger, but not much larger than k. We then apply the decomposition again to each fragment of the first decomposition, using the smaller parameter $k_1 = k_0/2$. We iterate this recursively where the value k_i of k' at recursion level i is $k_0/2^i$ stopping when k_i is $\widetilde{O}(1)$. This would decompose the string into smaller and smaller fragments ending with fragments described by constant size grammars. We call this a *decomposition tree* of the string. We can then do separate sketches for each of the levels of the tree. The sketches at the top of the tree (small i) are used to find edit operations that occur together in large clusters, and there can't be too many of these clusters. Sketches at the bottom of the tree (large i) are used to find edit operations in regions where the edit operations are well spread. These edit operations may be spread over $\Omega(k)$ pairs of fragments, but each such pair has edit distance $\widetilde{O}(1)$. The intermediate levels would handle cases where the density of edit operations is intermediate between these extremes.

To be more explicit, the sketch associated to decomposition level i is responsible for recovering pairs of compatible fragments at level i whose edit distance is (roughly) comparable to or larger than k_i, that could not be recovered from the sketches for previous levels of the decomposition. There are at most $\widetilde{O}(k/k_i)$ such pairs of fragments (counting only compatibly split pairs). Since the fragments are represented by grammars of size at most k_i, the level i scheme will need to find at most $\widetilde{O}(k)$ mismatches for compatibly split fragments.

For compatible pairs of fragments at level i whose edit distance is small compared to k_i, the decomposition procedure will split them compatibly (with *fairly high probability*[1]) and the edit distance for these will be recovered from the sketches corresponding to deeper levels of the decomposition. In this way, all of the edit operations will be found by some level of the decomposition.

While the sketch for level i only needs to identify at most $\widetilde{O}(k)$ mismatches, it can not use an ordinary mismatch recovery sketch for $\widetilde{O}(k)$ mismatches, because the sequences of the grammar encodings for the level i fragments may differ in many more than $\widetilde{O}(k)$ positions, since the $\widetilde{O}(k)$ positions account only for the differences due to pairs of fragments that must be recovered at level i but not those due to differences coming from other fragment pairs as explained further. To deal with this, we will need the extension of mismatch recovery mentioned earlier that handles hierarchical mismatch recovery (which will be discussed in more detail later in this section.)

Grammar representation. From the sketch at a level i which corresponds to grammars of size at most k_i we hope to recover those fragments that have roughly k_i edit operations that were not recovered from the sketches at earlier levels. Here it becomes important how we represent the grammars to the Hamming sketch. Each grammar is a subset of *rules* from a domain of polynomial size. The grammars that we produce have the following useful property: If we represent two

---

[1] Here *fairly high probability* means probability at least $1 - 1/\text{poly}\log n$ for a sufficiently large poly-logarithmic function.

fragments of x and y by grammars and the fragments have edit distance at most h then the grammars will differ in at most $\widetilde{O}(h)$ rules. So if we represent the grammars by their characteristic vectors the vectors will have Hamming distance at most $\widetilde{O}(h)$. This is useful since the fragments that contain much less than k_i edit operations and that should be recovered from a Hamming sketch for some level $> i$ will contribute to the Hamming sketch of level i by at most $\widetilde{O}(k)$ differences in their grammars in total. For fragments that have edit distance $\geq k_i$ (that were not recovered from earlier levels) we want to recover their complete grammars.

Watermarking using edit distance fingerprints. For the level i encoding, we want that if two corresponding fragments are at edit distance at least $t_i \approx k_i$, then recovering the mismatches in their grammar encodings is enough to fully recover both grammars and therefore both fragments. To ensure this, we will *watermark* the grammar encodings by a special fingerprint (computed from the entire fragment), and replace each 1 in the grammar encoding by the fingerprint. Assuming that the watermark of the two fragments are different, then this will allow one to recover both grammars from the set of mismatches. The watermark we use is a *threshold edit distance fingerprint.* This is a randomized fingerprinting scheme depending on parameter t, that maps each string to an integer so that if two strings have edit distance more than t, their fingerprints differ with high probability, and if two strings have edit distance less than t/P, for some parameter $P > 1$ then their fingerprints will be the same with fairly high probability (with no promise if the edit distance lies in the interval $[t/P, t]$.) The parameter P is a measure of the quality of the fingerprinting algorithm; with smaller P being higher quality. Such a fingerprinting scheme can be obtained from an embedding of the edit distance metric into the ℓ_1 metric. We use the embedding of Ostrovsky and Rabani [OR07] which has distortion $s_{\mathrm{OR}} = 2^{\sqrt{\log n \log \log n}}$ and this distortion translates into the parameter P of the fingerprinting scheme.

Canonical edit operations. For the sketching procedure as outlined, the recovery algorithm will recover corresponding pairs of unequal but compatible fragments and compute their edit distance. While the recovered fragment pairs are likely to encompass most edit differences between the two strings, it will typically miss some pairs, and may also incorrectly recover a small fraction of the fragment pairs. This means that the set of edit operations recovered will only be approximate. To address this we will need to do multiple independent sketches and when doing recovery, we combine the outcomes recovered from each independent sketch (as described below). As mentioned earlier if $\mathbf{ED}(x, y) = k$, there could be many different sets of k edit operations that transform x to y. To output edit operations consistently among independent runs of the sketching algorithm we will always opt for a *canonical* choice of the edit operations. The canonical choice prefers insertions into x over substitutions which in turn are preferred over deletions from x. This preference is applied on edit operations from left to right in x so the choice of edit operations corresponds to the leftmost shortest path in the usual edit distance graph of x and y. Importantly, the canonical path is consistent under taking substrings of x and y. See Section 3.2 for details on the choice of the path.

Bad splits. For two strings x and y, as we split the fragments of each through successive levels we will inevitably split some fragments near an edit operation; this is called a *bad split.* A bad split might cause x to be divided at a particular location but not y. This causes two problems: (1) sub-fragments of the badly split fragment may not align, and (2) the bad split may cause fragments to the right of the badly split fragment to be misaligned. Both problems need to be dealt with.

Regularization of trees. To handle the second issue we regularize the decomposition tree. The decomposition tree is of depth at most $O(\log n)$ and it might have degree up to n. We make it regular as follows: for any node with fewer than n children we append dummy children to the

right of the real children; these are thought of as representing empty strings. We do this at all levels so that the tree has degree n in all the internal nodes and all the leaves are at depth exactly $d = O(\log k + \sqrt{\log n \log \log n})$. Each node of the tree is labeled by a fragment of x and its corresponding grammar. This ensures that the underlying tree for the decomposition is the same for every input string. Now, in constructing the level $i + 1$ decomposition from the level i, the fragments that correspond to the children of node v in the two trees are paired with each other. A bad split arising from the decomposition of the two fragments corresponding to tree node v may cause misalignment among the descendants of v but will not affect the alignment of nodes to v's right.

Sparse representation. The regularized tree is of super-polynomial size $\leq n^{\log n}$, but only has $\widetilde{O}(n)$ nontrivial nodes that represent nonempty strings. As a result, the tree can be constructed in polynomial time, and represented concisely by the set of ordered pairs of (node, fragment) for nonempty fragments. Furthermore the sketching algorithm works well with the sparse representation and its running time is at most polynomial in the number of nontrivial nodes.

Mismatch floods. The more significant issue caused by bad splits is the first one, that a bad split at a node may result in bad splits at many of its descendants. Indeed, if the partition of the fragments x_v and y_v of x and y at node v are not compatibly split, then the fragment pairs of its children will not be aligned and the edit distances between fragments corresponding to children of v may be arbitrarily large. This misalignment will propagate down the tree possibly resulting in huge total edit distance between fragments for many nodes in the subtree rooted at v, and the grammar encodings of these fragments may have a very large Hamming distance.

A node in the tree that is in the subtree of a badly split node is referred to as *flooded* if its corresponding fragments need many edit operations. At level i of the tree, we only need to recover the grammars corresponding to the unflooded nodes of the level (because we expect that the edit operations for each flooded node will be recovered by the sketches corresponding to an unflooded ancestor in the tree.) The usual mismatch recovery sketch does not allow for such selective recovery of unflooded portions, because the flooded portion may cause the total number of mismatches to far exceed the capacity of the mismatch recovery scheme. Thus we design a new variant of Hamming schemes, which we call *hierarchical mismatch recovery scheme*, that recovers differences in the unflooded parts.

Hierarchical mismatch recovery. The *hierarchical mismatch recovery scheme* is applied to a vector that is indexed by the leaves of a tree. The specification of the problem includes an assignment of a positive capacity κ_v to each node v in the tree, where all nodes at level i have the same capacity κ_i. For any two strings x and y, the capacity function induces a *load function* $\widehat{\kappa}$, where the value $\widehat{\kappa}_v$ is 1 on leaves where x and y differ and 0 on other leaves, and the load on a node v is the minimum of the sum of the loads of its children and its own capacity. A node v is *underloaded* if its load is less than κ_v/R for some parameter R, and a leaf is *accessible* if every node on its path to the root is underloaded. The scheme is required to recover the mismatch information for all accessible leaves.

The sketch is implemented as follows: Let d be the depth of the tree. For each node at level $d-1$ we apply a (standard) Hamming mismatch recovery scheme to the vector of its children that handles (slightly more than) κ_{d-1} mismatches. Then working the way up the tree, for each node we apply a mismatch recovery to the vector consisting of the sketches computed at each of its children. This is passed upwards and the final sketch at the root is the output. The specific mismatch recovery scheme used is a *superposition scheme* which is described in Section 5.2. The hierarchical sketches for two strings x and y will allow for recovery of mismatches occurring at all accessible leaves.

6

The intuition behind this is as follows. For two strings x and y, label each node by the pair of intermediate sketches for x and y at that node. This pair of sketches encodes information about the mismatches of x and y at the leaves of its subtree. This information at node v is a *compression* to $\widetilde{O}(\kappa_i)$ bits of the information from its children. Inductively one can show that for an underloaded node the compressed string it computes has enough bits to preserve the information about accessible mismatches that is encoded in the sketches of its children. For overloaded nodes, the compression will destroy the information about its children, but this is not a problem because the scheme is not required to recover mismatch information for leaves below an overloaded node. An important thing to note is that the impact of overloaded nodes among v's children (which may have a large number of mismatches among its leaves) is controlled by the fact that the sketch size at those nodes is restricted by its capacity.

In our application to sketching the grammars we assign the capacity to each level of the tree so that the capacity is proportional to the grammar size k_i we expect at that level in the decomposition tree. Nodes that correspond to misaligned fragments (due to a bad split at an ancestor) and have a huge edit distance will correspond to overloaded nodes, and as discussed above, the hierarchical scheme contains the damage caused by the error flood to the subtree below the occurrence of the bad split. The idea behind the choice of the parameters is that the probability of a bad split is proportional to the number of edit operations in the fragment; it is roughly $\widetilde{O}(\# \text{ of edit op's}/k_i)$. Hence, in expectation each edit operation is responsible for $\widetilde{O}(1)$ mismatches resulting from bad splits that contribute to possible flooding of a node. We can adjust the parameters so that the flooding of a node is in expectation only a tiny fraction of its capacity. We can then apply Markov's inequality to argue that with a good probability nodes along a chosen path are not overloaded, i.e., flooded. Details for our hierarchical mismatch recovery scheme are given in Section 5.1.

Parameters. For our edit distance sketch we will set the parameters as follows: $\kappa_0 = \widetilde{O}(s_{\text{OR}}^2 k)$, $\kappa_i = \kappa_0/2^i$, $k_0 = \widetilde{O}(s_{\text{OR}} k)$, $k_i = k_0/2^i$ and $t_0 = \widetilde{O}(s_{\text{OR}} k)$, $t_i = t_0/2^i$. Our sketch will be obtained by applying the hierarchical mismatch recovery scheme to each level of the decomposition tree with those parameters.

Infrequent bad splits. The decomposition procedure is designed so that for a node v at level i with fragments x_v and y_v, if $\mathbf{ED}(x_v, y_v) \leq k_i/C$ where $C = \text{polylog}(n)$ the decompositions of x_v and y_v will be compatible with probability $1 - 1/\text{polylog}(n)$. While this is near 1, it is likely a non-trivial fraction of nodes will fail to be compatibly split even though the edit distance between the strings is small compared to k_i. As a result, for any given edit operation, there may be a small but non-trivial chance that the recovery algorithm fails to recover it. To ensure that we get all of the edit operations, we will need to run the scheme $O(\log n)$ times and include those edit operations produced by more than half the runs to guarantee that every edit operation gets recovered with good probability.

Location, location, location! The recovery procedure identifies pairs of fragments that differ, and can therefore reconstruct the canonical edits between those fragments. But as described so far, there is nothing that allows the reconstruction to pinpoint where these fragments appear in the full string. Without this information, we can not combine information obtained from the independent sketches as just described. Our sketch will need an additional component to properly position each recovered fragment.

Location tree. We will use technique of Belazzougui and Zhang [BZ16] (suggested to us by Tomasz Kociumaka). We turn the decomposition tree into a binary tree by expanding each node with n children into a binary tree of depth $\log(n)$ with n leaves. For each node in the binary tree,

we record the length of the substring represented by its left child. We watermark this size by the usual Karp-Rabin fingerprint of the node substring, and we sketch the sizes using the hierarchical mismatch sketch as in the grammar tree, i.e., level by level. Fragments that contain edit operations will differ in the Karp-Rabin fingerprint so they will reveal the size of their left child. For a given node that contains an edit operation all of its ancestors on the path to the root will also be watermarked. Hence we will be able to recover the information about the size of all the left children along the path. That suffices to calculate the position of each differing fragment. We will use the same setting of capacities for the hierarchical mismatch scheme that we use for grammars. That is clearly sufficient as grammars are larger objects than a single integer.

Putting things together. The actual sketch consists of multiple independent sketches. Each of these sketches is the output of the hierarchical mismatch recovery scheme applied to each of the levels of the grammar decomposition tree, and applied to each of the levels of the binary location tree. Its full details are provided in Section 6.

Recovery. We briefly explain the recovery of edit operations from sketches for two strings. The reconstruction starts by running the recovery procedure for all the hierarchical mismatch recovery sketches. This recovers various pairs of grammars with information about their location within the original input strings x and y. From those grammars we pick only those which do not have any ancestor grammar node recovered as well. (Descendant grammars are superseded by the ancestor grammars.) For every pair of grammars we reconstruct the fragments they represent and compute the associated edit operations. (The edit operations could be actually computed without decompressing the grammars [GKLS22].) For each pair of recovered fragments, we use the location tree sketches to recover the exact location of that fragment.

So for each pair of recovered fragments we calculate the canonical sequence of edit operations and given the exact location of the fragment, we can determine the exact location within x and y where each edit occurred. We repeat this for each independent copy of the sketch. Our final output is the set of edit operations that appear in the majority of the copies. These are all the operations we were supposed to find. We provide details of the analysis in Section 7.1. The analysis of the process is somewhat delicate as one needs to deal with various dependencies among the probabilistic events. Overall it reflects the intuition provided above.

3 Preliminaries

3.1 Strings, sequences, trees, and string decomposition

For an alphabet Γ, Γ^* denotes the set of (finite) strings of symbols from Γ, Γ^n is the strings of length exactly n and $\Gamma^{\leq n}$ is the set of strings of length at most n. We write ε for the empty string of length 0. The length of string x is denoted $|x|$. For an index $i \in \{1, \ldots, |x|\}$, x_i is the i-th symbol of x.

We will consider the input alphabet of our strings to be $\Sigma = \{0, 1, \ldots\}$ which is the set of natural numbers. For strings of length up-to n we will assume they are over sub-alphabet $\Sigma_n = \{0, 1, \ldots, n^3 - 1\}$ so typically we will assume our input comes from the set $\Sigma_n^{\leq n}$. This assumption is justified as for computing edit or Hamming distance of two strings of length at most n we can hash any larger alphabet randomly to the range Σ_n without affecting the distance of the two strings with high probability.

If Γ is linearly ordered then Γ^* is also linearly ordered under lexicographic order, denoted by

$<_{\mathrm{lex}}$, given by $x <_{\mathrm{lex}} y$ if x is prefix of y or if $x_j < y_j$ where j is the least index i for which $x_i \neq y_i$. We usually write $x < y$ instead of $x <_{\mathrm{lex}} y$.

We write $x \circ y$ for the concatenated string x followed by y and for a list $z_1, \ldots, z_k$ of strings we write $\odot_{i=1}^{k} z_i$ for $z_1 \circ \cdots \circ z_k$.

Substrings and fragments For a string x and an interval $I \subseteq \{1, \ldots, |x|\}$, a string z is a *substring of x located at I* if $|z| = |I|$ and for all $i \in I$, $z_{i-\min(I)+1} = x_i$. We denote this substring by x_I. When using intervals to index substrings, it is convenient to represent intervals in the form $(i, j] = \{i+1, \ldots, j\}$ and $(i, i]$ denotes the empty set for any i. (So a substring is always a consecutive sub-sequence of a string.) We can also say that z is the *substring of x starting at position* $\min(I)$. Furthermore, z is a *substring of x* if z is a substring of x starting at some position. However, the statement that z is a substring of x says nothing about where z appears in x, and there may be multiple (possibly overlapping) occurrences of z in x. For us it will be important where a substring appears. For a string x and an interval $I \subseteq \{1, \ldots, |x|\}$, the *fragment located at I* is the pair x_I together with I.

Sequences and Hamming Distance We will consider finite sequences of elements from some domain. It will be convenient to allow sequences to have index sets other than the usual integers $\{1, \ldots, n\}$. If D is any set, a *D-sequence* is an indexed collection $a = (a_i : i \in D)$. A *D-sequence over the set A* is a D-sequence with entries in A. A^D denotes the set of D-sequences over A. The *Hamming Distance* $\mathbf{Ham}(x, y)$ between two D-sequences x and y is the number of indices $i \in D$ for which $x_i \neq y_i$. We let $I_{\neq}(x, y) = \{i \in D; x_i \neq y_i\}$.

Trees Our algorithm will organize the processed data in a tree structure. To simplify our presentation we will give the tree very regular structure. For finite sets $L_1, \ldots, L_d$, $T(L_1 \times \cdots \times L_d)$ denotes the rooted tree of depth d where for each $j \in \{0, \ldots, d-1\}$ every internal node v at depth j has $|L_{j+1}|$ children, and the edges from v to its children are labeled by distinct elements of L_{j+1}. Each node v at depth j is identified with the length j sequence of edge labels on the path from the root to v; under this correspondence the set of nodes at level j is $L_1 \times \cdots \times L_j$; the root is the empty sequence ε. For an internal node v at depth j, its children are nodes of the form $v \circ a$ where $a \in L_{j+1}$. We denote their set by $\mathbf{child}(v) = \{v \circ a, a \in L_{j+1}\}$. The path from ε to v at depth j is equal to the sequence of nodes $v_{\leq 0}, v_{\leq 1}, \ldots, v_{\leq j}$ where $v_{\leq i}$ is the prefix of v of length i.

Usually in this paper, the sets $L_1, \ldots, L_d$ are all equal to the same set L and in this case $T(L_1 \times \cdots \times L_d)$ is denoted $T(L^d)$. Usually, L is a linearly ordered set and it is useful to visualize the planar drawing of T in which the left-to-right order of the children of an internal node corresponds to the total ordering on the edge labels. For a tree T and $i \geq 0$, $T^{\leq i}$ is the subtree of T consisting of nodes of depth at most i.

String decompositions, and tree decompositions A *decomposition* of a string x is a sequence $z_1, \ldots, z_r$ of strings such that $x = \odot_{i=1}^{r} z_i$. More generally if L is a linearly ordered set then an *L-sequence* $(z_i : i \in L)$ where each z_i is a string is a decomposition of x if $x = \odot_{i \in L} z_i$ where the concatenation is done in the order determined by L. Given a decomposition $(z_i : i \in L)$ of x, each substring z_i is naturally associated to a location interval $\mathbf{Loc}(z)_i = (s_i, t_i]$ where $s_i = \sum_{j \in L, j < i} |z_j|$ and $t_i = s_i + |z_i|$.

Supppose that $z = (z_v : v \in L_1 \times \cdots \times L_d)$ is a labeling of the leaves of $T = T(L_1 \times \cdots \times L_d)$ by strings. Given such a labeling, we extend it to all nodes of the tree so that for any node v, z_v is defined to be the concatenation of the strings z_ℓ where ℓ is a leaf below v and the concatenation is done in lexicographic order according to the leaves. If x_ε is the string labeling the root, then the T with the node labeling z is said to be a *decomposition tree for x*. Note that for any $j \in \{1, \ldots, d\}$, $z^j = (z_v : v \in L_1 \times \cdots \times L_j)$ is the decomposition of x corresponding to the level j nodes in the tree, and z^j is a refinement of the decomposition z^{j-1}. In a decomposition tree z, each node v corresponds to a specific fragment whose location in x is the interval $\mathbf{Loc}(z)_v = (s_v, t_v]$ where $s_v = \sum_{u \in L^d, u < v} |z_u|$ and $t_v = s_v + |z_v|$. Hence, $x_{\mathbf{Loc}(z)_v} = z_v$.

3.2 Edit distance and its representation in grid graphs

For $x \in \Sigma^*$, we consider three *edit operations* on x:

- **ins**(i, a) where $i \in \{1, \ldots, |x| + 1\}$ and $a \in \Sigma$, which means insert a immediately following the prefix of length $i - 1$. In the resulting sequence the i-th entry is a.

- **del**(i) where $i \in \{1, \ldots, |x|\}$, deletes the i-th entry of x.

- **sub**(i, b): replace x_i by b.

For strings x, y, the *edit distance of x and y*, $\mathbf{ED}(x, y)$, is the minimum length of a sequence of operations that transforms x to y. It is well-known and easy to show that $\mathbf{ED}(x, y) = \mathbf{ED}(y, x)$.

Representing edit distance by paths in weighted grids. We define **Grid** to be the directed graph whose vertex set $V(\mathbf{Grid})$ is the set $\mathbb{N} \times \mathbb{N}$ (*points*) and whose edge set $\mathbf{E}(\mathbf{Grid})$ consists of three types of directed edges: *horizontal edges* of the form $\langle i, j \rangle \to \langle i + 1, j \rangle$, *vertical edges* of the form $\langle i, j \rangle \to \langle i, j + 1 \rangle$ and *diagonal edges* of the form $\langle i, j \rangle \to \langle i + 1, j + 1 \rangle$ for any $i, j \geq \mathbb{N}$. For non-empty intervals $I, J \subseteq \mathbb{N}$ not-containing zero, we define the $\mathbf{Grid}_{I \times J}$ to be the subgraph of **Grid** induced on the set $(I \cup \{\min(I) - 1\}) \times (J \cup \{\min(J) - 1\})$, and for $P \subseteq E(\mathbf{Grid})$, the restriction of P to $I \times J$ is $P_{I \times J} = P \cap E(\mathbf{Grid}_{I \times J})$. We call $I \times J$ a *box*. A directed path from $\langle \min(I) - 1, \min(J) - 1 \rangle$ to $\langle \max(I), \max(J) \rangle$ is called a *spanning path* of $\mathbf{Grid}_{I \times J}$.

As is well known, the edit distance problem for a pair of strings x, y can be represented as a shortest path problem on a grid with weighted edges (see e.g. [LMS98]). The *grid of x, y*, $\mathbf{Grid}(x, y)$, is the subgraph $\mathbf{Grid}_{(0, |x|] \times (0, |y|]}$ with edge set $\mathbf{E}(x, y) \subseteq \mathbf{E}(\mathbf{Grid})$. For an edge $e = \langle i, j \rangle \to \langle i', j' \rangle$ in $\mathbf{Grid}(x, y)$, let $x_e = x_{i'}$ if $i' = i + 1$ and $x_e = \varepsilon$ if $i' = i$. Similarly, let $y_e = y_{j'}$ if $j' = j + 1$ and $y_e = \varepsilon$ if $j' = j$. We assign a cost to edge e to be 0 if $y_e = x_e$ and it is 1 otherwise. In particular, every horizontal edge and every vertical edge costs 1, and diagonal edges cost 0 or 1 depending on whether the corresponding symbols of x and y differ. An edge of non-zero cost is *costly*. If P is a set of edges, the *costly part of P*, $\mathbf{costly}(P)$, is the set of costly edges. Define the cost of P to be $\mathrm{cost}(P) = |\mathbf{costly}(P)|$.

We define an *annotated edge* to be a triple (e, a, b) where $a, b \in \Sigma \cup \{\varepsilon\}$. The *$(x, y)$-annotation of e* is the annotated edge (e, x_e, y_e), which is denoted $e^+(x, y)$. An annotated edge (e, a, b) is said to be *(x, y)-consistent* or simply *consistent* if $a = x_e$ and $\beta = y_e$. We emphasize that each edge e has a unique consistent annotation with respect to any given x and y.

10

For a set of edges P we write $P^+(x, y)$ for the set of annotated edges $\{e^+(x, y) : e \in P\}$. In particular if P is a path then $\mathbf{costly}(P^+(x, y))$ is the set of costly edges of P with their x, y-annotations. When the pair x, y of strings is fixed by the context (which is almost always the case) we write e^+ for $e^+(x, y)$ and for a set P of edges, we write P^+ for $P^+(x, y)$. In particular, $\mathbf{E}^+$ for the set $\{e^+ : e \in \mathbf{E}(x, y)\}$.

It is well known and easy to see that there is a correspondence between spanning paths of $\mathbf{Grid}(x, y)$ and sequences of edit operations that transform x to y where a sequence of k edit operations corresponds to a spanning path of cost k. Thus, we will refer to a spanning path of $\mathbf{Grid}(x, y)$ as an *alignment of x and y*. We have:

Proposition 3.1. $\mathbf{ED}(x, y)$ *is equal to the minimum cost of an alignment of x and y, i.e., the minimum over all alignments P of $|\mathbf{costly}(P)|$.*

We have the following:

Proposition 3.2.   *1. An alignment P of x and y is uniquely determined by $\mathbf{costly}(P)$.*

    2. For any alignment P of x and y, given the set $\mathbf{costly}(P^+)$ of annotated costly edges and either of the strings x and y, the other string is determined.

Proof. For the first part, an alignment P can be partitioned into subpaths (sets of edges) $P_0, P_1, \ldots, P_k$ where for each even i, P_i consists of 0-cost edges, and for i odd, P_i consists of costly edges. All the paths are nonempty except for (possibly) the first. Trivially $\mathbf{costly}(P)$ is the union of all the paths P_i for i odd. For each i even, we can determine P_i since we know the start and end vertex, and the path consists entirely of diagonal edges.

For the second part, we show that $\mathbf{costly}(P^+)$ and x determine y; the result with x and y exchanged is proved similarly. From the first part of the proposition $\mathbf{costly}(P)$ determines P. Let $e_1, \ldots, e_t$ be the subsequence of P obtained by deleting all vertical edges. Then $|y| = t$, since the undeleted edges are in 1-1 correspondence with the indices of y. Furthermore y_j is determined by the j-th undeleted edge as follows: if the j-th edge is costly, (either vertical, or a costly diagonal edge) then its y annotation is equal to y_j, and if the j-th edge is a non-costly diagonal edge $\langle i-1, j-1 \rangle \to \langle i, j \rangle$, then $y_j = x_i$. □

A pair x, y of strings together with a box $I \times J$ with $I \subseteq (0, |x|]$ and $J \subseteq (0, |y|]$ specifies the edit distance sub-problem $\mathbf{ED}(x_I, y_J)$. $\mathbf{Grid}_{I \times J}(x, y)$ denotes the (edge-weighted) sub-graph of $\mathbf{Grid}(x, y)$ induced on $(I \cup \{\min(I) - 1\}) \times (J \cup \{\min(J) - 1\})$.

There may be many optimal alignments. We will need a *canonical alignment* for each x and y that is unique. For any graph $\mathbf{Grid}_{I \times J}(x, y)$, define the *canonical alignment of $\mathbf{Grid}_{I \times J}(x, y)$* as follows: Associate each path in $\mathbf{Grid}$ to the sequence from $\{\mathbf{vertical}, \mathbf{diagonal}, \mathbf{horizontal}\}$ which records the edge types along the path. The *canonical alignment of $\mathbf{Grid}_{I \times J}(x, y)$* is the optimal spanning path of $\mathbf{Grid}_{I \times J}(x, y)$ that is lexicographically maximum with respect to the order $\mathbf{vertical} > \mathbf{diagonal} > \mathbf{horizontal}$. The *canonical alignment $\mathbf{canon}(x, y)$ of x and y* is the canonical alignment of $\mathbf{Grid}(x, y)$.

The proof of the following is left to the reader.

Proposition 3.3. *For strings x, y and box $I \times J \subseteq (0, |x|] \times (0, |y|]$, the (edge-weighted) graph $\mathbf{Grid}_{I \times J}(x, y)$ is isomorphic (in the graph theoretic sense) to $\mathbf{Grid}(x_I, y_J)$ and so $\mathbf{ED}(x_I, y_J)$ is equal to the length of the shortest spanning path of $\mathbf{Grid}_{I \times J}(x, y)$. Also, the canonical alignment of*

x_I and y_J is isomorphic to the canonical alignment of $\mathbf{Grid}_{I \times J}(x, y)$ (when viewed as paths of their respective graphs).

Let P be an alignment of x and y. A box $I \times J$ is *compatible with P* provided that P passes through $\langle \min(I) - 1, \min(J) - 1 \rangle$ and $\langle \max(I), \max(J) \rangle$, and for such a box, the *restriction* $P_{I \times J}$ *of P to $I \times J$* is the portion of P joining $\langle \min(I) - 1, \min(J) - 1 \rangle$ and $\langle \max(I), \max(J) \rangle$. This restriction is an alignment for $\mathbf{Grid}_{I \times J}(x, y)$. The following proposition is straightforward.

Proposition 3.4. *If P is an optimal alignment of x and y and $I \times J$ is compatible with P then $P_{I \times J}$ is an optimal alignment for the sub-problem $\mathbf{Grid}_{I \times J}(x, y)$.*

For strings x and y, we say a box $I \times J$ is (x, y)-compatible if $I \times J$ is compatible with the canonical alignment $\mathbf{canon}(x, y)$. We have:

Proposition 3.5. *Let x, y be strings and let $I \times J$ be a box that is (x, y)-compatible. The restriction $\mathbf{canon}(x, y)_{I \times J}$ is equal to the canonical alignment of $\mathbf{Grid}_{I \times J}(x, y)$.*

Proof. By Proposition 3.4, the restriction $\mathbf{canon}(x, y)_{I \times J}$ is an optimal spanning path for $\mathbf{Grid}_{I \times J}(x, y)$. We claim that this restriction is equal to the canonical alignment of $\mathbf{Grid}_{I \times J}(x, y)$. Otherwise we could modify $\mathbf{canon}(x, y)$ by replacing the subpath on $I \times J$ by the canonical alignment of $\mathbf{Grid}_{I \times J}(x, y)$ and the result would be lexicographically larger. $\qquad\square$

3.3 Computational Considerations

Our construction uses a finite field of large characteristics to design hierarchical mismatch recovery schemes (a certain kind of Hamming sketches). For that our algorithm picks a prime p of size roughly $n^{O(\log n)}$ during its initialization. Such a prime can be picked at random and tested for its primality using standard algorithms (randomized or deterministic). The prime p can be stored as a part of the sketch. All arithmetics over $\mathbb{F}_p$ can be done in time poly-logarithmic in p or n, respectively. Our sketch could be modified to use primes of only polynomial size but for the simplicity of presentation we use a large prime.

3.4 Function families and randomized functions

Let A, B be sets. We consider families $\mathbf{f} = (f_\rho : \rho \in \Gamma)$ of functions from A to B. The set Γ is the *randomizing parameter space* and the subscript $\rho \in \Gamma$ is the *randomizing parameter*. If μ is a probability distribution over Γ, μ induces a distribution on the function family $(f_\rho : \rho \in \Gamma)$, and we say that f_ρ is a randomized function. We often suppress the index ρ and say that f is a randomized function.

Evaluating a randomized function at a domain element x is in two parts. The first part is the selection of the parameter $\rho \in \Gamma$ according to μ, which is done by a randomized algorithm. Once ρ is selected, f_ρ is a fixed (deterministic) function and the second part is to evaluate $f_\rho(x)$; the algorithm for doing this may possibly be randomized. We say that (f_ρ, μ) is *efficiently computable* provided that the sampling of ρ according to μ and the evaluation $f_\rho(a)$ can be done in time polynomial in the size of the bit-representation of a.

A randomized $A \to B$ function $(\mathbf{f}, \mu)$ is *pairwise independent* provided that for all $a, a' \in A$ and $b, b' \in B$ such that $a \neq a'$ we have $\mathbf{Pr}_{\rho \sim \mu}[(f_\rho(a) = b) \wedge (f_\rho(a') = b')] = \frac{1}{|B|^2}$. We have:

Proposition 3.6 (Dietzfelbinger [Die96])**.** *For all positive integers m, n with m a power of 2 there is a pairwise independent family of functions $(\boldsymbol{h}, \mu)$ mapping $\{0, \ldots, n-1\}$ to $\{0, \ldots, m-1\}$ that is efficiently computable, where members of the family can be specified with $O(\log m + \log n)$ bits.*

We note here a version of the Chernoff-Hoeffding bound [AS08] that will be needed in our algorithm analysis.

Lemma 3.7. *Let $X_1, \ldots, X_r$ be i.i.d. 0-1 random variables with $\boldsymbol{Pr}[X_i = 1] = p$. For $\delta \geq 0$,*

$$\boldsymbol{Pr}[|\frac{1}{r}\sum_{i=1}^{r} X_i - p| \geq \delta] \leq 2e^{-2\delta^2 r}.$$

4   Three auxiliary procedures

The sketching scheme for edit distance that we present in Section 6 uses several auxiliary sketching schemes. Three of these are from previous work and one is new to this paper. In this section, we describe the key properties of the previous three schemes.

4.1   Fingerprinting

The *fingerprinting problem* for a Σ^* is to define a family of sketching functions that distinguishes between distinct elements of Σ^*. More formally, given a parameter n we want a family $\text{FINGERPRINT}_\rho(\cdot; n) : \Sigma_{\overline{n}}^{\leq n} \to \mathbb{Z}^+$ such that for any strings $x, y \in \Sigma_{\overline{n}}^{\leq n}$ with $x \neq y$

$$\mathbf{Pr}_\rho[\text{FINGERPRINT}_\rho(x; n) = \text{FINGERPRINT}_\rho(y; n)] < 1/n^4.$$

The following classic result of Karp-Rabin [KR87] provides an efficient fingerprinting scheme. We refer to it as the *Karp-Rabin fingerprint*.

Theorem 4.1. *There is an efficiently computable randomized function $\text{FINGERPRINT}_\rho(\cdot; n) : \Sigma_{\overline{n}}^{\leq n} \to \{1, \ldots, n^5\}$ such that for any $x, y \in \Sigma_{\overline{n}}^{\leq n}$ with $x \neq y$, $\boldsymbol{Pr}_\rho[\text{FINGERPRINT}_\rho(x; n) = \text{FINGERPRINT}_\rho(y; n)] < 1/n^4$. The number of bits needed to describe ρ is $O(\log n)$. The time to compute $\text{FINGERPRINT}_\rho(x; n)$ is $O(|x| \cdot \log^{O(1)} n)$.*

4.2   Threshold edit distance fingerprinting

In the *threshold edit distance fingerprinting* we are given a parameter n and a threshold parameter k. We want a family of sketching functions $\text{TED-FINGERPRINT}_\rho(\cdot; k, n) : \Sigma^* \to \mathbb{Z}^+$ such that for all $x, y \in \Sigma_{\overline{n}}^{\leq n}$ such that $\mathbf{ED}(x, y) \geq k$, $\mathbf{Pr}[\text{TED-FINGERPRINT}_\rho(x; k, n) = \text{TED-FINGERPRINT}_\rho(y; k, n)] \leq 1/n^4$, i.e. the sketch is very likely to distinguish strings x, y that are far. We also want that for some *inaccuracy gap $s > 1$*, for strings x, y such that $\mathbf{ED}(x, y) < \frac{k}{s}$, $\mathbf{Pr}[\text{TED-FINGERPRINT}(x) \neq \text{TED-FINGERPRINT}(y)]$ is small. Precisely we want that for all x, y:

$$\mathbf{Pr}[\text{TED-FINGERPRINT}_\rho(x) \neq \text{TED-FINGERPRINT}_\rho(y)] \leq \frac{\mathbf{ED}(x, y)}{k} \cdot s.$$

Note that the requirement becomes easier as s gets larger. The problem of constructing a threshold edit distance fingerprinting scheme with inaccuracy gap $s > 1$ is closely related to the problem of finding an approximate embedding function f that maps strings to vectors in $\mathbb{R}^d$ (for some d) so

that $\mathbf{ED}(x,y)$ is approximated by $|f(x) - f(y)|_1 = \sum_{i=1}^{d} |f(x) - f(y)|$ within a factor s. This latter problem was investigated by Ostrovsky and Rabani [OR07] and their results yield the following consequence:

Theorem 4.2. *There is an algorithm* OR-FINGERPRINT$(x; k, n)$ *for the threshold edit distance fingerprinting problem that for arbitrary $k, n \in \mathbb{N}$ has inaccuracy gap $s_{\mathrm{OR}} = 2^{O(\sqrt{\log(n) \log \log(n)})}$ and for any string $x \in \Sigma_n^{\leq n}$ gives a fingerprint of value at most $O(n^4)$. The fingerprinting algorithm uses a randomness parameter of length $O(\log^3 n)$ and runs in time polynomial in n and k.*

We refer to OR-FINGERPRINT$(x; k, n)$ as Ostrovsky-Rabani fingerprint.

We make a few remarks on this theorem.[2] [OR07] does not provide explicit bound on the amount of randomness needed. In particular, Lemma 9 samples random subsets and uses tail-bound inequalities to bound the probability of bad events. One can use $O(\log n)$-wise independent samples to reduce the necessary randomness [FNSS92].

4.3 The procedure BASIC-DECOMP

The previous sketch for edit distance of [BK23] gave sketches of size $\tilde{O}(k^2)$ that allowed for recovery of edit distance between strings x, y with $\mathbf{ED}(x,y) \leq k$. That work is the starting point for this paper, and one of the key procedures in their algorithm is an essential part of our sketch-and-recover scheme, and we now describe its important properties.

[BK23] gives a decomposition algorithm BASIC-DECOMP that takes a string x of length at most n and partitions x into a sequence of fragments where each fragment is represented by a small *grammar* of size $\tilde{O}(k)$. A *grammar* is a subset of *rules* from a certain domain of size polynomial in n. For us the actual meaning of the grammar is irrelevant; however we use a procedure BASIC-DECODE implicitly defined in [BK23] that takes as input a grammar and either outputs a string in $\Sigma^{\leq n}$ or UNDEFINED. We will represent grammars by their characteristic vector within their domain. We say that a bit-vector is *t-sparse* if it has at most t 1's in it.

We now restate the properties of algorithm BASIC-DECOMP in a form suitable for us. The algorithm takes as input a string x, parameter n and an integer *sparsity parameter* k, where $|x| \leq n$. We denote this by BASIC-DECOMP$(x; n, k)$.

BASIC-DECOMP$(x; n, k)$ outputs:

- A string decomposition of x, $z = (z_i : i \in W)$, where W is the set $\{0, 1\}^{\lceil \log n \rceil}$ with the lexicographic order. (Some strings may be empty.)

- A collection $\mathbf{grams} = (\mathbf{grams}_i : i \in W)$ of k-sparse bit-vectors of length $N = n^{60}$ such that if $z_i \neq \varepsilon$ then BASIC-DECODE$(\mathbf{grams}_i) = z_i$ (so $\mathbf{grams}_i$ is a k-sparse encoding of z_i) and if $z_i = \varepsilon$ then $\mathbf{grams}_i$ is the all 0 vector 0^N. Furthermore, $\mathbf{grams}_i$ has the following *minimality property*: for any bit-vector b that is bit-wise less than $\mathbf{grams}_i$, BASIC-DECODE(b) is UNDEFINED. (For technical convenience we require BASIC-DECODE(0^N) to be also UNDEFINED although 0^N represents the empty string.) Each $\mathbf{grams}_i$ is represented as a list of positions that are set to 1.

---

[2]We had difficulties reading the paper [OR07]. Indeed, there is a minor correctable issue in the way Lemmas 8 and 9 are presented in the paper. The lemmas claim output from ℓ_1 which allows for vectors with arbitrary real numbers. Indeed, because of the scaling in their proofs they do output vectors with real numbers. However, both lemmas need to be applied iteratively and they assume input to be a 0-1-vector. This disparity can be corrected using standard means but it requires additional effort.

Theorem 4.3. *The algorithm* BASIC-DECOMP *has the following properties for all n, k:*

1. *For any input x of length at most n, the running time of* BASIC-DECOMP$(x; n, k)$ *as well as the total number of ones in $\textbf{grams}_i$'s is bounded by $O(|x| \cdot \log^{O(1)} n)$.*

2. *For any pair of inputs x, y, suppose $z = (z_i : i \in W)$ and $\textbf{grams} = (\textbf{grams}_i : i \in W)$ is the output of* BASIC-DECOMP$(x; k)$, *and $z' = (z'_i : i \in W)$ and $\textbf{grams}' = (\textbf{grams}'_i : i \in W)$ is the output of* BASIC-DECOMP$(y; k)$.

   (a) *With probability at least $1 - s_{\text{split}} \cdot \frac{\textbf{ED}(x,y)}{k}$ where $s_{\text{split}} = O(\log^4 n)$, for all $i \in W$, the box $\textbf{Loc}(z)_i \times \textbf{Loc}(z')_i$ is (x, y)-compatible whenever z_i is non-empty, and z_i is non-empty iff z'_i is non-empty.*

   (b) *For all $i \in W$, $\textbf{Ham}(\textbf{grams}_i, \textbf{grams}'_i) \le s_{\text{E} \to \text{H}} \cdot \textbf{ED}(z_i, z'_i)$ where $s_{\text{E} \to \text{H}} = O(\log^2 n)$.*

The significance of Item 2a is that if for each $i \in W$, the box $\textbf{Loc}(z)_i \times \textbf{Loc}(z')_i$ is (x, y)-compatible then $\textbf{ED}(x, y) = \sum_{i \in W} \textbf{ED}(z_i, z'_i)$.

Note, for Item 1, [BK23] allows the decomposition procedure to fail with probability $O(1/n)$. We modify the procedure that instead of failing it produces the trivial decomposition of its input into fragments of length 1. When $x \ne y$, this increases the failure probability in Item 2a by a negligible amount. (For $x = y$, the procedure never fails.) For Item 2a, [BK23] does not specify the success probability explicitly the way we do but our formula is immediate from their analysis of probability of what they call *undesirable split*. The upper bound on the difference between two grammars in terms of edit distance of x and y is also implicit in the same analysis.

Given a grammar G by its sparse representation as a list of t positions of 1's, BASIC-DECODE(G) runs in time $O(t + |x|)$ if G represents a string x and in time $O(t)$ otherwise. A grammar representing x contains at most $s_{\text{E} \to \text{H}} \cdot |x|$ ones.

5 Sketch-and-recover schemes

We consider the following general setting: We have a universe of objects (such as strings, matrices, or graphs) for which we have a specific way of comparing two objects in the universe. In this paper the universe is strings over a given alphabet and for each pair x, y of strings, we are interested in determining $\textbf{ED}(x, y)$. A *sketch-and-recover* scheme consists of two algorithms: a *sketching algorithm* that takes a string x over some alphabet and produces a short *sketch* SK(x) and a *recovery algorithm* that takes a pair of sketches SK(x) and SK(y) and recovers the desired information about the relationship between x and y.

The sketch and recovery algorithms typically take auxiliary parameters. For edit distance, the auxiliary parameters are n, an upper bound on the length of strings the sketch will handle, and k, a threshold parameter such that given any two inputs x, y if $\textbf{ED}(x, y) \le k$ the recovery algorithm will determine $\textbf{ED}(x, y)$. (The precise requirements of our scheme will be given later.)

Sketch-and-recover schemes are typically randomized. We assume that the algorithms use a *randomizing parameter* ρ, which is computed from a publicly available random string so that the same value of ρ is used by the sketching and recovery algorithms. We allow the recovery algorithm to fail with a small probability over ρ (with output UNDEFINED). We usually suppress the parameters ρ, n and k and write simply SK(x). By the *total sketch* we mean the pair $(\rho, \text{SK}_\rho(x; k, n))$, the sketch together with its randomizing parameter.

We want to implement sketch-and-recover schemes by efficient algorithms. Here efficiency is measured by:

- The *sketch length* bit-length($\text{SK}_\rho(x; k, n)$), which is the number of bits in the binary encoding of $\text{SK}_\rho(x; k, n)$.

- The *randomness length* bit-length(ρ) is the number of bits for (the binary encoding of) the parameter ρ. We think of ρ as being determined by a randomized algorithm which runs using a shared public string of random bits. We could just take ρ to be this shared string, but in some cases the number of bits needed to represent ρ may be less than the number of bits needed to generate ρ. For example, if ρ is supposed to be uniformly distributed among s-bit primes, then bit-length(ρ) = s but the number of random bits needed by an algorithm to generate ρ is larger than s.

- The running time of the sketch and recovery functions.

Our primary focus here is on achieving small sketch length. We also want the running time of the sketch and recovery algorithms to be at most polynomial in the length of the strings being sketched, but we don't try to optimize the polynomial here.

In designing a sketch for recovering edit distance, we want our recovery algorithm to do more than just recover the edit distance of two strings x and y from their sketch; we would also like the recovery algorithm to provide enough information so that given one of the strings, and the output of the recovery algorithm, one can determine the other string. For example, in the related but easier case of sketches for Hamming distance, we want to recover the *mismatch information* for x, y, $\text{MIS}(x, y)$, which is the set $\{(i, x_i, y_i) : i \in I_{\neq}(x, y)\}$, where $I_{\neq}(x, y) = \{i : x_i \neq y_i\}$. We refer to (i, x_i, y_i) as a *mismatch triple* for x, y

The analog of mismatch information for edit distance is the sequence of edit operations which is determined by the *set of costly annotated edges of the canonical alignment of x and y*, as defined in Section 3.2. In proposition 3.2 it is shown that an alignment is completely determined by its set of costly edges and also that given the set of costly edges with their annotations and either string x or y, one can recover the other string. We require the alignment to be canonical because the optimal alignment is not, in general, unique and we need that the set being recovered is uniquely determined by x and y.

5.1 Hierarchical mismatch recovery

The problem of Hamming distance recovery can be viewed as the special case of edit distance recovery when the only edit operations allowed are substitutions (which preserve the position of letters in the string). To formalize the problem we fix an index set D and alphabet Γ and take as our universe the set of D-sequences over Γ. We want a sketch-and-recover scheme that supports *mismatch recovery*, which is the recovery of the mismatch information $\text{MIS}(u, w)$ defined in Section 5.

In the standard formulation, we have a parameter k and require recovery of the mismatch information $\text{MIS}(x, y)$ from the sketches of x and y whenever their Hamming distance is at most k. We will need a more general formulation of mismatch recovery in which we only require recovery of mismatch information for some indices. Those indices will be determined for each pair of strings u and w separately.

This more general formulation is called *targeted mismatch recovery*. Before giving a formal definition, we give a simple example. Consider sequences of length $n = m^2$ for some integer m,

where we think of a sequence u as the concatenation of m fragments $u^1, \ldots, u^m$ each of length m. For strings u, w we say that fragment i is overloaded if there is more than one mismatch in the fragment, and underloaded if there is at most one mismatch in the fragment. Let $F(u, w)$ be the set of mismatch indices belonging to underloaded fragments. The set $F(u, w)$ defines a targeted mismatch recovery problem where our goal is to provide sketching and recovery algorithms so that for any pair $u, w \in \Gamma^D$, given the sketches of u and w, the recovery algorithm finds all mismatch triples corresponding to $F(u, w)$. Thus one needs to recover up to m mismatches in underloaded fragments even if other fragments are overloaded fragments. This particular example has a simple sketch of size $\widetilde{O}(m)$ by separately constructing an $\widetilde{O}(1)$-size sketch of each fragment that supports recovery of a single error.

In general, a *targeted mismatch recovery* problem is specified by a target function F which for each pair $u, w \in \Gamma^D$, is a subset $F(u, w)$ of the set $I_{\neq}(u, w)$ of indices i such that $u_i \neq w_i$. We write $\mathrm{MIS}_F(u, w)$ as the set $\{(i, u_i, w_i) : i \in F(u, w)\}$, this is the subset of mismatch triples corresponding to indices in $F(u, w)$. The targeted mismatch recovery problem for a target function F is denoted $\mathbf{TMR}(F)$. The output of the RECOVER function applied to two sketches is required to be a set of triples (i, a, b) where $i \in D$ and $a, b \in \Gamma$. Such a triple is called a *mismatch triple*. The success conditions for $\mathbf{TMR}(F)$ for the pair u, w are:

Soundness. RECOVER($\mathrm{SK}(u), \mathrm{SK}(w)$) $\subseteq \mathrm{MIS}(u, w)$, i.e. every mismatch claim is correct.

Completeness. $\mathrm{MIS}_F(u, w) \subseteq$ RECOVER($\mathrm{SK}(u), \mathrm{SK}(w)$) so the algorithm recovers all mismatch triples indexed by $F(u, w)$.

Notice, we do not require that RECOVER($\mathrm{SK}(u), \mathrm{SK}(w)$) $\subseteq \mathrm{MIS}_F(u, w)$, i.e., we allow the recovery algorithm to output mismatch triples not indexed by $F(u, w)$ as long as they are correct mismatch triples. Thus, while Completeness depends on the target F, Soundness does not. A scheme for targeted mismatch recovery with target function F has failure probability at most δ provided that for any $u, w \in \Gamma^D$, with probability at least $1 - \delta$, both Completeness and Soundness hold. Here, the probability is taken over the randomness of the sketching and recovery algorithms. (Our recovery algorithms will actually be deterministic, so all the randomness is coming from the sketch.)

Hierarchical mismatch recovery (**HMR**) is a special case of targeted mismatch recovery It is applicable to D-sequences indexed by a product set $D = L_1 \times \cdots \times L_d$, which we view as the leaves of the tree $T = T(L_1 \times \cdots \times L_d)$ as in Section 3.1. The target function depends on a *capacity function* κ and an *overload parameter* R. The capacity function is a positive valued function defined on the levels of the tree with κ_j denoting the capacity for nodes at level j. We require $1 \leq \kappa_d \leq \kappa_{d-1} \leq \cdots \leq \kappa_0$. The overload parameter R is a positive integer.

The triple consisting of $L_1 \times \cdots \times L_d$, κ, and R determines a *hierarchical mismatch recovery problem* which is a targeted mismatch recovery problem whose target set on sequences u, w, denoted $F_{T, \kappa, R}(u, w)$, is defined as follows.

First, the *load function* $\widehat{\kappa}(u, w)$ induced by κ and the pair u, w is defined on the vertices of the tree T as follows:

- For a leaf $v \in L^d$, $\widehat{\kappa}_v = 1$ if $u_v \neq w_v$, and is 0 otherwise.

- For an internal node v at level $j < d$, $\widehat{\kappa}_v = \min(\kappa_j, \sum_{v' \in \mathbf{child}(v)} \widehat{\kappa}_{v'})$.

Clearly $\widehat{\kappa}_v \leq \kappa_j$ for every vertex v at level j. We say that an internal node v at level j is *R-overloaded*, if $\widehat{\kappa}_v \geq \frac{1}{R}\kappa_j$ and is *R-underloaded* otherwise. (Later we will fix the parameter R to be $4d$, and refer to nodes simply as overloaded or underloaded.)

Notice that a R-underloaded v satisfies $\widehat{\kappa}_v = \sum_{v' \in \mathbf{child}(v)} \widehat{\kappa}_{v'}$. A leaf is said to be *R-accessible* if its path to the root consists entirely of R-underloaded nodes.

We define $F_{T,\kappa,R}(u,w)$ to be the set R-accessible leaves ℓ where $u_\ell \neq w_\ell$. Intuitively, the leaves below a R-overloaded node are "crowded" with mismatches and we do not require them to be recovered. We denote by $\mathbf{HMR}(T,\kappa,R)$ the targeted mismatch recovery problem with target function $F_{T,\kappa,R}$. In Section 5.4 we will prove:

Theorem 5.1. *Let $T = T(L_1 \times \cdots \times L_d)$ be a level-uniform tree as defined in Section 3.1 and let $\kappa = (\kappa_0, \ldots, \kappa_d)$ be a capacity function $1 \leq \kappa_d \leq \kappa_{d-1} \leq \cdots \leq \kappa_0 \leq \prod_{j=1}^{d} |L_j|$. Assume that for all $j \in \{0, \ldots, d\}$, $|L_j|$ and all κ_j are powers of two. Let $\Gamma = \mathbb{F}_p$ where $p \geq 4\prod_{j=1}^{d} |L_j|^2$. There is a sketch-and-recover scheme for hierarchical mismatch recovery for $(L_1 \times \cdots \times L_d)$-sequences over Γ defined by procedures* HMR-SKETCH *and* HMR-RECOVER *that given $\delta > 0$ satisfies:*

1. *If $R \geq 4d$ then for any two inputs x and y, the probabilty that $\mathbf{HMR}(T, \kappa, R)$ fails to satisfy Soundness and Completeness is at most δ.*

2. *The sketch bit-size is $O(\kappa_0 \cdot \log |\Gamma| \cdot (\sum_{j=1}^{d} \log |L_j| + \log(1/\delta)))$ bits.*

3. *The sketching algorithm runs in time $O(\prod_{j=1}^{d} |L_j| \cdot \log^{O(1)} |\Gamma| \cdot \log(1/\delta))$.*

4. *The recovery algorithm runs in time $O(\kappa_0 \cdot \log^{O(1)} |\Gamma| \cdot \log(1/\delta))$.*

5. *If the $(L_1 \times \cdots \times L_d)$-sequence u is given via the sparse representation $\{(j, u_j) : j \in \mathbf{supp}(u)\}$ where $\mathbf{supp}(u) = \{\ell \in L_1 \times \cdots \times L_d : u_\ell \neq 0\}$ then the time to construct the sketch is $O((\kappa_0 + |\mathbf{supp}(u)|) \cdot \log^{O(1)} |\Gamma| \cdot \log(1/\delta))$.*

6. *The number of mismatch pairs output by the algorithm is at most the capacity κ_0 of the root.*

The sketch HMR-SKETCH(u) depends on $L_1 \times \cdots \times L_d$, κ, R, and the error parameter δ and when we apply this sketch in our edit distance algorithm we denote it by HMR-SKETCH$(u; L_1 \times \cdots \times L_d, \kappa, R, \delta)$ for the sketch of u and HMR-RECOVER$(u', w'; L_1 \times \cdots \times L_d, \kappa, R, \delta)$ for the recovery function which takes as input the sketches u' and w' output by HMR-SKETCH on strings u' and w'. The dependence of both the sketching and recovery functions on the random parameter ρ is left implicit.

We mention that our application will require a sketch-and-recovery scheme for the trivial case that $d = 0$. In this case the tree is a single node and $u \in \Gamma$ and we take the sketch to consist of the value itself, and recovery is trivial.

5.2 Superposition sketch-and-recover schemes for targeted mismatch recovery

The next two subsections present a general approach to targeted mismatch recovery. We apply this approach in Section 5.4 to construct the sketch-and-recover scheme HMR-SKETCH and HMR-RECOVER for hierarchical mismatch recovery and prove Theorem 5.1.

We make the following assumptions:

- $D = \{0, \ldots, |D| - 1\}$.

- Γ is the field $\mathbb{F}_p$ for some prime larger than $|D|$ so $D \subseteq \Gamma$.

If these assumptions do not hold we can often reduce our situation to one where it does hold. We do not need D to be integers; it is enough that there is an easily computable 1-1 mapping m from D to the nonnegative integers. Letting $m = \max_{j \in D} m(j) + 1$, we can think of D as a subset of $\{0, \ldots, m-1\}$, and enlarge the domain to $\{0, \ldots, m-1\}$ defining any D-sequence to be 0 on those indices outside of D. Similarly we can replace the range Γ by $\mathbb{F}_p$ for some p whose size is at least $\max\{|\Gamma|, |D|\}$, where we interpret Γ as a subset of $\mathbb{F}_p$ via some easily computable 1-1 map.

Let u, w be D-sequences over Γ. Here we use a basic technique from [PL07] (see also [CEPR09, CKP19] for related constructions), that allows for the recovery of $\mathrm{MIS}(u, w)$ at a specific index i. For a parameter $\alpha \in \Gamma$, the *trace* of $u \in \Gamma^D$ (with respect to α), denoted $\mathbf{tr}_\alpha(u)$ is the D-sequence over Γ^4 where for each $i \in D$, $\mathbf{tr}_\alpha(u)_i$ has entries:

$$
\begin{aligned}
\mathbf{tr}_\alpha(u)_{i,\text{value}} &= u_i, \\
\mathbf{tr}_\alpha(u)_{i,\text{product}} &= i \cdot u_i, \\
\mathbf{tr}_\alpha(u)_{i,\text{square}} &= u_i^2, \\
\mathbf{tr}_\alpha(u)_{i,\text{hash}} &= \alpha^i u_i.
\end{aligned}
$$

All the calculations are done over $\mathbb{F}_p$. We refer to a vector in Γ^4 with indices from {value, product, square, hash} as a *trace vector* and we refer to α as the *trace parameter*

For $u, w \in \Gamma^D$, the *trace difference of* u, w is $\Delta_\alpha(u, w) = \mathbf{tr}_\alpha(u) - \mathbf{tr}_\alpha(w)$. Here, for each i, $\Delta_\alpha(u, w)_i = \mathbf{tr}_\alpha(u)_i - \mathbf{tr}_\alpha(w)_i$ is a trace vector obtained by coordinate-wise subtraction.

Define the function **restore** which maps trace vectors t to Γ^3 as follows:

$$
\begin{aligned}
\mathbf{restore}(t)_{\text{index}} &= \frac{t_{\text{product}}}{t_{\text{value}}} \\
\mathbf{restore}(t)_{\text{x-val}} &= \frac{t_{\text{square}} + t_{\text{value}}^2}{2 t_{\text{value}}} \\
\mathbf{restore}(t)_{\text{y-val}} &= \frac{t_{\text{square}} - t_{\text{value}}^2}{2 t_{\text{value}}}.
\end{aligned}
$$

It is easy to check:

Proposition 5.2. *The mismatch information for (u, w) at i is completely determined by $\Delta_\alpha(u, w)_i$ as follows: i is a mismatch index of u, w if and only if $\Delta_\alpha(u, w)_i \neq 0$ and for such an i,*

$$
\begin{aligned}
\boldsymbol{restore}(\Delta_\alpha(u, w))_{i,index} &= i, \\
\boldsymbol{restore}(\Delta_\alpha(u, w))_{i,x\text{-}val} &= u_i, \\
\boldsymbol{restore}(\Delta_\alpha(u, w))_{i,y\text{-}val} &= w_i,
\end{aligned}
$$

and therefore $\boldsymbol{restore}(\Delta_\alpha(u, w))_i = (i, u_i, w_i) = \mathrm{MIS}(u, w)_i$.

We remark that the division by 2 in the definition of **restore** is the reason why we need that Γ does not have characteristic 2. Also, note that $\mathbf{tr}_\alpha(u)_{i,\text{hash}}$ is not used in **restore**, but is used later to check soundness. In standard binary representation of integers, all arithmetic operations over $\mathbb{F}_p$ that are necessary to compute trace or its restoration at a single coordinate can be computed in time $O(\log^{O(1)} p)$.

We are now ready to define the class of *superposition sketches* for functions from D to Γ.

Definition 1 (Superposition sketch). *Let S be a set, $h : D \to S$ and $\alpha \in \Gamma$. The superposition sketch induced by (α, h) is the function $\boldsymbol{tr}_{\alpha,h}$ that maps $u \in \Gamma^D$ to $\boldsymbol{tr}_{\alpha,h}(u) \in (\Gamma^4)^S$ where for $j \in S$:*

$$\boldsymbol{tr}_{\alpha,h}(u)_j = \sum_{i \in h^{-1}(j)} \boldsymbol{tr}_{\alpha}(u)_i.$$

In words, the function h is used to partition D into $|S|$ classes $h^{-1}(j)$, and $\boldsymbol{tr}_{\alpha,h}(u)$ at $j \in S$ is the sum of the trace vectors of u corresponding to indices of D in the class $h^{-1}(j)$. The size (in bits) of the output is $O(|S| \log |\Gamma|)$.

Note that we can compute the superposition sketch of any D-sequence over Γ easily: Initialize $\boldsymbol{tr}_{\alpha,h}(u)$ to all zero and then for each $i \in D$ add the trace vector $\boldsymbol{tr}_{\alpha}(u)_i$ to $\boldsymbol{tr}_{\alpha,h}(u)_j$ where $j = h(i)$.

For $u, w \in \Gamma^D$ and $h : D \to \Gamma$, a mismatch index $i \in D$ is *recoverable for u, w, h* if $h^{-1}(h(i)) \cap I_{\neq}(u, w) = \{i\}$. We now define a procedure $\text{RECOVER}_{\alpha,h}$ that recovers all recoverable indices.

Algorithm 1 $\text{RECOVER}_{\alpha,h}(\boldsymbol{tr}_{\alpha,h}(u), \boldsymbol{tr}_{\alpha,h}(w), \alpha, h)$

Input: Traces $\boldsymbol{tr}_{\alpha,h}(u), \boldsymbol{tr}_{\alpha,h}(w)$ for two strings $u, w \in \Gamma^D$, trace parameters $\alpha \in \Gamma$, $h : D \to S$.
Output: The set $M_{\alpha,h}(u, w)$ of mismatch triples.

---

1 Let $\Delta_{\alpha,h}(u, w) = \boldsymbol{tr}_{\alpha,h}(u) - \boldsymbol{tr}_{\alpha,h}(w)$.
2 Let J be the set of $j \in S$ such that $\Delta_{\alpha,h}(u, w)_{j,\text{value}} \neq 0$.
3 **Rebuilding step:** For each $j \in J$ let $z_j = \textbf{restore}(\Delta_{\alpha,h}(u, w)_j)$.
4 **Filtering step:** Let $I = \{j \in J, \; z_{j,\text{index}} < |D| \wedge \Delta_{\alpha,h}(u, w)_{j,\text{hash}} = \alpha^{z_{j,\text{index}}}(z_{j,\text{x-val}} - z_{j,\text{y-val}})\}$.
5 Return $M_{\alpha,h}(u, w) = \{z_j : j \in I\}$.

In the rebuilding step, the algorithm produces a list of mismatch triples by applying **restore** to every trace vector it can among the $\Delta_{\alpha,h}(u, w)_j$. In the filtering step, it eliminates some of these mismatch triples, and then it outputs the rest. The following lemma shows that (1) The rebuilding step produces all mismatch triples corresponding to recoverable indices (and possibly some others) and (2) The filtering step with high probability eliminates all mismatch triples corresponding to indices that are not recoverable, so Soundness holds with high probability.

Lemma 5.3. *Let $u, w \in \Gamma^D$ and $h : D \to S$ be fixed. Let I and J be as in the $\text{RECOVER}_{\alpha,h}(u, w)$. For each $j \in S$:*

1. *If $|h^{-1}(j) \cap I_{\neq}(u, w)| = 0$ then $j \notin J$.*

2. *If $|h^{-1}(j) \cap I_{\neq}(u, w)| = 1$ then $z_j \in M_{\alpha,h}(u, w)$ and $z_j = (i, u_i, w_i)$ where i is the unique mismatch index such that $h(i) = j$.*

3. *The probability that $M_{\alpha,h}(u, w)$ outputs a triple that is not in $\text{MIS}(u, w)$ (i.e. that Soundness fails) is at most $\frac{(|D|-1) \cdot |S|}{|\Gamma|}$ over a uniformly random choice of $\alpha \in \Gamma$.*

Proof. For the first part, if $|h^{-1}(j) \cap I_{\neq}(u, w)| = 0$ then $\Delta_{\alpha,h}(u, w)_j$ is equal to $(0, 0, 0, 0)$ and so $j \notin J$.

For the second part, suppose i is the unique mismatch index in $h^{-1}(j)$. Then $\Delta_{\alpha,h}(u, w)_j = \Delta_{\alpha}(u, w)_i$. Since $u_i \neq w_i$, $\Delta_{\alpha}(u, w)_{i,\text{value}} \neq 0$ and so $z_j = \textbf{restore}(\Delta_{\alpha,h}(u, w)_i) = (i, u_i, w_i) \in$

$\text{MIS}(u, w)$, by Proposition 5.2. This triple is not eliminated by the filtering step since $z_{j,\text{index}} = i < |D|$ and $\Delta_\alpha(u, w)_{i,\text{hash}} = (u_i - w_i) \cdot \alpha^i = (z_{j,\text{x-val}} - z_{j,\text{y-val}}) \cdot \alpha^{z_{j,\text{index}}}$.

For the third part, we only need to look on z_j where $|h^{-1}(j) \cap I_{\neq}(u, w)| > 1$ and show that they will be filtered out. Namely, we will show that $\mathbf{Pr}_\alpha[z_j \in M_{\alpha,h}] \leq \frac{|D|-1}{|\Gamma|}$. Then a union bound over $j \in S$ completes the proof. If $\Delta_{\alpha,h}(u, w)_{j,\text{value}} = 0$ then $j \notin J$ irrespective of α. Otherwise the filtering step excludes z_j unless $z_{j,\text{index}} < |D|$ and $\Delta_{\alpha,h}(u, w)_{j,\text{hash}} - \alpha^{z_{j,\text{index}}}(z_{j,\text{x-val}} - z_{j,\text{y-val}}) = 0$. Letting I' be the set of mismatch indices in $h^{-1}(j)$, the left hand side of this equation equals $\sum_{i' \in I'} \alpha^{i'}(u_{i'} - w_{i'}) - \alpha^{z_{j,\text{index}}}(z_{j,\text{x-val}} - z_{j,\text{y-val}})$. Here, $z_{j,\text{index}}, z_{j,\text{x-val}}, z_{j,\text{y-val}}$ do not depend on α so for fixed h, u, w this is a nonzero polynomial in α of degree at most $|D| - 1$ (nonzero because the sum has at least two non-zero terms, and at most one is cancelled by the subtracted term). This polynomial has at most $|D| - 1$ roots, so the probability that α is a root is at most $\frac{|D|-1}{|\Gamma|}$. $\qquad \square$

The following result gives upper bounds on the running time of the sketch and recover algorithms, and on the space needed for the sketch

Proposition 5.4. *Let $\Gamma = \mathbb{F}_p$ and let $D = \{0, \ldots, |D|-1\}$ with $|D| \leq p$. Let S be a set, $h : D \to S$ and $\alpha \in \Gamma$. The superposition sketch $\mathbf{tr}_{\alpha,h}$ maps a D-sequence u over Γ to a sketch of bit-length $O(|S| \log |\Gamma|)$. The running time for the sketch algorithm is $O(|D| \cdot T)$ and the running time of the recover algorithm is $O(|S| \cdot T)$, where T is an upper bound on the time to perform a single arithmetic operation over Γ and evaluate h at a single point.*

Furthermore if a D-sequence u over Γ is given via a sparse representation, via $\{(j, u_j) : j \in \mathbf{supp}(u)\}$ where $\mathbf{supp}(u) = \{j \in D : u_j \neq 0\}$ then the running time of the sketch algorithm is $O((|S| + |\mathbf{supp}(u)|) \cdot T)$.

5.3 Randomized superposition sketches

In order to apply the superposition sketch we need to select a good h. However, one can hardly hope that if S is comparable in size to $\text{MIS}(u, w)$ then one can find a single $h : D \to S$ for which all mismatch indices $I_{\neq}(u, w)$ will be recoverable. Hence, we will try superposition sketches for *multiple randomly chosen h's*. Fix a (small) family $H \subseteq \{h : D \to S\}$ and a probability distribution μ on H (not necessarily uniform). For $\beta \leq 1$, we say that i is *β-recoverable for u, w, μ* provided that for $h \sim \mu$, the probability that i is recoverable for u, w, h is at least β. (Recall that if i is recoverable for u, w, h then for any choice of α, the output of recovery procedure from the sketches $\mathbf{tr}_{\alpha,h}(u)$ and $\mathbf{tr}_{\alpha,h}(w)$ includes the triple (i, u_i, w_i).)

We select hash functions $h_1, \ldots, h_\ell$ independently according to μ, for some *redundancy parameter* ℓ. We also select trace parameters $\alpha_1, \ldots, \alpha_\ell$ uniformly at random from Γ. The sketch of u consists of the sequences $h_1, \ldots, h_\ell$ and $\alpha_1, \ldots, \alpha_\ell$ together with $\mathbf{tr}_{\alpha_1,h_1}(u), \ldots, \mathbf{tr}_{\alpha_\ell,h_\ell}(u)$. For the recovery algorithm, given the sketches for u and w we compute each of the sets $M_{\alpha_i,h_i}(u, w)$ for $i \in [\ell]$ and define $M_{\alpha_1,\ldots,\alpha_\ell,h_1,\ldots,h_\ell}(u, w)$ to be the set of triples that appear in strictly more than half of the sets.

We refer to a scheme of the above type as a *randomized superposition scheme*. Such a scheme is determined by the distribution (H, μ) over hash functions and the redundancy ℓ.

The size of the sketch in bits (not including the description of the hash functions used) is $O(\ell \cdot |S| \cdot \log |\Gamma|)$. The description of the hash functions depends on the method used to represent members of H. For standard explicit choices of H (such as explicit families of $O(1)$-wise independent functions), members of h are represented in $O(\log |H|)$ space.

21

Proposition 5.5. *Let D and S be sets and let Γ be a field of size at least $4(|D| - 1)| \cdot |S|$. For each pair of strings $u, w \in \Gamma^D$, suppose $F(u, w)$ is a subset of D. Let (H, μ) be a distribution over hash functions from D to S. Suppose that for every u, w, every index belonging to $F(u, w)$ is 3/4-recoverable for u, w, μ. Then for any $\delta > 0$, the superposition sketch using (H, μ) with redundancy $\ell \geq 8(\ln |D| + \ln \frac{1}{\delta} + 2)$ satisfies the Completeness and Soundness conditions for F with failure probability at most δ.*

Proof. We show that each of Soundness and Completness fail with probability at most $\delta/2$.

To bound the probability that Completeness fails, we claim that for each 3/4-recoverable mismatch index $i \in D$, it is the case that for every fixed choice of $\alpha_1, \ldots, \alpha_\ell$, the probability (with respect to the choice of $h_1, \ldots, h_\ell$) that $M_{\alpha_1, \ldots, \alpha_\ell, h_1, \ldots, h_\ell}(u, w)$ does not include index i is at most $\delta/(2D)$. If this claim is true then summing over the at most $|D|$ mismatch indices that are 3/4-recoverable, the probability that the output of the scheme omits at least one such index is at most $\delta/2$.

So we verify the claim. Fix $\alpha_1, \ldots, \alpha_\ell$ and a 3/4-recoverable mismatch index $i \in D$. For $j \in \{1, \ldots, \ell\}$, let Z_j be the 0-1 indicator of the event that M_{α_j, h_j} includes index j. Since j is 3/4-recoverable, Z_j is 1 with probability at least 3/4. Let $Z = \sum_{j=1}^{\ell} Z_j$. The expectation of Z is at least $3\ell/4$. Index i is not output by the algorithm if $Z \leq \ell/2$. By the Chernoff-Hoeffding bound (Lemma 3.7), the probability that $Z \leq \ell/2$ is at most $2e^{-\ell/8} \leq e^{-\ln|D| - \ln(1/\delta) - 1} \leq \frac{\delta}{2D}$, as required to prove the claim.

Next we bound the probability that Soundness fails. For each $j \in \{1, \ldots, \ell\}$, let Y_j be the indicator of the event that $M_{\alpha_j, h_j}(u, w)$ contains at least one triple that is not in $\text{MIS}(u, w)$. By part 3 of Lemma 5.3, for any fixed value of h_j, the probability, with respect to α_j that M_{α_j, h_j} contains any triples that do not belong to $\text{MIS}(u, w)$ is at most $\frac{(|D| - 1)|S|}{|\Gamma|}$ which is at most 1/4 by hypothesis. Let $Y = \sum_j Y_j$. If $Y \leq \ell/2$ then there is no incorrect triple that is in the output of more than $\ell/2$ out of the ℓ runs, and so it suffices to show that $\Pr[Y > \ell/2] \leq \frac{\delta}{2}$. We have $\mathbb{E}[Y] = \sum_j \mathbb{E}[Y_j] \leq \ell/4$. By the Chernoff-Hoeffding bound, the probability that the total number of such triples exceeds $\ell/2$ is at most $2e^{-\ell/8} \leq \frac{\delta}{2}$, as required. $\square$

As an example we give a simple application of random superposition schemes that recover all mismatch triples (with high probability) whenever $\textbf{Ham}(u, w) \leq K$. As mentioned in the introduction better sketches already exist.

Corollary 5.6. *For any $C > 1$, there is a randomized superposition scheme that given alphabet Γ of size at most n, domain D, $|D| \leq n$, and positive integer K, for any string from Γ^D produces a sketch of size $O(K \log^2 n + \log^2 n)$, and on input of two sketches for strings $u, w \in \Gamma^D$ with $\textbf{Ham}(u, w) \leq K$ recovers all mismatches with probability $\geq 1 - 1/n^C$.*

Proof. Choose S to be a set of size between $4K$ and $8K$. Let H be a pairwise independent family of hash functions from D to S. By Proposition 3.6 we can describe an $h \in H$ by $O(\log(n))$ bits, and the superposition sketch induced by h requires $O(K \log n)$ bits.

We claim that for any $u, w \in \Sigma^D$ such that $\textbf{Ham}(u, w) \leq K$ every mismatch index is 3/4-recoverable for u, w, μ. For $h \sim \mu$, the probability that i is not recoverable for u, w, h is as most $\sum_{i'} \Pr[h(i) = h(i')]$ where the sum is over $i' \neq i$ such that i' is a mismatch index. By pairwise independence, $\Pr[h(i) = h(i')] = 1/|S|$ and so the probability that i is not recoverable for u, w, h is at most $K/|S| \leq 1/4$.

The result now follows immediately from Proposition 5.5. $\square$

5.4 Proof of Theorem 5.1

In this section we show how to apply the randomized superposition schemes of Section 5.3 to construct a sketch-and-recover scheme for the hierarchical mismatch recovery and prove Theorem 5.1. The reader should review the set-up for hierarchical mismatch recovery in Section 5.1.

Recall that the scheme depends on the domain $L_1 \times \cdots \times L_d$, the capacity function κ where $\kappa = (\kappa_0, \ldots, \kappa_d)$, for each j, $\kappa_j \geq \kappa_{j+1}$ and an overload parameter R which we fix to be $4d$. We view $L_1 \times \cdots \times L_d$ as the set of leaves of $T = T(L_1 \times \cdots \times L_d)$. In preparation for describing the sketch-and-recover scheme, each node v at level $j < d$ of T is associated with a set of *buckets* which are ordered pairs (v, i) where $1 \leq i \leq \kappa_j$. We refer to (v, i) as a v-bucket, and a bucket of the root is a *root-bucket*. Each leaf ℓ has only one bucket, $(\ell, 1)$.

The hash functions of our superposition scheme are **leaf→root** functions, which are functions that map the set of leaves $L_1 \times \cdots \times L_d$ to the set of root-buckets. We will use Proposition 5.5 to prove that the scheme works by:

1. Describing a distribution μ over **leaf→root** functions.

2. Showing that for any pair of strings u, w, every leaf that belongs to $F_{T(L_1 \times \cdots \times L_d), \kappa, R}(u, w)$ is 3/4-recoverable for u, w, μ.

To describe the distribution μ on **leaf→root** functions, we consider a specific representation of a **leaf→root** functions, and for this we need the notions of a *trajectory* and *routing functions*.

For a leaf $\ell \in \prod_{j=1}^{d} L_j$, the modes on the path from ℓ to the root ε in T are the nodes identified by the strings $\ell_{\leq d}, \ell_{\leq d-1}, \ldots, \ell_0$. We define a *trajectory for* ℓ to be a sequence of buckets one for each node on the path from ℓ to ε, $(\ell_{\leq d}, i_d), (\ell_{\leq d-1}, i_{d-1}), \ldots, (\ell_0, i_0)$ where $i_j \in \{1, \ldots, \kappa_j\}$. A trajectory is uniquely determined by the leaf ℓ and the sequence of indices $(i_d, i_{d-1}, \ldots, i_0)$. Note that i_d must equal 1.

We want a way to specify a trajectory for every leaf. We do this using a collection $r = (r_j : j < d)$ of *routing functions*, one for each internal level of the tree. The routing function r_j is a function from $L_{j+1} \times \{1, \ldots, \kappa_{j+1}\}$ to $\{1, \ldots, \kappa_j\}$. For each v at level j r_j is used to specify a function that maps buckets corresponding to children of v to buckets of v as follows: for $a \in L_{j+1}$ and $i \in \{1, \ldots, \kappa_{j+1}\}$, the bucket $(v \circ a, i)$ is mapped to $(v, r_j(a, i))$. Thus the collection of routing functions determines a trajectory for every leaf ℓ with sequence of indices $i_d(\ell) = 1$ and for $j < d$, $i_j(\ell) = r_j(\ell_{j+1}, i_{j+1}(\ell))$. This induces the **leaf→root** mapping that maps each $\ell \in \prod_{j=1}^{d} L_j$ to the bucket $(\varepsilon, i_0(\ell))$.

We are now ready to specify the distribution μ. For each level $0 \leq j < d$, let $H_j = \{h : L_{j+1} \times \{1, \ldots, \kappa_{j+1}\} \to \{1, \ldots, \kappa_j\}\}$ be a pairwise independent family of routing functions for level j. Independently select $r_0, \ldots, r_{d-1}$ from $H_1, \ldots, H_{d-1}$. The distribution μ on **leaf→root** functions is the distribution induced by the selection of $r_0, \ldots, r_{d-1}$. Since $|L_{j+1}|$ and all values of κ_j are powers of 2, by Proposition 3.6, we can choose H_j so that members of H_j can be indexed with $O(\log|L_{j+1}| + \log \kappa_{j+1} + \log \kappa_j)$ bits. Thus the total number of bits to represent a **leaf→root** function in the family is $O(\sum_{j=1}^{d} \log|L_j| + d \log \kappa_0))$.

Lemma 5.7. *For $T, L_1, \ldots, L_d$ and κ as in Theorem 5.1, let μ be the distribution on **leaf→root** functions induced by choosing routing functions $r_0, \ldots, r_{d-1}$ independently from pairwise independent distribution. For any two strings u, w in $\Gamma^{L_1 \times \cdots \times L_d}$, every $\ell \in F_{T, \kappa, 4d}(u, w)$, i.e., every $4d$-accessible mismatch leaf, is 3/4-recoverable for u, w, μ.*

Proof. Let $\ell \in L_1 \times \cdots \times L_d$ be a $4d$-accessible leaf with respect to u, w where $u_\ell \neq w_\ell$. We must show that ℓ is $3/4$-recoverable. Recall that ℓ is $4d$-accessible if each node along the path from ℓ to root is $4d$-underloaded, i.e., for each $j < d$, $\widehat{\kappa}_{\ell_{\leq j}} < \kappa_j/4d$. Let $(r_j : 0 \leq j < d)$ be the sequence of random routing functions selected as above and let f be the induced **leaf→root** function.

Let $(i_d, \ldots, i_0)$ denote the sequence of indices $(i_d(\ell), \ldots, i_0(\ell))$ for the trajectory of ℓ. This is a random variable depending on the choice of $r_0, \ldots, r_{d-1}$. By definition, ℓ is not recoverable if and only if there is a mismatch leaf $\ell' \neq \ell$ such that $f(\ell) = f(\ell')$. If $\ell' \neq \ell$ is a leaf such that $f(\ell') = f(\ell)$ then the trajectories of ℓ and ℓ' have non-empty intersection. We say that ℓ and ℓ' *merge at level j* if they are in different buckets at level $j + 1$, but in the same bucket at level j. (Note that once the trajectories merge, they remain the same all the way to the root.)

For $j \in \{0, \ldots, d-1\}$, let MERGE$_j$ be the event that there is a mismatch leaf $\ell' \neq \ell$ that merges with ℓ at level j. We now fix j and prove that $\mathbf{Pr}[\text{MERGE}_j] \leq 1/4d$. This will finish the proof, since summing over all the levels, we will get that the probability that ℓ is not recoverable is at most $1/4$.

We condition on the $r_{j+1}, \ldots, r_{d-1}$, which determines the trajectory of all leaves up to level $j + 1$. In particular this determines $i_{j+1}, \ldots, i_d$,

Consider the set of child buckets of $\ell_{\leq j}$. These have the form $(\ell_{\leq j} \circ a, i)$ where $(a, i) \in L_{j+1} \times \{1, \ldots, \kappa_{j+1}\}$. This includes the bucket $(\ell_{\leq j} \circ \ell_{j+1}, i_{j+1})$ on the trajectory of ℓ.

Say that a bucket (v, i) is *occupied* if it lies on the trajectory of some mismatch leaf. Let OCC be the set of pairs $(a, i) \neq (\ell_{j+1}, i_{j+1})$ such that $(\ell_{\leq j} \circ a, i)$ is occupied. The event MERGE$_j$ is equivalent to the event that there is an $(a, i) \in \text{OCC}$ such that $r_j(a, i) = r_j(\ell_{j+1}, i_{j+1})$. For each $(a, i) \in \text{OCC}$, $\mathbf{Pr}[r_j(a, i) = r_j(\ell_{j+1}, i_{j+1})] = \frac{1}{\kappa_j}$ since r_j is a pairwise independent map whose range has size κ_j, and so the conditional probability of MERGE$_j$ given $r_{d-1}, \ldots, r_{j+1}$ is at most $|\text{OCC}|/\kappa_j$.

We need to upper bound $|\text{OCC}|$. Let $\mathbf{occ}(v)$ denote the number of occupied v-buckets. In the above analysis $|\text{OCC}| = \sum_{a \in L_{j+1}} \mathbf{occ}(\ell_{\leq j} \circ a) - 1$. We claim:

Proposition 5.8. *For any choice of routing functions $r_{d-1}, \ldots, r_0$:*

1. *For any internal node v at level $j' < d$, $\mathbf{occ}(v) \leq \min(\kappa_{j'}, \sum_{v' \in \mathbf{child}(v)} \mathbf{occ}(v'))$.*

2. *For any node v at level $j' \leq d$, $\mathbf{occ}(v) \leq \widehat{\kappa}(v)$.*

Proof. For the first part, let v be an internal node. Then $\mathbf{occ}(v)$ is trivially at most $\kappa_{j'}$. Also, a v-bucket is occupied if and only if some occupied child maps to it, so $\mathbf{occ}(v) \leq \sum_{v' \in \mathbf{child}(v)} \mathbf{occ}(v')$.

For the second part, if v is a leaf then $\mathbf{occ}(v) = 1$ if v is a mismatch leaf and 0 otherwise, so $\mathbf{occ}(v) = \widehat{\kappa}(v)$. If v is an internal node, the first part implies $\mathbf{occ}(v) \leq \min(\kappa_{j'}, \sum_{v' \in \mathbf{child}(v)} \mathbf{occ}(v'))$ and applying induction and the definition of $\widehat{\kappa}(v)$ we have $\min(\kappa_{j'}, \sum_{v' \in \mathbf{child}(v)} \widehat{\kappa}(v')) = \widehat{\kappa}(v)$. ☐

Thus $|\text{OCC}| \leq \sum_{a \in L_{j+1}} \widehat{\kappa}(\ell_{\leq j} \circ a) - 1$. We know that $\widehat{\kappa}(\ell_{\leq j}) \leq \kappa_j/4d$ and in particular, $\widehat{\kappa}(\ell_{\leq j}) < \kappa_j$. Hence, $\widehat{\kappa}(\ell_{\leq j}) = \sum_{a \in L_{j+1}} \widehat{\kappa}(\ell_{\leq j} \circ a) > |\text{OCC}|$. It follows that for any choice of the routing functions $r_{d-1}, \ldots, r_{j+1}$, $|\text{OCC}| < \kappa_j/4d$. Therefore $\mathbf{Pr}[\text{MERGE}_j] \leq \frac{1}{4d}$, as required to complete the proof of the theorem. ☐

We are ready to conclude Theorem 5.1. Let us define the sketching function HMR-SKETCH$(u; T, \kappa, \delta)$ for targeted mismatch recovery $\mathbf{HMR}(T, \kappa, 4d)$ to be the superposition sketching function on the tree T with the capacity function κ, where the distribution on **leaf→root** functions is as given in the above lemma, and with the redundancy set to $\lceil 8(\sum_{j=1}^{d} \ln|L_j| + \log(1/\delta) + 2) \rceil$. Let us define

HMR-RECOVER to be the associated recovery function. To conclude the correctness of the scheme (the first item of Theorem 5.1) we apply Proposition 5.5 together with Lemma 5.7 with parameters set as follows: $D = \prod_{j=1}^{d} |L_j|$, $S = \{1, \ldots, \kappa_0\}$, redundancy $\ell = \lceil 8(\sum_{j=1}^{d} \ln |L_j| + \log(1/\delta) + 2) \rceil$, and (H, μ) as defined above for (T, κ). (Notice, the theorem hypothesis that $|\Gamma| \geq 4 \prod_{j=1}^{d} |L_j|^2$ implies that $|\Gamma| \geq 4(|D| - 1) \cdot |S|$ as required by Proposition 5.5.) The sketch consists of $O(\ell \cdot |S|)$ elements from Γ so it takes $O(\kappa_0 \cdot \log |\Gamma| \cdot (\log \prod_{i=1}^{d} |L_j| + \log(1/\delta)))$ bits. Evaluating a hash function from H at a single point takes time $O(d \log^{O(1)} |\Gamma|)$ so by Proposition 5.4, the sketching algorithm runs in time $O(\ell \cdot \prod_{i=1}^{d} |L_j| \cdot d \log^{O(1)} |\Gamma|) = O(\prod_{i=1}^{d} |L_j| \cdot \log^{O(1)} |\Gamma| \cdot \log(1/\delta))$. If u is given via its sparse representation then the time to construct the sketch is $O((\kappa_0 + |\mathbf{supp}(u)|) \cdot \log^{O(1)} |\Gamma| \cdot \log(1/\delta))$. The recovery algorithm runs in time $O(\kappa_0 \cdot \log^{O(1)} |\Gamma| \cdot \log(1/\delta))$ as required. Finally, each mismatch pair that is output by the recovery algorithms must appear in more than half of the ℓ redundant sketches. As each sketch outputs at most $|S|$ elements, the number of mismatch pairs output by the algorithm is at most κ_0.

6 Sketch-and-recover scheme for edit distance

In this section we describe our sketch-and-recover scheme for edit distance.

This scheme takes a *distance parameter* k and size parameter n and has the property that given strings x, y of length at most n, with probability at least $2/3$, if $\mathbf{ED}(x, y) \leq k$ the recover algorithm on sketches of x and y will output the set of costly edges in the canonical alignment of x and y, as defined in Section 3.2. If $\mathbf{ED}(x, y) > k$ the recover algorithm should output LARGE. The strings x and y are over the alphabet $\Sigma_n = \{1, \ldots, n^3\}$. We make the simplifying assumption that $k \leq n/4$ otherwise each string can serve as its own sketch.

Let $s = \log n$. Our goal is that the length of the sketch should be "nearly linear" in k, which means $k \times T$ where T is "small". We achieve this with $T = s_{\mathrm{OR}}^2 s^{O(1)}$, where $s_{\mathrm{OR}} = 2^{O(\sqrt{s \log s})}$ is the gap parameter from Theorem 4.2, so this is the dominant term. Our construction is multi-layered and intricate but fairly clean. We are careless with factors of s and of constants in the sketch size; it is clearly possible to reduce some of these factors, but it would complicate our already involved construction.

An important note is that we use the gap threshold algorithm from Theorem 4.2 as a black box. Improving the value of s_{OR} in Theorem 4.2 will translate into a reduction in the sketch size.

The sketch and recover algorithms build recursive structures that are represented as labelled trees $T(W^d)$ or $T(\{0, 1\}^{ds})$. Both of these trees have a number of nodes that is superpolynomial in n, so explicitly evaluating the labeling at all nodes leads to superpolynomial running time. We can avoid this because each of these trees has the property that all but polynomially many nodes are labelled by a *default value*, and the set of nodes not labelled by the default value is a rooted subtree. Thus we only have to explicitly compute the labeling at nodes whose value is not the default value. For example, a key step in building the sketch for x is to build a decomposition for x on the tree $T(W^d)$ where we assign x to the root, and work our way down the tree: if node v has been assigned x_v then the children of v are assigned a (carefully chosen) decomposition of x_v. At most polynomially many nodes will be assigned a nonempty string (because there are at most $|x|$ nonempty substrings at each level of the tree) and the default value is the empty string ε. As we build the tree, when we encounter a node u that is assigned the empty string whose parent is assigned a nonempty string, we note that $x_u = \varepsilon$ and implicitly assign all descendants to ε without doing any computation on the descendants. The set of explicitly assigned nodes is a rooted subtree

25

of polynomial size and any node not appearing in the subtree is implicitly assigned to ε, and our algorithms only pay for the explicitly assigned values.

6.1 Pseudo-code for edit distance sketch-and-recover algorithms

Our sketching algorithms and their analysis depend on various parameters. We review the parameters here. They are implicitly part of the input of all of our procedures.

The following parameters are set by the user of the scheme: k is the edit distance bound that the scheme is designed to handle, and n is the upper bound on the size of the input strings. We assume n is a power of two and that it is larger than $40k s_{\mathrm{OR}}$, where s_{OR} is the distortion of the Ostrovsky-Rabani fingerprint. We use $s = \log n$ as a shorthand.

We recall the following parameters that were introduced in formulating the properties of the Ostrovsky-Rabani fingerprint and the function BASIC-DECOMP. They are used to define other parameters of our algorithm: $N = n^{60}$ is the length of bit-vectors that represent grammars output by BASIC-DECOMP. $s_{\mathrm{E}\to\mathrm{H}} = O(\log^2 n)$ appears in Part 2b of Theorem 4.3 concerning BASIC-DECOMP. It bounds how much is the edit distance of two strings multiplied when we measure the Hamming distance of their representation by grammars. $s_{\mathrm{split}} = O(\log^4 n)$ is the multiplier that appears in the failure probability in Part 2a of Theorem 4.3. The failure probability is the ratio between the edit distance of two strings and the desired grammar sparsity multiplied by s_{split}. $s_{\mathrm{OR}} = 2^{O(\sqrt{\log(n)\log\log(n)})}$ is the gap parameter (*distortion*) from Theorem 4.2 concerning the Ostrovsky-Rabani fingerprints. We can assume that all three constants $s_{\mathrm{E}\to\mathrm{H}}$, s_{split} and s_{OR} are integers.

Our algorithm uses some large field $\mathbb{F} = \mathbb{F}_p$ to represent fingerprints and compute hierarchical mismatch recovery schemes. Here p is a prime whose bit length is at least $4(\log N \cdot \log n)$ which is sufficient for all those applications. We assume that the prime p is selected during the initialization of our algorithm (see Section 3.3).

The sketching algorithm for a string x operates by implicitly building some tree data structures to represent x, which are then compressed. The trees underlying these data structures are fixed. The decomposition tree produced by MAIN-DECOMP is $T(W^d)$, where $W = \{0,1\}^s$ indexes its edges, and $d = \lceil \log(20k \cdot s_{\mathrm{OR}}) \rceil$ is its depth. MAIN-DECOMP also builds some data structures on the binary tree $T(\{0,1\}^{sd})$. The nodes at level j of $T(W^d)$ correspond in the obvious way to the nodes at level sj in $T(\{0,1\}^{sd})$.

The following parameters are chosen carefully for the algorithm and its analysis. They bound quantities at nodes of $T(W^d)$: t_i is the threshold of Ostrovsky-Rabani fingerprint used for substrings at level i of the tree, k_i is the sparsity of grammar bit-vectors at level i of the tree, κ_i is the capacity of nodes at level i of the hierarchical mismatch recovery scheme on $T(W^d)$. The quantities shrink by factor of 2 at each level and the latter two quantities can be defined in terms of t_i. The parameters are chosen to satisfy various constraints that arise in the analysis and we summarize the constraints in Section 9.

Ostrovsky-Rabani fingerprint threshold	Grammar sparsity	Hierarchical mismatch recovery capacity
$t_0 = \lceil 20k \cdot s_{\mathrm{OR}} \rceil_2$	$k_0 = (32 s_{\mathrm{split}} \log^6 n) \cdot t_0$	$\kappa_0 = (512 s_{\mathrm{split}} s_{\mathrm{OR}} \log^{12} n) \cdot t_0$
$t_i = t_{i-1}/2$	$k_i = k_{i-1}/2$	$\kappa_i = \kappa_{i-1}/2$
$\quad = t_0/2^{d-i}$	$\quad = (32 s_{\mathrm{split}} \log^6 n) \cdot t_i$	$\quad = (512 s_{\mathrm{split}} s_{\mathrm{OR}} \log^{12} n) \cdot t_i$

Here by $\lceil a \rceil_2$ we mean the smallest power of two larger than a. We set $s_{\text{load}} = 64 s_{\text{split}} \, s_{\text{OR}} \log^6 n$, which is used in Proposition 8.10. For $j \in \{1, \ldots, d\}$, κ^j is the capacity function on $T(W^j \times \{1, \ldots, N\})$ where for $i \in \{0, \ldots, j+1\}$, $\kappa_i^{\leq j} = \kappa_i$. Additionally we need to define a capacity for a hierarchical mismatch recovery scheme on $T(\{0,1\}^{sd})$. For $j \in \{0, \ldots, ds-1\}$, $\lambda^{\leq j}$ is the capacity function on $T(\{0,1\}^j)$ where for $i \in \{0, \ldots, j\}$, $\lambda_i^{\leq j} = \kappa_{\lfloor i/s \rfloor}$.

Algorithm 2 MAIN-DECOMP(x)

Input: String $x \in \Sigma_{\bar{n}}^{\leq n}$.

Output: Tree decomposition **substrs** of x indexed by $W^{\leq d}$ with associated tree **OR-prints** of Ostrovsky-Rabani fingerprints and grammars **grams**, **vis-nodes** which is the support of **substrs**, refinement **substrs*** of the tree decomposition of x indexed by $\{0,1\}^{\leq ds}$ with the tree of fingerprints **fingerprints*** and left child size tree **left-size***, and **vis-nodes*** which is the support of **substrs***.

---

6 Set **substrs**$_\varepsilon = x$; **vis-nodes** $= \{\varepsilon\}$.
7 **for** $j = 0, \ldots, d-1$ **do**
8   **for** $v \in$ **vis-nodes** $\cap \, W^j$ **do**
9   **if** *substrs$_v \neq \varepsilon$* **then**
10   Call BASIC-DECOMP(**substrs**$_v$; k_{j+1}) to obtain decomposition $(z_i : i \in W)$ of **substrs**$_v$ with associated grammars $(G_i : i \in W)$ where $G_i \in \{0,1\}^N$.
11   **for** *each $i \in W$* **do**
12   **if** $z_i \neq \varepsilon$ **then**
13   **substrs**$_{v \circ i} = z_i$;
14   **grams**$_{v \circ i} = G_i$;
15   **OR-prints**$_{v \circ i} = $ OR-FINGERPRINT$(z_i; t_{j+1})$;
16   **vis-nodes** $= $ **vis-nodes** $\cup \{v \circ i\}$.
17   **end**
18   **end**
19   **end**
20   **end**
21 **end**
22 **vis-nodes**$^* = $ **vis-nodes** $\cap \{0,1\}^{ds}$;
23 **for** *each $v \in$ **vis-nodes**** **do substrs**$_v^* = $ **substrs**$_v$;
24 **for** $j = ds-1, \ldots, 0$ **do**
25   **for** *each $v \in \{0,1\}^j$ where $v \circ 0$ or $v \circ 1 \in$ **vis-nodes**** **do**
26   **substrs**$_v^* = $ **substrs**$_{v \circ 0}^* \circ$ **substrs**$_{v \circ 1}^*$;
27   **fingerprints**$_v^* = $ FINGERPRINT(**substrs**$_v^*$);
28   **left-size**$_v^* = |$**substrs**$_{v \circ 0}^*|$;
29   **vis-nodes**$^* = $ **vis-nodes**$^* \cup \{v\}$.
30   **end**
31 **end**
32 Output **substrs**, **grams**, **OR-prints**, **vis-nodes**, **substrs***, **left-size***, **fingerprints***, **vis-nodes***.

The main component of our sketching algorithm is procedure MAIN-DECOMP. It takes the input

string x and recursively applies procedure BASIC-DECOMP to it with sparsity parameter k_j, where j is the recursion level. Each call to procedure BASIC-DECOMP splits a substring of x into smaller substrings, each represented by a *grammar* of sparsity k_j. We label the tree $T(W^d)$ by the iteratively obtained substrings of x. The set **vis-nodes** traces which nodes of the tree are non-empty.

Thus we build two vertex labelings of the tree $T(W^d)$: **substrs** is the labeling by substrings of x that is a decomposition tree for x, and **grams** labels each node, except the root, by a bit-vector that represents a grammar that encodes **substrs**$_v$. All of the labelings are represented via a sparse representation. The algorithm constructs the set **vis-nodes** of nodes consisting of the root **substrs**$_\varepsilon$ and all nodes v for which **substrs**$_v \neq \varepsilon$. The labels **substrs**$_v$ and **grams**$_v$ are assigned values explicitly only if $v \in$ **vis-nodes**; for $v \notin$ **vis-nodes**, **substrs**$_v$ and **grams**$_v$ are not explicitly assigned, and are implicitly set, respectively, to ε and the 0 vector.

Additionally, for each non-empty substring **substrs**$_v$ we also calculate its Ostrovsky-Rabani fingerprint for a suitable threshold parameter. The fingerprints form another labeling **OR-prints** of $T(W^d)$. The fingerprints are interpreted as elements from $\mathbb{F}$.

In the second part of MAIN-DECOMP we build a refinement **substrs*** of **substrs**. **substrs*** is a vertex labeling of the binary tree $T(\{0,1\}^{ds})$ where **substrs**$_v^*$ is the concatenation of **substrs**$_u$ for $u \in W^d$ that have v as its prefix. (Here we view u as a binary string of length ds.) This can be efficiently built bottom-up starting from leaves. For each **substrs**$_v^*$ we calculate its Karp-Rabin fingerprint **fingerprints**$_v^*$ and we set **left-size**$_v^*$ to be the length of **substrs**$_{v\circ0}^*$, the left child of v. Hence, **fingerprints*** and **left-size*** are both labelings of $T(\{0,1\}^{ds})$. The labeling **substrs** of $T(W^d)$ corresponds to a partial labeling of the binary tree $T(\{0,1\}^{ds})$ where only nodes at levels that are multiples of s are labeled. **substrs*** "interpolates" this partial labeling to a full labeling of the binary tree. As with **substrs**, **substrs**$_v^*$ is only explicitly defined for nodes where it is nonempty. The set **vis-nodes*** traces the non-empty nodes v of **substrs***. For all other v, **substrs**$_v^*$ is implicitly equal to ε and **left-size**$_v^*$ is implicitly set to 0.

Algorithm 3 GRAMMAR-CONDENSE(**grams**, **OR-prints**, **vis-nodes**)

Input: Tree of grammars **grams** indexed by $W^{\leq d}$ with support **vis-nodes**, and tree **OR-prints** of associated Ostrovsky-Rabani fingerprints.
Output: HMR sketch for each level of the **grams**.

---

33 **for** *each $j \in \{1, \ldots, d\}$* **do**
34   Let **grams**^{j} be the all-zero D-sequence over $\mathbb{F}$ where $D = W^j \times \{1, \ldots, N\}$
35   **for** $v \in$ ***vis-nodes*** $\cap\, W^j$ *and* $i \in \{1, \ldots, N\}$ **do**
36   **grams**$_{v\circ i}^j =$ **OR-prints**$_v \times_\mathbb{F}$ **grams**$_{v,i}$.
37   **end**
38   **str-sk**$^j =$ HMR-SKETCH(**grams**^{j}; $T(W^j \times \{1, \ldots, N\}), \kappa^{\leq j}, 4(d+1), 1/n^4$).
39 **end**
40 Output **str-sk** $= ($**str-sk**$^1, \ldots,$ **str-sk**$^d)$.

After computing the decomposition of a string x, the decomposition is *condensed* into hierarchical mismatch recovery sketch in procedures GRAMMAR-CONDENSE and LOCATION-CONDENSE. The former is used to sketch the grammar tree **grams** and the latter is used to sketch the tree of child-sizes **left-size***. The first sketch will allow us to recover grammars of differing substrings between x and y, the second will be used to determine the exact position of the differing substrings in x and

y. The GRAMMAR-CONDENSE takes each of the layers of **grams**, and applies to it HMR-SKETCH with a suitable choice of parameters. Each layer of **grams** is a sequence of grammars represented by a binary vector. Each of the binary vectors is multiplied by a corresponding Ostrovsky-Rabani fingerprint so that if two corresponding nodes in the decomposition of x and y are labelled by substrings of x and y of large edit distance then the multiplied grammar vectors will differ in each non-zero entry. Those non-zero positions will be recovered from the hierarchical sketch if the path to the root in the tree of grammars is not overloaded. The collection of the hierarchical sketches, one for each layer of $T(W^d)$, is the sketch output by GRAMMAR-CONDENSE.

Algorithm 4 LOCATION-CONDENSE(**left-size***, **fingerprints***, **vis-nodes***)

Input: Tree of Karp-Rabin fingerprints **fingerprints*** for the refined tree decomposition **substrs*** of x indexed by $\{0,1\}^{\leq ds}$ with left child size tree **left-size*** of **substrs*** supported on **vis-nodes***.

Output: **HMR** sketch for each level of the **left-size***.

---

41 **for** *each* $j \in \{0, \ldots, ds-1\}$ **do**
42   Let **left-size**^{j} be the all-zero D-sequence over $\mathbb{F}$ where $D = \{0,1\}^j$.
43   **for** $v \in$ **vis-nodes**$^* \cap \{0,1\}^j$ **do**
44   **left-size**$^j_v = n \times_\mathbb{F}$ **fingerprints**$^*_v +_\mathbb{F}$ **left-size**$^*_v$.
45   **end**
46   **loc-sk**^{j} = HMR-SKETCH(**left-size**^{j}; $T(\{0,1\}^j), \lambda^{\leq j}, 4ds, 1/n^4$).
47 **end**
48 Output **loc-sk** = (**loc-sk**$^0, \ldots,$ **loc-sk**$^{ds-1}$).

Similarly, each of the sd layers of **left-size*** consists of integers from $\{1, \ldots, n\}$. Each of the integers is offset by a corresponding Karp-Rabin fingerprint from **fingerprints*** so that if the fingerprints in x and y differ the corresponding values will differ. (Here, we assume that the field $\mathbb{F}$ is of characteristic larger than the maximum value of the fingerprint times n so the mapping is invertible.) Again, hierarchical mismatch recovery sketch of each fingerprinted layer with suitable parameters allows to recover sizes of left substrings for differing pairs of corresponding nodes in **substrs*** for x and y. Summing-up the sizes of the left subtrees along a path from some node in $T(\{0,1\}^{ds})$ to the root allows to recover the position of a differing substring in the decomposition of x and y, respectively. So the collection of the hierarchical sketches, one for each layer of **left-size***, is the sketch output by LOCATION-CONDENSE.

Algorithm 5 MAIN-SKETCH(x)

Input: String $x \in \Sigma_n^{\leq n}$.

Output: A sketch **main-sk** of x.

---

49 Invoke MAIN-DECOMP(x) which outputs **substrs**, **grams**, **OR-prints**, **vis-nodes**, **substrs***, **left-size***, **fingerprints***, **vis-nodes***.
50 **str-sk** = GRAMMAR-CONDENSE(**grams**, **OR-prints**, **vis-nodes**).
51 **loc-sk** = LOCATION-CONDENSE(**left-size***, **fingerprints***, **vis-nodes***).
52 Output **main-sk** = (**str-sk**, **loc-sk**).

Procedure MAIN-SKETCH builds a single instance of a sketch for x using a single call to each of

the procedures MAIN-DECOMP, GRAMMAR-CONDENSE, LOCATION-CONDENSE. ED-SKETCH applies the procedure MAIN-SKETCH $(10s + 50)$-times using independent randomness to get a sequence of sketches for x. Their concatenation is the sketch for x.

Algorithm 6 ED-SKETCH(x, k, n)

Input: String $x \in \Sigma_{\tilde{n}}^{\leq n}$, integer parameters k and n.
Output: A sketch **sk** of x.

---

53 Determine $t_1, \ldots, t_d, \kappa^{\leq 1}, \ldots, \kappa^{\leq d}, \lambda^{\leq 0}, \ldots, \lambda^{\leq ds-1}$.
54 **OR-print** = OR-FINGERPRINT$(x; 20ks_{\mathrm{OR}})$.
55 **for** $i \in \{1, \ldots, 10s + 50\}$ **do** **main-sk**$_i$ = MAIN-SKETCH(x);
56 Output **sk** = (**OR-print**, **main-sk**$_1, \ldots,$ **main-sk**$_{10s+50}$).

6.2 Pseudo-code for the recovery algorithm

The recovery of edit distance from sketches of x and y starts in procedures FIND-STRINGS and FIND-LOCATIONS. FIND-STRINGS takes two sketches **str-sk**(x) and **str-sk**(y) that were produced by GRAMMAR-CONDENSE. **str-sk**(x) consists of d hierarchical mismatch recovery sketches, each of them sketches a level of the grammar tree of x. Similarly for **str-sk**(y).

Algorithm 7 FIND-STRINGS(**str-sk**(x), **str-sk**(y))

Input: HMR-SKETCH sketches **str-sk**(x) and **str-sk**(y) of grammars for tree decompositions of x and y.
Output: Sequences **found-str**$_v(x)$ and **found-str**$_v(y)$ indexed by some **str-nodes** $\subseteq W^{\leq d}$ of substrings on which x and y differ.

---

57 Set **str-nodes** = $\emptyset$.
58 **for** $j \in \{1, \ldots, d\}$ **do**
59 $\quad$ **gram-mismatch**^{j} = HMR-RECOVER(**str-sk**$^j(x)$, **str-sk**$^j(y)$).
60 $\quad$ **str-nodes**^{j} = $\{v \in W^j; \ ((v, i), \alpha, \beta) \in$ **gram-mismatch**^{j} for some $i \in \{1, \ldots, N\}, \alpha, \beta \in \mathbb{F}\}$.
61 $\quad$ **for** $v \in$ **str-nodes**^{j} **do**
62 $\quad\quad$ **found-gram**$_v(x)$ = **found-gram**$_v(y)$ = 0^N.
63 $\quad\quad$ **for** *each* $((v', i), \alpha, \beta) \in$ **gram-mismatch**^{j} *where* $v' = v$ **do**
64 $\quad\quad\quad$ **if** $\alpha \neq 0$ **then found-gram**$_{v,i}(x) = 1$;
65 $\quad\quad\quad$ **if** $\beta \neq 0$ **then found-gram**$_{v,i}(y) = 1$;
66 $\quad\quad$ **end**
67 $\quad\quad$ **found-str**$_v(x)$ = BASIC-DECODE(**found-gram**$_v(x)$).
68 $\quad\quad$ **found-str**$_v(y)$ = BASIC-DECODE(**found-gram**$_v(x)$).
69 $\quad\quad$ **if** *found-str*$_v(x)$ *and* *found-str*$_v(y)$ *are both defined* **then**
70 $\quad\quad\quad$ Add v to **str-nodes**.
71 $\quad\quad$ **end**
72 $\quad$ **end**
73 **end**
74 Output (**found-str**$_v(x) : v \in$ **str-nodes**) and (**found-str**$_v(y) : v \in$ **str-nodes**).

FIND-STRINGS calls the recovery procedure HMR-RECOVER on each corresponding pair of hier-

archical sketches. The call to HMR-RECOVER provides a list **gram-mismatch** of mismatch triples. Each mismatch triple $((v, i), \alpha, \beta)$ asserts that the i-th bit of the grammar at node v of the decomposition tree **grams** for x is 1 if α is non-zero, and the same bit of a grammar for y is 1 if β is non-zero. Grammars for which we get some mismatch triple are decoded by BASIC-DECOMP.

If all non-zero entries of the grammar were recovered then BASIC-DECOMP returns the corresponding string otherwise it returns UNDEFINED. (The latter case occurs if only a subset of ones was recovered for the given grammar vector.) This provides a collection of substrings of x from its **substrs**, and similarly for y. We organize the substrings as a partial labeling of $T(W^d)$, for which **str-nodes** identifies the support of the labeling. The two partial labelings for x and y are the output of the procedure.

Algorithm 8 FIND-LOCATIONS($\mathbf{loc\text{-}sk}(x), \mathbf{loc\text{-}sk}(y), \mathbf{str\text{-}nodes}$)

Input: HMR-SKETCH sketches **loc-sk**(x) and **loc-sk**(y) of substring locations for refined tree decompositions of x and y, **str-nodes** are nodes of the decomposition for which we want to recover their starting position.

Output: Sequences **found-start**$_v(x)$ and **found-start**$_v(y)$ indexed by some **loc-nodes** $\subseteq$ **str-nodes** of starting positions of substrings on which x and y differ.

---

75 **for** $j \in \{0, \ldots, ds-1\}$ **do**
76   **size-mismatch**^{j} = HMR-RECOVER(**loc-sk**$^j(x)$, **loc-sk**$^j(y)$).
77   **for** $(v, \alpha, \beta) \in$ ***size-mismatch**^{j}* **do**
78   **found-size**$_v(x) = \alpha \mod n$ (over $\mathbb{N}$)
79   **found-size**$_v(y) = \beta \mod n$ (over $\mathbb{N}$)
80   **end**
81 **end**
82 **loc-nodes** $= \emptyset$.
83 **for** $v \in$ ***str-nodes*** **do**
84   **found-start**$_v(x) = \sum_{u \text{ a left ancestor of } v}$ **found-size**$_u(x)$.
85   **found-start**$_v(y) = \sum_{u \text{ a left ancestor of } v}$ **found-size**$_u(y)$.
86   COMMENT: $u \in \{0,1\}^{<|v|}$ is a left ancestor of v if u is a prefix of v and $v_{|u|+1} = 1$.
87   **if** ***found-start**$_v(x)$ and **found-start**$_v(y)$ are both defined* **then**
88   Add v to **loc-nodes**.
89   **end**
90 **end**
91 Output (**found-start**$_v(x) : v \in$ **loc-nodes**) and (**found-start**$_v(y) : v \in$ **loc-nodes**).

Similarly, FIND-LOCATIONS gets two sketches **loc-sk**(x) and **loc-sk**(y) that were produced by LOCATION-CONDENSE, and a set of target nodes **str-nodes** from the tree decomposition of x and y. **loc-sk**(x) consists of $sd - 1$ hierarchical mismatch recovery sketches, each of them sketches a level of the **left-size*** tree of x. Similarly for y. On each pair of sketches for x and y we apply HMR-RECOVER to identify *differences* between the values at a given level of **left-size*** tree for x and y. (A difference at node v might come from two sources: either the actual values **left-size**$^*_v$ for x and y differ or the associated **substrs**$^*_v$ differ.) For each node v where the recovery procedure identifies a *difference* we recover the value **left-size**$^*_v$ for x and y, respectively, and store it in **found-size**$_v$. After we find all recoverable *differences* between **left-size*** for x and y, for each node $v \in$ **str-nodes**

where we identified a *difference*, we attempt to calculate the starting position in x and y, resp., of the substring $\mathbf{substrs}_v$ of x and y. The starting position of $\mathbf{substrs}_v$ is given by the sum of $\mathbf{left\text{-}size}^*_u$ over all nodes u in $T(\{0,1\}^{ds})$ which we reach from right on the path from v to root. (If any of the $\mathbf{left\text{-}size}^*_u$ does not have its value defined then the starting position for v will remain undefined.) We return two sequences $\mathbf{found\text{-}start}_v(x)$ and $\mathbf{found\text{-}start}_v(y)$ of starting positions we calculated.

Procedures FIND-STRINGS and FIND-LOCATIONS are called from MAIN-RECONSTRUCT which attempts to reconstruct edit distance information from a pair of sketches $\mathbf{main\text{-}sk}(x)$ and $\mathbf{main\text{-}sk}(y)$ obtained from a single run of MAIN-SKETCH. $\mathbf{main\text{-}sk}(x)$ consists of $\mathbf{str\text{-}sk}(x)$ and $\mathbf{loc\text{-}sk}(x)$, and similarly for $\mathbf{main\text{-}sk}(y)$. Sketches $\mathbf{str\text{-}sk}(x)$ and $\mathbf{str\text{-}sk}(y)$ are sent to FIND-STRINGS to recover differing pairs in $\mathbf{substrs}$ of x and y: collections $\mathbf{found\text{-}str}(x)$ and $\mathbf{found\text{-}str}(y)$ indexed by some subset $\mathbf{str\text{-}nodes}$ of the tree nodes.

Then FIND-LOCATIONS is applied on $\mathbf{loc\text{-}sk}(x)$, $\mathbf{loc\text{-}sk}(y)$ and $\mathbf{str\text{-}nodes}$ to obtain starting locations $\mathbf{found\text{-}start}(x)$ of substrings $\mathbf{found\text{-}str}(x)$ within x and similarly starting positions $\mathbf{found\text{-}start}(y)$ of substrings of y. The set of substrings for which the starting position was recovered is identified by $\mathbf{loc\text{-}nodes}$. We let $\mathbf{top\text{-}nodes}$ be the set of nodes from $\mathbf{loc\text{-}nodes}$ that do not have any other node from $\mathbf{loc\text{-}nodes}$ on their path to the root in $T(W^d)$. For each node $v \in \mathbf{top\text{-}nodes}$ we have recovered substrings $\mathbf{found\text{-}str}_v(x)$ and $\mathbf{found\text{-}str}_v(y)$ and their respective positions $\mathbf{found\text{-}start}_v(x)$ and $\mathbf{found\text{-}start}_v(y)$ in x and y. We evaluate the edit distance of the two substrings positioned at those locations and we calculate the costly edges of their canonical alignment. Union of all the edges over $v \in \mathbf{top\text{-}nodes}$ is the output of MAIN-RECONSTRUCT.

Algorithm 9 MAIN-RECONSTRUCT($\mathbf{main\text{-}sk}(x), \mathbf{main\text{-}sk}(y)$)

Input: A pair $\mathbf{main\text{-}sk}(x)$, $\mathbf{main\text{-}sk}(y)$ from a single run of MAIN-SKETCH, where $\mathbf{main\text{-}sk}(x)$ consists of $\mathbf{str\text{-}sk}(x)$ and $\mathbf{loc\text{-}sk}(x)$.

Output: Set of candidate costly edges for some pairs of substrings where x and y differ.

---

92 Call FIND-STRINGS($\mathbf{str\text{-}sk}(x), \mathbf{str\text{-}sk}(y)$) to get ($\mathbf{found\text{-}str}_v(x)$: $v \in \mathbf{str\text{-}nodes}$) and ($\mathbf{found\text{-}str}_v(y)$: $v \in \mathbf{str\text{-}nodes}$) for some $\mathbf{str\text{-}nodes} \subseteq W^{\leq d}$.

93 Call FIND-LOCATIONS($\mathbf{loc\text{-}sk}(x), \mathbf{loc\text{-}sk}(y), \mathbf{str\text{-}nodes}$) to get ($\mathbf{found\text{-}start}_v(x)$: $v \in \mathbf{loc\text{-}nodes}$) and ($\mathbf{found\text{-}start}_v(y)$: $v \in \mathbf{loc\text{-}nodes}$) for some $\mathbf{loc\text{-}nodes} \subseteq \mathbf{str\text{-}nodes}$.

94 Let $\mathbf{top\text{-}nodes} = \{v \in \mathbf{loc\text{-}nodes};\ v$ has no proper prefix in $\mathbf{loc\text{-}nodes}\}$.

95 **for** $v \in \boldsymbol{top\text{-}nodes}$ **do**

96   Calculate the canonical alignment $\mathbf{canon}^+(\mathbf{found\text{-}str}_v(x), \mathbf{found\text{-}str}_v(y))$ of the string $\mathbf{found\text{-}str}_v(x)$ starting at location $\mathbf{found\text{-}start}_v(x)$ and the string $\mathbf{found\text{-}str}_v(y)$ starting at location $\mathbf{found\text{-}start}_v(y)$.

97   Let $\mathbf{candids}_v = \mathbf{costly}(\mathbf{canon}^+(\mathbf{found\text{-}str}_v(x), \mathbf{found\text{-}str}_v(y))$.

98 **end**

99 $\mathbf{candids} \leftarrow \bigcup_{w \in \mathbf{top\text{-}nodes}} \mathbf{candids}_v$.

100 Output $\mathbf{candids}$.

The main recovery function is given by ED-RECOVER. It receives two sketches for x and y. The sketch of x contains Ostrovsky-Rabani fingerprint $\mathbf{OR\text{-}print}(x)$ of the whole x and a sequence of sketches $\mathbf{main\text{-}sk}_1(x), \ldots, \mathbf{main\text{-}sk}_{10s+50}(x)$. Similarly for the sketch of y. If $\mathbf{OR\text{-}print}(x)$ differs from $\mathbf{OR\text{-}print}(y)$ then x and y are of edit distance more than k. Otherwise we invoke procedure MAIN-RECONSTRUCT for each $\mathbf{main\text{-}sk}_i(x)$ and $\mathbf{main\text{-}sk}_i(y)$, $i = 1, \ldots, 10s+50$. Each call to

MAIN-RECONSTRUCT produces a collection of costly edges that should be part of the canonical alignment of x and y. Some of the edges might be misidentified by a given call to MAIN-RECONSTRUCT so as the final output of ED-RECOVER we output only edges that MAIN-RECONSTRUCT outputs more than half of the time.

Algorithm 10 ED-RECOVER($\mathbf{sk}(x), \mathbf{sk}(y); k, n$)

Input: Sketch of x: $\mathbf{sk}(x) = (\mathbf{OR\text{-}print}(x), \mathbf{main\text{-}sk}_1(x), \ldots, \mathbf{main\text{-}sk}_{10s+50}(x))$ and sketch of
$\quad\quad\quad$ y: $\mathbf{sk}(y) = (\mathbf{OR\text{-}print}(y), \mathbf{main\text{-}sk}_1(y), \ldots, \mathbf{main\text{-}sk}_{10s+50}(y))$.
Output: Edit distance of x and y if it is smaller than k, and LARGE otherwise.

---

101 **if** $\mathbf{OR\text{-}print}(x) \neq \mathbf{OR\text{-}print}(y)$ **then** return LARGE;
102 **for** $i \in \{1, \ldots, 10s + 50\}$ **do**
103 $\quad$ $\mathbf{candids}_i = $ MAIN-RECONSTRUCT($\mathbf{main\text{-}sk}_i(x), \mathbf{main\text{-}sk}_i(y)$)
104 **end**
105 Let **found-edges** be the set of edges that appear in more than half of the sets $\mathbf{candids}_i$.
106 **if** $|\mathbf{found\text{-}edges}| > k$ **then** return LARGE;
107 Return **found-edges**.

---

7 Analysis of the sketch-and-recover scheme

7.1 Main result

An execution of ED-SKETCH and ED-RECOVER on a pair of strings x, y consists of evaluation of ED-SKETCH(x) and ED-SKETCH(y) and of ED-RECOVER(ED-SKETCH(x), ED-SKETCH(y)) where all three evaluations use the same randomizing parameter ρ. The execution of ED-SKETCH(x), ED-SKETCH(y) and ED-RECOVER(ED-SKETCH(x), ED-SKETCH(y)) *succeeds* provided that

- If $\mathbf{ED}(x, y) > k$ then ED-RECOVER outputs LARGE, and

- If $\mathbf{ED}(x, y) \leq k$ then ED-RECOVER outputs $\mathbf{costly}(\mathbf{canon}^+(x, y))$,

and fails otherwise.

In this section we prove the main result of the paper:

Theorem 7.1. *The pair of algorithms* ED-SKETCH *and* ED-RECOVER *provides a sketch-and-recover scheme for edit distance. Given sufficiently large size parameter n, $s = \log n$, distance parameter $k \leq n/40s_{\mathrm{OR}}^2$ and alphabet Σ_n:*

1. ED-SKETCH *runs on strings x of length at most n and outputs a sketch $\mathbf{str\text{-}sk}(x)$ of length at most $O(ks_{\mathrm{OR}}^2 s^{O(1)})$.*

2. *The running time of* ED-SKETCH *is $\widetilde{O}(|x| \cdot T(n))$ and the running time of* ED-RECOVER *is $\widetilde{O}(\min(n^2, k^3 s_{\mathrm{OR}}^2))$, where $T(n) \geq n$ is the time to compute* OR-FINGERPRINT *on inputs of length n. (Recall, $T(n)$ is polynomial in n.)*

3. *For all x, y of length at most n, the execution of* ED-SKETCH(x), ED-SKETCH(y), *and* ED-RECOVER(ED-SKETCH(x), ED-SKETCH(y)) *fails with probability at most $1/3$.*

As usual, the value $1/3$ for failure probability is arbitrary, and can be replaced by any function $\delta > 0$ by repetition: construct a sketch consisting of $r = O(\log(1/\delta))$ independent sketches of x and of y, apply the recovery algorithm to each pair of corresponding sketches, and take majority vote.

The bounds on the sketch size and running time are straightforward and given in Section 7.2.

The most significant part of the analysis is the upper bound on the failure probability. This proof is contained in Sections 7.4 to 8.2.

7.2   Time and space analysis

In this section we prove parts 1 and 2 of Theorem 7.1. For part 1 we bound the size of the output of ED-SKETCH. The output consists of $10s + 50$ independent instances of the output of GRAMMAR-CONDENSE and LOCATION-CONDENSE. Each call to GRAMMAR-CONDENSE produces d sketches using HMR-SKETCH, each of bit-size $\widetilde{O}(s_{\mathrm{OR}}^2 k)$. Each call to LOCATION-CONDENSE produces $ds - 1$ sketches using HMR-SKETCH, each of bit-size $\widetilde{O}(s_{\mathrm{OR}}^2 k)$. Hence, the total size of the sketch is $\widetilde{O}(k 2^{O(\sqrt{\log(n)\log\log(n)})})$.

For part 2 we bound the time of ED-SKETCH and ED-RECOVER. Let $m = |x|$ and $T(n)$ be the time to compute OR-FINGERPRINT on strings of length at most n. ED-SKETCH makes one call to OR-FINGERPRINT which takes time $T(n)$ and then makes $10s + 50 = \widetilde{O}(1)$ independent calls to MAIN-SKETCH. MAIN-SKETCH calls MAIN-DECOMP, GRAMMAR-CONDENSE, and LOCATION-CONDENSE.

MAIN-DECOMP constructs **substrs**, **grams**, **OR-prints**, **substrs***, **left-size***, and **fingerprints***. Even though each of these is defined on trees which have (roughly) n^d nodes, which is superpolynomial in n, the time to compute them will be polynomial in m as we use sparse representation for each labeling of the tree. Sets **vis-nodes** and **vis-nodes*** determine the nodes with non-empty labels in those trees.

Since **substrs** is a decomposition tree for x, each layer is a decomposition of x and has at most m nonempty strings, so $|\textbf{vis-nodes}| \le dm$. Thus, the total cost of calls to BASIC-DECOMP for strings on a single layer of **substrs** is $\widetilde{O}(m)$. Hence, each layer of **grams** contains in total $\widetilde{O}(m)$ ones in the grammar bit-vectors. To construct **OR-prints**, we apply OR-FINGERPRINT to each nonempty **substrs**$_v$. This takes total time $\widetilde{O}(m \cdot T(n))$.

The decomposition tree **substrs*** also uses a sparse representation and is explicitly defined on the set **vis-nodes*** which consists of all nodes in the binary tree that are ancestors of a node in **vis-nodes**. Since each node in **vis-nodes** has at most ds ancestors $|\textbf{vis-nodes}^*| = \widetilde{O}(m)$. Hence, **left-size**$_v^*$ can be computed in time $\widetilde{O}(m)$. We apply FINGERPRINT to each node of **substrs*** in **vis-nodes***. We need $\widetilde{O}(m)$ time to compute the fingerprints for nodes on a single layer of **substrs***. Thus the total time to compute **substrs***, **left-size***, and **fingerprints*** is $\widetilde{O}(m)$.

GRAMMAR-CONDENSE involves the computation of d instances of HMR-SKETCH. The j-th instance is for **grams**^{j} which is the j-th layer of **grams**. Thus the sparse representation of **grams**^{j} contains at most $\widetilde{O}(m)$ entries so the time to compute a single HMR-SKETCH is $\widetilde{O}(m + s_{\mathrm{OR}}^2 k)$.

Similarly LOCATION-CONDENSE involves the computation of $ds - 1$ instances of HMR-SKETCH. The j-th instance is for the function **left-size**^{j} which has at most m non-zero entries so again the time of HMR-SKETCH is $\widetilde{O}(m + s_{\mathrm{OR}}^2 k)$.

Thus the overall time of ED-SKETCH is $\widetilde{O}(m \cdot T(n) + m + s_{\mathrm{OR}}^2 k)$.

Next we consider the time of ED-RECOVER. We will argue first that its running time is at most $\widetilde{O}(n^2)$. Then we point out how to make it faster. FIND-STRINGS is applied to each pair

str-sk$^j(x)$, **str-sk**$^j(y)$ for $j \in \{1, \ldots, d\}$ and FIND-LOCATIONS is applied to each pair **loc-sk**$^j(x)$, **loc-sk**$^j(y)$ for $j \in \{0, \ldots, ds-1\}$.

Each call to FIND-STRINGS runs HMR-RECOVER whose running time is $\widetilde{O}(s_{\mathrm{OR}}^2 k)$. This produces a set of at most $\widetilde{O}(s_{\mathrm{OR}}^2 k)$ mismatch pairs. This determines the set **str-nodes** of nodes, also of size at most the number of mismatch pairs. For each node $v \in$ **str-nodes** we use the mismatch pairs to construct **found-gram**^j_v, which is a proposed reconstruction of **grams**$_v$. We apply BASIC-DECODE to each of these to get **found-str**$_v$. The total time is within $\widetilde{O}(n)$.

Similarly, each call to FIND-LOCATIONS runs HMR-RECOVER. This produces a set of at most $\widetilde{O}(s_{\mathrm{OR}}^2 k)$ mismatch pairs. For each node in **str-nodes** we calculate a sum of at most sd elements. Hence, the total time is $\widetilde{O}(n)$.

Last, for each node v in **str-nodes** we have to compute the edit distance of **found-str**$_v(x)$ and **found-str**$_v(y)$ which takes time $O(|$**found-str**$_v(x)| \cdot |$**found-str**$_v(y)|)$. By Corollary 7.8 (which is stated later) under *normal execution*, both **found-str**$_v(x)$ and **found-str**$_v(y)$ are real substrings of x and y, respectively. Thus the sum of lengths of **found-str**$_v(x)$ at the same layer of the decomposition tree is at most n. Thus the total time needed for the reconstruction is $\widetilde{O}(n^2)$ under *normal execution*. If the recovery algorithm runs for more time than $\widetilde{O}(n^2)$ we can terminate it as the execution is *abnormal* and the result might be incorrect. That happens with only small probability as shown later.

To obtain a faster running time for the recovery, observe that one does not have to expand the grammars to compute their canonical alignment. Using the result of [GKLS22] one can compute the set of costly edges of the canonical alignment for two strings represented by grammars of size at most g in time $\widetilde{O}(g + k'^2)$ where k' is the edit distance of the two strings. The total size of grammars we will recover is $\widetilde{O}(ks_{\mathrm{OR}}^2)$, their number is also at most $\widetilde{O}(ks_{\mathrm{OR}}^2)$, and the edit distance we care for is k. Hence running the edit distance algorithm on all recovered pairs of grammars costs at most $\widetilde{O}(k^3 s_{\mathrm{OR}}^2)$.

7.3 Upper bound on the failure probability

Having proved parts 1 and 2 of Theorem 7.1, we now turn to the proof of Part 3. In this section we state Theorem 7.2 and use it to prove Part 3 of Theorem 7.1. The proof of Theorem 7.2 will be divided into several subsections.

We fix strings x and y and consider an execution of ED-SKETCH(x), ED-SKETCH(y) and ED-RECOVER(ED-SKETCH(x), ED-SKETCH(y)). We want to show that the probability of failure is at most $1/3$.

Recall that the algorithm ED-RECOVER first compares **OR-print**(x) and **OR-print**(y) and outputs LARGE if they are unequal. If they are equal then it executes ED-RECOVER to produce the set **found-edges** of annotated edges and outputs LARGE if $|$**found-edges**$| > k$ and outputs **found-edges** if $|$**found-edges**$| \leq k$.

It simplifies the discussion to modify ED-RECOVER so that it starts by computing **found-edges** and only then compares **OR-print**(x) to **OR-print**(y) and outputs LARGE if the fingerprints differ and outputs **found-edges** otherwise. This modified algorithm clearly has the same output behavior as ED-RECOVER but has the analytic advantage that the (random) value of **found-edges** is independent of the event **OR-print**$(x) = $ **OR-print**(y).

We will prove:

Theorem 7.2. *If* $\boldsymbol{ED}(x, y) < 20ks_{\mathrm{OR}}$ *then* $\boldsymbol{Pr}[\boldsymbol{found\text{-}edges} \neq \boldsymbol{costly}(\boldsymbol{canon}^+(x, y))] \leq 1/4$.

Using this theorem, it is easy to bound the failure probability of ED-RECOVER to complete the proof of the third part of Theorem 7.1.

We divide into three cases depending on $\mathbf{ED}(x, y)$.

Case (1): $\mathbf{ED}(x, y) \geq 20ks_{\mathrm{OR}}$. The algorithm correctly outputs LARGE unless OR-FINGERPRINT$(x) =$ OR-FINGERPRINT(y) and by Theorem 4.2, OR-FINGERPRINT$(x) =$ OR-FINGERPRINT(y) happens with probability at most $1/n^4$.

Case (2): $20ks_{\mathrm{OR}} > \mathbf{ED}(x, y) > k$. The algorithm correctly outputs LARGE if OR-FINGERPRINT$(x) \neq$ OR-FINGERPRINT(y) or if **found-edges** $= \mathbf{costly}(\mathbf{canon}^+(x, y))$ (since $|\mathbf{costly}(\mathbf{canon}^+(x, y))| = \mathbf{ED}(x, y)$). So the output will be incorrect only if **found-edges** $\neq \mathbf{costly}(\mathbf{canon}^+(x, y))$. By Theorem 7.2 this happens with probability at most $1/4$.

Case (3): $k \geq \mathbf{ED}(x, y)$. Success requires the algorithm to output $\mathbf{costly}(\mathbf{canon}^+(x, y))$. This will fail only if OR-FINGERPRINT$(x) \neq$ OR-FINGERPRINT(y) or **found-edges** $\neq \mathbf{costly}(\mathbf{canon}^+(x, y))$. By Theorem 4.2, $\mathbf{Pr}[\text{OR-FINGERPRINT}(x) \neq \text{OR-FINGERPRINT}(y)] \leq \frac{ks_{\mathrm{OR}}}{20ks_{\mathrm{OR}}} = \frac{1}{20}$ and by Theorem 7.2, $\mathbf{Pr}[\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y))] \leq \frac{1}{4}$ so the overall error probability is less than $1/3$..

It remains to prove Theorem 7.2 which is done in the sections that follow. Since the hypothesis of Theorem 7.2 is that $\mathbf{ED}(x, y) < 20ks_{\mathrm{OR}}$ this assumption is made throughout.

7.4 Normal executions

In this section we identify certain events in the execution of ED-SKETCH and ED-RECOVER as *abnormal* and prove that the probability of an abnormal execution tends to 0 as n gets large.

On input x, ED-SKETCH constructs a sequence of sketches $\mathbf{main\text{-}sk}_i(x)$ for $i \in \{1, \ldots, 10s+50\}$.

For each corresponding pair $\mathbf{main\text{-}sk}_i(x)$ and $\mathbf{main\text{-}sk}_i(y)$, the recovery algorithm applies MAIN-RECONSTRUCT$(\mathbf{main\text{-}sk}_i(x), \mathbf{main\text{-}sk}_i(y))$. If the output is defined then it is a set $\mathbf{candids}_i$ of annotated edges. The set $\mathbf{candids}_i$ is supposed to "approximate" $\mathbf{costly}(\mathbf{canon}^+(x, y))$ in a suitable sense. The final output $\mathbf{found\text{-}edges}$ of ED-RECOVER is the set of annotated edges that appear in more than half the sets of $\mathbf{candids}_i$.

Each run of MAIN-SKETCH(x) constructs a labeling $\mathbf{substrs}(x)$ of the tree $T(W^d)$ by substrings and another labeling $\mathbf{grams}(x)$ of the same tree where for each $v \neq \varepsilon$, $\mathbf{grams}_v$ is a bit-vector of length N that encodes a grammar for $\mathbf{grams}(x)$. By the properties of BASIC-DECOMP these labelings satisfy:

D$_1$(x): $\mathbf{substrs}(x)$ is a decomposition tree for x

D$_2$(x): For each node v at a level $j \geq 1$ for which $\mathbf{substrs}_v(x) \neq \varepsilon$, $\mathbf{grams}_v(x)$ is an encoding of $\mathbf{substrs}_v(x)$, i.e. $\mathbf{substrs}_v(x) =$ BASIC-DECODE$(\mathbf{grams}_v(x))$, and the number of 1's in $\mathbf{grams}_v(x)$ is at most k_j

We now identify four *abnormal events*. We will prove that it is very unlikely that any of these conditions occur.

Fingerprinting abnormality. There is a node v, such that $\mathbf{substrs}_v^*(x) \neq \mathbf{substrs}_v^*(y)$ and $\mathbf{fingerprints}_v^*(x) = \mathbf{fingerprints}_v^*(y)$.

Threshold detection abnormality. There is a node v at a level j, such that $\mathbf{ED}(\mathbf{substrs}_v(x), \mathbf{substrs}_v(y)) \geq t_j$ and $\mathbf{OR\text{-}prints}_v(x) = \mathbf{OR\text{-}prints}_v(y)$.

36

HMR abnormality. One of the calls to HMR-RECOVER within the recovery algorithm fails to satisfy the Completeness and Soundness conditions (see Theorem 5.1).

An execution of ED-SKETCH(x), ED-SKETCH(y), and ED-RECOVER(ED-SKETCH(x), ED-SKETCH(y)) is *abnormal* if any of these abnormalities ever occurs and is *normal* if no such abnormality occurs. We define **Normal** to denote the event that the execution is normal, and **Abnormal** to denote the event that the execution is abnormal.

Lemma 7.3. *For any x, y, $Pr[$**Abnormal**$]$ tends to 0 as n gets large.*

Proof. First consider decomposition abnormality for x. The decomposition tree is built in MAIN-DECOMP, which grows a decomposition starting at the root by calling BASIC-DECOMP on each node for which **substrs**$_v$ is nonempty. $\mathbf{D}_1$ or $\mathbf{D}_2$ fail only if some of the calls to BASIC-DECOMP returns UNDEFINED. By Theorem 4.3 the probability that a particular call to BASIC-DECOMP(z) returns UNDEFINED is at most $1/n^4$. In a given call to MAIN-DECOMP(x), the number of nonempty strings at each level of the decomposition is at most $|x| \le n$ so the number of calls made to BASIC-DECOMP(z) for nonempty z is at most dn. So the probability that there is a call to BASIC-DECOMP that returns UNDEFINED is at most d/n^3. If every call to BASIC-DECOMP is defined then a simple induction using the properties of BASIC-DECOMP ensures that the resulting tree satisfies $\mathbf{D}_1$ and $\mathbf{D}_2$. Taking a union bound over the $10s + 50$ calls to MAIN-DECOMP still gives a probability of abnormality that is $\le 1/n$ for n large enough. Similarly for y.

The probability that a fingerprinting abnormality occurs at a particular node $v \in$ **vis-nodes*** is at most $1/n^4$ by Theorem 4.1. Similarly, the probability that a threshold detection abnormality occurs at $v \in$ **vis-nodes** is at most $1/n^4$ by Theorem 4.2. Taking a union bound over the at most sdn nodes in **vis-nodes** and **vis-nodes***, and the $10s + 50$ calls to MAIN-SKETCH yields a probability of either of these abnormalities that is at most $1/n$ for n large enough.

An HMR abnormality occurs if one of the calls to HMR-RECOVER fails. (A failure of HMR-RECOVER is determined by the randomness used to build the sketches of x and y). We set the failure parameter of the hierarchical mismatch recovery scheme to be $1/n^4$ and the number of its instances is $(sd)^{O(1)}$ so the overall probability of failure tends to zero as n grows. $\square$

7.5 Reduction to the main lemma

In this section we introduce the Main Lemma (Lemma 7.5) and use it to prove Theorem 7.2.

We need to define some subsets of the edge set of $\mathbf{Grid}(x, y)$.

The *i-th vertical edge slice $S_V(i)$* of the grid graph $\mathbf{Grid}(x, y)$ consists of all edges that join a vertex with first coordinate $i - 1$ to a vertex with first coordinate i. The *j-th horizontal edge slice $S_H(j)$* consists of all edges that join a vertex with second coordinate $j - 1$ to a vertex with second coordinate j.

Note that a vertical edge slice consists of horizontal and diagonal edges but not vertical edges, and a horizontal edge slice consists of vertical and diagonal edges but not horizontal edges. The union of all of the slices is the entire edge set of $\mathbf{Grid}(x, y)$. We leave the proof of the following observation to the reader.

Proposition 7.4. *Any spanning path of $\mathbf{Grid}(x, y)$ contains exactly one edge from each horizontal slice and from each vertical slice. In particular, $\mathbf{costly}(\mathbf{canon}^+(x, y))$ contains at most one edge from any slice.*

The sketch-and-recover scheme ED-SKETCH+ED-RECOVER operates by performing $10s+50$ independent executions of MAIN-SKETCH+MAIN-RECONSTRUCT and taking majority vote. The Main Lemma below states the key property of a single instance of MAIN-SKETCH + MAIN-RECONSTRUCT, and Theorem 7.2 follows easily from it.

Lemma 7.5. *(**Main Lemma**) Let x, y be strings such that $\mathbf{ED}(x, y) < 20ks_{\mathrm{OR}}$. Consider a single run of* MAIN-SKETCH *on x and y producing $\mathbf{main\text{-}sk}(x)$ and $\mathbf{main\text{-}sk}(y)$, followed by a run of* MAIN-RECONSTRUCT *on $\mathbf{main\text{-}sk}(x), \mathbf{main\text{-}sk}(y)$, which produces the set $\mathbf{candids}$ of annotated edges. For any (horizontal or vertical) edge slice S*

$$Pr[(\mathbf{candids} \cap S \neq \mathbf{costly}(\mathbf{canon}^+(x, y)) \cap S) \wedge \mathbf{Normal}]$$

tends to 0 as n gets large.

Proof of Theorem 7.2 assuming Lemma 7.5. Assume $\mathbf{ED}(x, y) < 20ks_{\mathrm{OR}}$. We must establish an upper bound on $\mathbf{Pr}[\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y))]$. We have:

$$\mathbf{Pr}[\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y))]$$
$$\leq \mathbf{Pr}[\mathbf{Abnormal}] + \mathbf{Pr}[\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y)) \wedge \mathbf{Normal}].$$

Since the first term tends to 0 as n gets large it suffices to prove:

$$\mathbf{Pr}[\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y)) \wedge \mathbf{Normal}] \leq \frac{1}{5}, \tag{1}$$

The hypothesis of the theorem assumes n is sufficiently large. We choose n large enough so that in Lemma 7.5 for any edge slice S the probability that $\mathbf{candids}(x, y) \cap S \neq \mathbf{costly}(\mathbf{canon}(x, y)) \cap S$ is at most $1/10$.

We note that every edge of $\mathbf{Grid}(x, y)$ lies in some (horizontal or vertical) edge slice. It follows that $\mathbf{found\text{-}edges} = \mathbf{costly}(\mathbf{canon}^+(x, y))$ if and only if for all edge slices S we have $\mathbf{found\text{-}edges} \cap S = \mathbf{costly}(\mathbf{canon}^+(x, y)) \cap S$. Define the bad event $B(S)$ to be the event that for more than half of the sets $(\mathbf{candids}_i : i \in \{1, \ldots, 10s + 50\})$, $\mathbf{candids}_i \cap S \neq \mathbf{costly}(\mathbf{canon}^+(x, y)) \cap S$. By the definition of $\mathbf{found\text{-}edges}$, if $\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y))$ then there is at least one slice S for which $B(S)$ happens and so:

$$\mathbf{Pr}[\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y)) \wedge \mathbf{Normal}] \leq \sum_S \mathbf{Pr}[B(S) \wedge \mathbf{Normal}].$$

Consider a particular slice S. By Lemma 7.5 and the above choice of n, for each i, $\mathbf{Pr}[(\mathbf{candids}_i \cap S \neq \mathbf{costly}(\mathbf{canon}^+(x, y)) \cap S) \wedge \mathbf{Normal}] \leq 1/10$. Therefore the Chernoff-Hoeffding bound (Lemma 3.7), implies that $\mathbf{Pr}[B(S) \wedge \mathbf{Normal}] \leq 2e^{-0.32(10s+50)}$ which is easily less than $\frac{1}{10}2^{-s} \leq \frac{1}{10n}$. Since the number of horizontal and vertical slices are each at most n:

$$\mathbf{Pr}[\mathbf{found\text{-}edges} \neq \mathbf{costly}(\mathbf{canon}^+(x, y)) \wedge \mathbf{Normal}] \leq 2n \max_S \mathbf{Pr}[B(S) \wedge \mathbf{Normal}]$$
$$\leq \frac{1}{5},$$

as required to complete the proof of Theorem 7.2. $\square$

It remains to prove Lemma 7.5.

7.6 Preliminaries for the proof of Lemma 7.5

Lemma 7.5 concerns a single execution of MAIN-SKETCH applied to x and y producing outputs **main-sk**(x) and **main-sk**(y) and a single instance of MAIN-RECONSTRUCT applied to **main-sk**(x) and **main-sk**(y).

MAIN-DECOMP(x) generates three (partial) labelings **substrs**, **grams**, and **OR-prints** of the nodes of the tree $T(W^d)$. **substrs**(x) is a string decomposition tree of x. The set **vis-nodes**(x) is the set of nodes v such that **substrs**$_v(x) \neq \varepsilon$. For all $v \in$ **vis-nodes**, **grams**$_v(x)$ is a bit-vector of length N such that
BASIC-DECODE(**grams**$_v(x)$) = **substrs**$_v(x)$; **OR-prints**$_v(x)$ is the Ostrovsky-Rabani fingerprint of **substrs**$_v(x)$.

MAIN-DECOMP(x) also generates three partial labelings **substrs***, **left-size***, and **fingerprints*** of the binary tree $T(\{0,1\}^{ds})$. **substrs**$^*(x)$ is a refinement of **substrs**(x) to the tree $T(\{0,1\}^{ds})$. **substrs**$^*_v(x)$ is explicitly defined on the set **vis-nodes***, consisting of those binary strings that are prefixes (ancestors) of nodes in **vis-nodes**. For $v \in$ **vis-nodes***, **left-size**$^*(x)$ is the size of the string assigned by **substrs*** to the left-child $v \circ 0$ of v, and **fingerprints**$^*_v(x)$ is the Karp-Rabin fingerprint of **substrs**$^*_v(x)$.

The sketch **main-sk**(x) consists of two separate sketches: **str-sk**(x) and **loc-sk**(x). **str-sk**(x) is a sequence of d HMR sketches **str-sk**$^j(x)$ produced by GRAMMAR-CONDENSE, one for each layer $j = 1, \ldots, d$ of grammars in **grams**. **loc-sk**(x) is a sequence of ds sketches **loc-sk**$^j(x)$ produced by LOCATION-CONDENSE for each layer $j = 0, \ldots, ds - 1$ of **left-size**$^*(x)$.

We make the following definitions associated to **substrs**(x) and **substrs**(y): For each node v:

- $\mathbf{I}_v(x)$ denotes the interval location $\mathbf{Loc}(\mathbf{substrs})_v(x)$ where **substrs**$_v(x)$ is located within x (as defined in Section 3.1). **start**$_v(x)$ is the left endpoint of $\mathbf{I}_v(x)$. This is equal to the sum of the lengths of all strings at leaves that are to the left of (lexicographically precede) v. Note that $\mathbf{I}_v(x) = (\mathbf{start}_v(x), \mathbf{start}_v(x) + |\mathbf{substrs}_v(x)|]$.

- $\mathbf{Grid}_v$ denotes the subgrid $\mathbf{Grid}_{\mathbf{I}_v(x) \times \mathbf{I}_v(y)}$, and $\mathbf{E}_v = \mathbf{E}_v(x,y)$ denotes the set of its edges. $\mathbf{E}_v^+$ denotes the set of annotated edges $\{e^+ : e \in \mathbf{E}_v\}$.

- $\mathbf{ED}_v = \mathbf{ED}(\mathbf{substrs}_v(x), \mathbf{substrs}_v(y))$, and the canonical path $\mathbf{canon}(x,y)_{\mathbf{I}_v(x) \times \mathbf{I}_v(y)}$ is denoted by $\mathbf{canon}_v$. $\mathbf{costly}(\mathbf{canon}_v)$ is the set of costly edges of $\mathbf{canon}_v$ (which has size $\mathbf{ED}_v$), and $\mathbf{costly}(\mathbf{canon}_v^+)$ is the set of costly edges with annotations.

The procedure MAIN-RECONSTRUCT, using subroutines FIND-STRINGS and FIND-LOCATIONS constructs a set **candids** of annotated edges as follows:

- FIND-STRINGS takes as input **str-sk**(x) and **str-sk**(y) and constructs (partial) labelings **found-str**(x) and **found-str**(y) of the tree $T(W^d)$. For a node $v \in W^{\leq d}$, if **found-str**$_v(x)$ (resp. **found-str**$_v(y)$) is defined, its value is a string.

- FIND-LOCATIONS takes as input **loc-sk**(x) and **loc-sk**(y) and constructs two (partial) labelings **found-start**(x) and **found-start**(y) of $T(W^d)$. For a node $v \in W^d$, if **found-start**$_v(x)$ (resp. **found-str**$_v(y)$) is defined, its value is a nonnegative integer.

- In MAIN-RECONSTRUCT, the set **loc-nodes** is defined to be the set of nodes v for which **found-str**$_v(x)$, **found-str**$_v(y)$, **found-start**$_v(x)$ and **found-start**$_v(y)$ are all defined.

- In MAIN-RECONSTRUCT the set **top-nodes** is defined to be the subset of **loc-nodes** consisting of those nodes in **loc-nodes** that have no ancestor in **loc-nodes**.

- For each node $v \in$ **top-nodes**, MAIN-RECONSTRUCT computes a set of annotated edges **candids**$_v$.

- The output of MAIN-RECONSTRUCT is the set **candids** $= \bigcup_{v \in \mathbf{top\text{-}nodes}}$ **candids**$_v$.

For each $v \in$ **top-nodes** the set **candids**$_v$ is computed in MAIN-RECONSTRUCT as follows. **found-str**$_v(x)$ and **found-str**$_v(y)$ together with **found-start**$_v(x)$ and **found-start**$_v(y)$ determine an edit distance problem, and that problem has a unique canonical path. The set of costly annotated edges in that path is **candids**$_v$. In the procedure MAIN-RECONSTRUCT, **candids**$_v$ is evaluated only for $v \in$ **top-nodes**, but the definition makes sense for all $v \in$ **loc-nodes**, and in what follows we will refer to **candids**$_v$ for $v \in$ **loc-nodes**.

We are hoping that **candids** is close, in a suitable probabilistic sense to the set **costly**(**canon**$^+(x,y)$). To show this we will need to consider certain properties for classifying nodes.

The first set of properties is determined by **substrs**$_v(x)$ and **substrs**$_v(y)$ produced by the MAIN-SKETCH.

- v is *compatible* if the box $\mathbf{I}_v(x) \times \mathbf{I}_v(y)$ is (x,y)-compatible. (Recall that the box $I \times J$ is (x,y)-compatible provided that the the canonical alignment **canon**(x,y) passes through the corner points of $\mathbf{I}_v(x) \times \mathbf{I}_v(y)$.) The importance of this property is that if v is compatible then **canon**$_v$ is equal to **canon**$(x,y) \cap \mathbf{E}_v$, the portion of **canon**(x,y) that lies inside **Grid**$_v$. This means that if we are able to reconstruct the substrings **substrs**$_v(x)$ and **substrs**$_v(y)$ and the intervals $\mathbf{I}_v(x)$ and $\mathbf{I}_v(y)$, **costly**(**canon**$_v^+$) will give us the portion of **costly**(**canon**$^+(x,y)$) that lies inside of **Grid**$_v$.

- v is *compatibly split* provided that all of the children of v are compatible. This depends on the run of BASIC-DECOMP on the strings **substrs**$_v(x)$ and **substrs**$_v(y)$ that occurs inside of MAIN-DECOMP.

We say that:

- **found-str**$_v(x)$ (respectively, **found-str**$_v(y)$) *is a correct reconstruction* provided that it is defined and equal to **substrs**$_v(x)$ (respective, **substrs**$_v(y)$).

- **found-start**$_v(x)$ (resp. **found-start**$_v(y)$) *is a correct reconstruction* provided that it is defined and equal to **start**$_v(x)$ (resp. **start**$_v(y)$).

- We say that v *is correctly reconstructed* if **found-str**$_v(x)$, **found-str**$_v(y)$, **found-start**$_v(x)$ and **found-start**$_v(y)$ are all correct reconstructions.

Lemma 7.6. *Let $v \in W^{\leq d}$. Under a normal execution:*

1. *If **found-str**$_v(x)$ and **found-str**$_v(y)$ are both defined by* FIND-STRINGS *then **found-str**$_v(x) =$ **substrs**$_v(x)$ and **found-str**$_v(y) =$ **substrs**$_v(y)$.*

2. *If **found-start**$_v(x)$ and **found-start**$_v(y)$ are both defined by* FIND-LOCATIONS *then **start**$_v(x) =$ **found-start**$_v(x)$ and **start**$_v(y) =$ **found-start**$_v(y)$.*

Proof. Fix a node $v \in W^{\leq d}$ and let $j = |v|$.

Proof of Part 1. $\textbf{found-str}_v(x)$ and $\textbf{found-str}_v(y)$ are determined during iteration j of the main loop in FIND-STRINGS as follows: HMR-RECOVER($\textbf{str-sk}^j(x), \textbf{str-sk}^j(y)$) is evaluated and assigned to $\textbf{gram-mismatch}^j$. This is a set of mismatch triples of the form $((w, i), \alpha, \beta)$ where $(w, i) \in W^j \times \{1, \ldots, N\}$, and $\alpha \neq \beta \in \mathbb{F}$. Letting $\textbf{gram-mismatch}_v^j$ denote the set of those triples involving the node v, $\textbf{found-gram}_v(x)$ and $\textbf{found-gram}_v(y)$ are determined by $\textbf{gram-mismatch}_v^j$.

Under a normal execution, the call to HMR-RECOVER is Sound, so $\textbf{gram-mismatch}_v^j$ is a subset of the mismatch information MIS($\textbf{grams}_v^j(x), \textbf{grams}_v^j(y)$).

Claim 7.7. $\textbf{found-gram}_v(x) \leq \textbf{grams}_v(x)$ *and* $\textbf{found-gram}_v(y) \leq \textbf{grams}_v(y)$, *where* $\leq$ *is entry-wise.*

Proof. We prove the first inequality; the second is analogous. Fix an index $i \in \{1, \ldots, N\}$. Assume that $\textbf{found-gram}_{v,i}(x) = 1$ we must show that $\textbf{grams}_{v,i}(x) = 1$. From the construction of $\textbf{found-gram}(x)$ in FIND-STRINGS, $\textbf{found-gram}_{v,i}(x) = 1$ implies that there is a mismatch triple $((v, i), \alpha, \beta) \in \textbf{gram-mismatch}^j$ such that $\alpha \neq 0$. Since $\textbf{gram-mismatch}^j \subseteq$ MIS($\textbf{grams}^j(x), \textbf{grams}^j(y)$) this implies that $\textbf{grams}_{v,i}^j(x) = \alpha \neq 0$ which implies that $\textbf{grams}_{v,i}(x) = 1$ by the definition of $\textbf{grams}^j$. $\square$

Now if $\textbf{found-gram}_v(x) = \textbf{grams}_v(x)$ and $\textbf{found-gram}_v(y) = \textbf{grams}_v(y)$ then we have $\textbf{found-str}_v(x) = $ BASIC-DECODE($\textbf{found-gram}_v(x)$) = BASIC-DECODE($\textbf{grams}_v(x)$) = $\textbf{substrs}_v(x)$ and similarly $\textbf{found-str}_v(y) = \textbf{substrs}_v(y)$. The conclusion of the first implication holds in this case. Now consider the case $\textbf{found-gram}_v(x) \neq \textbf{grams}_v(x)$. By Claim 7.7, $\textbf{found-gram}_v(x) < \textbf{grams}_v(x)$. By the minimality property of BASIC-DECODE (see Section 4.3) $\textbf{found-str}_v(x) = $ BASIC-DECODE($\textbf{found-gram}_v(x)$) is UNDEFINED. So the premise of the implication does not hold. Similarly when $\textbf{found-gram}_v(y) \neq \textbf{grams}_v(y)$. This completes the proof of the first part of the lemma.

The argument for the second part of the lemma is similar, but has important differences. $\textbf{found-start}_v(x)$ and $\textbf{found-start}_v(y)$ are determined in FIND-LOCATIONS. The main loop of FIND-LOCATIONS has ds iterations, corresponding to the levels of the binary tree $T(\{0, 1\}^{ds})$. In iteration $i = 0, \ldots, ds - 1$, HMR-RECOVER is applied to $\textbf{loc-sk}^i(x), \textbf{loc-sk}^i(y)$ to produce $\textbf{size-mismatch}^i$. The sketches $\textbf{loc-sk}^i(x)$ and $\textbf{loc-sk}^i(y)$ are sketches of the D-vectors $\textbf{left-size}^i(x)$ and $\textbf{left-size}^i(y)$ for $D = \{0, 1\}^i$. $\textbf{size-mismatch}^i$ is a set of mismatch triples of the form (w, α, β) where $w \in \{0, 1\}^i$ and $\alpha, \beta \in \mathbb{F}$. Let L_i be the subset of nodes u that appear as first coordinate of some mismatch triple in $\textbf{size-mismatch}^i$. $\textbf{size-mismatch}^i$ is used to define $\textbf{found-size}_u(x)$ and $\textbf{found-size}_u(y)$ for all $u \in L_i$. The soundness of HMR-RECOVER (since the execution is normal) implies $L_i \subseteq$ MIS($\textbf{left-size}^i(x), \textbf{left-size}^i(y)$) and this implies $\textbf{found-size}_u(x) = \textbf{left-size}_u(x)$ and $\textbf{found-size}_u(y) = \textbf{left-size}_u(y)$.

Now $\textbf{found-start}_v(x)$ is the sum of $\textbf{found-size}_u(x)$ over left ancestors u of v. It is UNDEFINED if any of the summands is undefined. If all summands are defined, then by the above, the sum is equal to the sum of $\textbf{left-size}_u(x)$ over all left ancestors u of v which is equal to $\textbf{start}_v(x)$. The same argument applies to show $\textbf{found-start}_v(y) = \textbf{start}_v(y)$, completing the proof of the second part of the lemma. $\square$

Corollary 7.8. *Under a normal execution, suppose that $v \in W^{\leq d}$ is placed in* $\textbf{loc-nodes}$ *by* MAIN-RECONSTRUCT. *Then*

1. v is correctly reconstructed, i.e. $\textbf{candids}_v = \textbf{costly}(\textbf{canon}_v^+)$.

2. If, in addition, v is compatible, then $\textbf{candids}_v = \textbf{costly}(\textbf{canon}^+(x, y)) \cap \textbf{E}_v^+$.

Proof. By the code for MAIN-RECONSTRUCT, $v \in \textbf{loc-nodes}$ implies $\textbf{found-str}_v(x)$, $\textbf{found-str}_v(y)$, $\textbf{found-start}_v(x)$ and $\textbf{found-start}_v(y)$ are all defined, and $\textbf{candids}_v$ is the set of costly edges in the canonical path for the edit distance problem defined by $\textbf{found-str}_v(x)$, $\textbf{found-str}_v(y)$, $\textbf{found-start}_v(x)$ and $\textbf{found-start}_v(y)$. By Lemma 7.6, this is equal to the set of costly edges in the canonical path associated to $\textbf{substrs}_v(x)$, $\textbf{substrs}_v(y)$, $\textbf{I}_v(x)$ and $\textbf{I}_v(y)$ which is the set $\textbf{canon}_v^+$. For the second part, when v is compatible, the conclusion follows from the first part together with Proposition 3.5. $\qquad\square$

8   Proof of Lemma 7.5.

Suppose S is an arbitrary slice. Without loss of generality, assume that $S = S_V(b)$ is the b-th vertical edge slice. We say that *slice S is correctly reconstructed* provided that $\textbf{candids}(x, y) \cap S = \textbf{costly}(\textbf{canon}^+(x, y)) \cap S$. We need to show:

$$\textbf{Pr}[\text{slice } S \text{ is not correctly reconstructed} \wedge \textbf{Normal}] \to 0 \qquad (2)$$

as n gets large.

Since $\textbf{substrs}(x)$ is a string decomposition tree of x, for each level $j \in \{0, \ldots, d\}$ there is a unique node $w(j)$ of $T(W^d)$ such that $b \in \textbf{I}_{w(j)}(x)$, and the sequence $w(0), \ldots, w(d)$ forms a root-to-leaf path. The following lemma implies that, under a normal execution, whether S is correctly reconstructed only depends on the reconstruction of nodes in $w(0), \ldots, w(d)$.

Proposition 8.1. *Assume a normal execution, and suppose S is the b-th vertical edge slice.*

1. $\textbf{candids} \cap S = \bigcup_{j \in \{0, \ldots, d\} : w(j) \in \textbf{top-nodes}} \textbf{candids}_{w(j)} \cap S$

2. *If $w(j)$ is compatible and $w(j) \in \textbf{loc-nodes}$ then $\textbf{candids}_{w(j)} \cap S = \textbf{costly}(\textbf{canon}^+) \cap S$.*

Proof. For the first part, by definition in MAIN-RECONSTRUCT $\textbf{candids} \cap S = \bigcup_{v \in \textbf{top-nodes}} \textbf{candids}_v \cap S$. To establish the claimed equality it suffices to show that $\textbf{candids}_v \cap S = \emptyset$ for $v \notin \{w(0), \ldots, w(d)\}$. By Corollary 7.8, $\textbf{candids}_v = \textbf{costly}(\textbf{canon}_v^+)$. The path $\textbf{canon}_v$ only contains edges in vertical slices $S_V(\ell)$ for which $\ell \in \textbf{I}_v(x)$. Since $b \notin \textbf{I}_v(x)$, $\textbf{canon}_v \cap S = \emptyset$.

For the second part, suppose $w(j)$ is compatible. By Corollary 7.8(2), $\textbf{candids}_{w(j)} = \textbf{costly}(\textbf{canon}^+) \cap \textbf{E}_{w(j)}^+$. Since $b \in \textbf{I}_{w(j)}(x)$, $\textbf{E}_{w(j)}^+$ includes the edge of $\textbf{canon}$ from slice S, and the equality follows. $\qquad\square$

As a corollary we get a sufficient condition for S to be correctly reconstructed:

Corollary 8.2. *Assume a normal execution. If either of the following conditions hold then S is correctly reconstructed:*

1. $\textbf{costly}(\textbf{canon}^+) \cap S = \emptyset$ *and none of the nodes $w(0), \ldots, w(d)$ is in $\textbf{loc-nodes}$.*

2. *There is an index $j \in \{1, \ldots, d\}$ such that $w(j) \in \textbf{loc-nodes}$ and $w(0), \ldots, w(j)$ are all compatible.*

Proof. For (1), if $\{w(0), \ldots, w(d)\} \cap \textbf{loc-nodes} = \emptyset$ then by Proposition 8.1(1), $\textbf{candids} \cap S = \emptyset = \textbf{costly}(\textbf{canon}^+) \cap S$.

For (2), let j be the least index satisfying the hypothesis of (2). By definition of **top-nodes** in MAIN-RECONSTRUCT, $w(j)$ is the unique node among $w(0), \ldots, w(d)$ that belongs to **top-nodes** and so by Proposition 8.1(1), $\textbf{candids} \cap S = \textbf{candids}_{w(j)} \cap S$, which is equal to $\textbf{costly}(\textbf{canon}^+) \cap S$ by Proposition 8.1(2). $\square$

Our strategy is to show that with fairly high probability one of the two sufficient conditions of this corollary is satisfied. To this end we define the following events, indexed by $j \in \{0, \ldots, d\}$:

$\textbf{T}(j)$ is the event that $\textbf{ED}_{w(j)} < t_j$.

$\textbf{T}(\leq j)$ is the event that $\textbf{ED}_{w(i)} < t_i$ for all $i \leq j$.

$\textbf{C}(j)$ is the event that $w(j)$ is compatible.

$\textbf{C}(\leq j)$ is the event that $w(0), \ldots, w(j)$ are all compatible.

Recall that we have a global hypothesis that $\textbf{ED}(x, y) < \lceil 20 k s_{\text{OR}} \rceil_2 = t_0$, so that $\textbf{T}(0)$ holds. Also, $\textbf{C}(0)$ holds trivially. The following proposition says that if $\textbf{ED}_{w(j)} < t_j$ and $w(j)$ is compatible then $w(j+1)$ is almost certainly compatible:

Proposition 8.3. *1. For all $j \in \{1, \ldots, d\}$, $\textbf{Pr}[\textbf{T}(j-1) \wedge \textbf{C}(j-1) \wedge \neg\textbf{C}(j) \wedge \textbf{Normal}] \leq \frac{1}{\log^2 n}$.*

2. $\textbf{Pr}[\exists j \in \{1, \ldots, d\}$ such that $\textbf{T}(\leq j-1) \wedge \neg\textbf{C}(\leq j) \wedge \textbf{Normal}] \leq \frac{1}{\log n}$.

Proof. For the first part, note that events $\textbf{T}(j-1)$ and $\textbf{C}(j-1)$ are determined by the first $j-1$ levels of $\textbf{substrs}(x)$ and $\textbf{substrs}(y)$. Consider an arbitrary setting of the first $j-1$ levels of $\textbf{substrs}(x)$ and $\textbf{substrs}(y)$ for which $\textbf{T}(j-1)$ and $\textbf{C}(j-1)$ hold. Then $\textbf{ED}_{w(j-1)} < t_{j-1}$. By Theorem 4.3, the conditional probability given this setting that $w(j-1)$ is not compatibly split is at most:

$$\frac{s_{\text{split}} \, \textbf{ED}_{w(j-1)}}{k_j} \leq \frac{s_{\text{split}} \, t_{j-1}}{k_j} \leq \frac{1}{\log^2 n}, \tag{3}$$

where the final inequality uses the definitions of t_{j-1} and k_j. Since this holds for any setting of the first $j-1$ levels of $\textbf{substrs}(x)$ and $\textbf{substrs}(y)$ for which $\textbf{T}(j-1)$ and $\textbf{C}(j-1)$ hold, we conclude that:

$$\textbf{Pr}[\textbf{T}(j-1) \wedge \textbf{C}(j-1) \wedge \neg\textbf{C}(j) \wedge \textbf{Normal}] \leq \frac{1}{\log^2 n},$$

as required for the first part.

For the second part, if there is an index j such that $\textbf{T}(\leq j-1) \wedge \neg\textbf{C}(\leq j)$ and i is the least such index then $\textbf{T}(i-1) \wedge \textbf{C}(i-1) \wedge \neg\textbf{C}(i)$ holds. Therefore:

$$\textbf{Pr}[\exists j \in \{1, \ldots, d\}, \textbf{T}(\leq j-1) \wedge \neg\textbf{C}(\leq j) \wedge \textbf{Normal}]$$

$$\leq \sum_{i=1}^{j} \textbf{Pr}[\textbf{T}(j-1) \wedge \textbf{C}(j-1) \wedge \neg\textbf{C}(j) \wedge \textbf{Normal}] \leq \frac{j}{\log^2 n} \leq \frac{1}{\log n} \tag{4}$$

where the second to last inequality uses the first part of this proposition. $\square$

Let $q \in \{0, \ldots, d\}$ be the largest index for which $\mathbf{T}(\leq q)$ holds. We emphasize that q is a random variable that depends on the execution. The following result says that with fairly high probability $w(0), \ldots, w(q)$ are all compatible and if $q < d$ then with fairly high probability $w(q+1)$ is also compatible.

Corollary 8.4. *1. $\mathbf{Pr}[(q < d) \wedge w(0), \ldots, w(q+1)$ are not all compatible $\wedge \mathbf{Normal}] \leq \frac{1}{\log n}$.*

2. $\mathbf{Pr}[(q = d) \wedge w(0), \ldots, w(d)$ are not all compatible $\wedge \mathbf{Normal}] \leq \frac{1}{\log n}$.

Proof. For both parts, $\mathbf{T}(\leq q)$ holds so the event whose probability is being bounded is included in the event whose probability is bounded in the second part of Proposition 8.3. $\square$

To prove (2) we need an upper bound on:

$\mathbf{Pr}[S$ is not correctly reconstructed $\wedge \mathbf{Normal}]$
$$= \mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q = d)]$$
$$+ \mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q < d)] \quad (5)$$

The first term is easy to bound:

Proposition 8.5.

$$\mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q = d)] \leq \frac{1}{\log n}$$

Proof. We write the probability as:

$\mathbf{Pr}[S$ is not correctly reconstructed $\wedge \mathbf{Normal} \wedge (q = d)]$
$$= \mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q = d) \wedge \mathbf{C}(\leq d)]$$
$$+ \mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q = d) \wedge \neg\mathbf{C}(\leq d)]. \quad (6)$$

In the final expression, the second term is at most $\frac{1}{\log n}$ by Corollary 8.4(2). We claim that the first term is 0. To prove this we assume a normal execution where $q = d$ and $\mathbf{C}(\leq d)$ both hold and show that S is necessarily correctly reconstructed. Since $\mathbf{C}(\leq d)$ holds, every node in $w(0), w(1), \ldots, w(d)$ is compatible. If one of them belongs to **loc-nodes** then by Corollary 8.2(2), S is correctly reconstructed.

Now assume that no node of $\{w(0), \ldots, w(d)\}$ belongs to **loc-nodes**. By assumption, $\mathbf{ED}_{w(d)} < t_d = 1$, which implies that $\mathbf{substrs}_{w(d)}(x)$ and $\mathbf{substrs}_{w(d)}(y)$ are the same string, which implies that $\mathbf{costly}(\mathbf{canon}_{w(d)}) = \emptyset$, and the desired conclusion follows from Corollary 8.2(1). $\square$

To bound the second term in (5), we have:

$\mathbf{Pr}[S$ not correctly reconstructed $\wedge \mathbf{Normal} \wedge (q < d)] =$
$$\mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q < d) \wedge \neg\mathbf{C}(\leq q+1)]+$$
$$\mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q < d) \wedge \mathbf{C}(\leq q+1)]. \quad (7)$$

The first term is at most $\frac{1}{\log n}$ by the Corollary 8.4(1). To bound the second term define the events:

STR(v): **found-str**$_v(x)$ and **found-str**$_v(y)$ are both defined.

LOC(v): **found-start**$_v(x)$ and **found-start**$_v(y)$ are both defined.

SIZ(v): **found-size**$_v(x)$ and **found-size**$_v(y)$ are both defined.

We will prove:

Lemma 8.6. $Pr[\neg\boldsymbol{STR}(w(q+1)) \wedge \boldsymbol{Normal} \wedge (q < d) \wedge \boldsymbol{C}(\leq q+1)] \leq \frac{1}{\log n}$.

Lemma 8.7. $Pr[\neg\boldsymbol{LOC}(w(q+1)) \wedge \boldsymbol{Normal} \wedge (q < d) \wedge \boldsymbol{C}(\leq q+1)] \leq \frac{1}{\log n}$.

STR$(w(q+1))$ and **LOC**$(w(q+1))$ together imply that $w(q+1)$ is correctly reconstructed. If in addition **C**$(w(q+1))$ holds then by Corollary 8.2(2), S is correctly reconstructed. Therefore these two lemmas imply:

$$\mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q < d) \wedge \mathbf{C}(\leq q+1)] \leq$$
$$\mathbf{Pr}[\neg\mathbf{STR}(w(q+1)) \wedge \mathbf{Normal} \wedge (q < d) \wedge \mathbf{C}(\leq q+1)]+$$
$$\mathbf{Pr}[\neg\mathbf{LOC}(w(q+1)) \wedge \mathbf{Normal} \wedge (q < d) \wedge \mathbf{C}(\leq q+1)] \leq \frac{2}{\log n}.$$

This bounds the second term on the right of (7). Combined with the upper bound on the first term we have $\mathbf{Pr}[S \text{ not correctly reconstructed} \wedge \mathbf{Normal} \wedge (q < d)] \leq \frac{3}{\log n}$, and combining this with Proposition 8.5 yields $\mathbf{Pr}[S \text{ is not correctly reconstructed} \wedge \mathbf{Normal}] \leq \frac{4}{\log n}$ as required for Lemma 7.5.

To prove Lemmas 8.6 and 8.7 we will need.

Proposition 8.8. *Assume a normal execution for strings x,y. Let $v \in W^{\leq d}$ with $j = |v|$ such that $\boldsymbol{substrs}_v(x) \neq \boldsymbol{substrs}_v(y)$.*

1. *The following two conditions are together sufficient for $\boldsymbol{found\text{-}str}_v(x)$ and $\boldsymbol{found\text{-}str}_v(y)$ to both be defined:*

   (a) *$\boldsymbol{OR\text{-}prints}_v(x) \neq \boldsymbol{OR\text{-}prints}_v(y)$*

   (b) *For the HMR-recovery problem associated to $(\boldsymbol{grams}^j, T(W^j \times \{1, \ldots, N\}), \kappa^{\leq j}, 4(d+1), 1/n^4)$ (see the procedure GRAMMAR-CONDENSE), all of the nodes $v_{\leq 0}, \ldots, v_{\leq j-1}$ are $\frac{1}{4(d+1)}$-underloaded (as defined in Section 5.1)*

2. *The following condition is sufficient for $\boldsymbol{found\text{-}start}_v(x)$ and $\boldsymbol{found\text{-}start}_v(y)$ to both be defined: View v as a node of $T(\{0,1\}^{ds})$ where its depth is js. For every $i \leq js$, for the HMR-recovery problem associated to $(\boldsymbol{left\text{-}size}^i, T(\{0,1\}^i), \lambda^{\leq i}, 4ds, 1/n^4)$ all of the nodes $v_{\leq 0}, \ldots, v_{\leq i-1}$ are $\frac{1}{4ds}$-underloaded.*

Proof. In both parts we apply Theorem 5.1. Since the execution is normal, there is no HMR-abnormality so all executions of HMR-RECOVER satisfy Completeness (see Section 5.1), that every mismatch triple with accessible index is correctly recovered, and Soundness, that no incorrect triple is recovered.

When HMR-RECOVER is applied to HMR-RECOVER($\mathbf{str\text{-}sk}^j(x), \mathbf{str\text{-}sk}^j(y)$) it recovers all $\frac{1}{4d}$-accessible mismatch triples correctly. We claim that all indices belonging to $\{(v, i) : i \in \{1, \ldots, N\}\}$ are $\frac{1}{4d}$-accessible and for this we need that $v_{\leq 0}, \ldots, v_{\leq j} = v$ are all $\frac{1}{4d}$-underloaded in $T(W^j \times \{1, \ldots, N\})$. Condition 2(b) says that $v_{\leq 0}, \ldots, v_{\leq j-1}$ are all $\frac{1}{4d}$-underloaded. v itself is also $\frac{1}{4d}$-underloaded, because the load on v is at most the number of mismatches between $\mathbf{grams}_v^j(x)$ and $\mathbf{grams}_v^j(y)$ (defined by GRAMMAR-CONDENSE) which is at most the total number of non-zero entries in both. By Theorem 4.3, $\mathbf{grams}_v(x)$ and $\mathbf{grams}_v(y)$ each have at most k_j non-zero entries, so the load of v is at most $2k_j$ which is at most $\frac{1}{4\log(n)}\kappa_j$ (with room to spare).

Therefore all indices in $\{(v, i) : i \in \{1, \ldots, N\}\}$ are $\frac{1}{4d}$-accessible and so all mismatches between $\mathbf{grams}_v^j(x)$ and $\mathbf{grams}_v^j(y)$ are recovered correctly. Condition 2(a) requires $\mathbf{OR\text{-}prints}_v(x) \neq \mathbf{OR\text{-}prints}_v(y)$ hence all non-zero entries of $\mathbf{grams}_v^j(x)$ and $\mathbf{grams}_v^j(y)$ are mismatches. Thus $\mathbf{found\text{-}gram}_v(x) = \mathbf{grams}_v(x)$ and $\mathbf{found\text{-}gram}_v(y) = \mathbf{grams}_v(y)$ which implies BASIC-DECODE($\mathbf{found\text{-}gram}_v(x)) = \mathbf{substrs}_v(x)$ and BASIC-DECODE($\mathbf{found\text{-}gram}_v(y) = \mathbf{substrs}_v(y)$, completing the proof of the first part.

For the second part, we are now working on the binary tree $T(\{0, 1\}^{ds})$, from the code for FIND-LOCATIONS, if $\mathbf{found\text{-}size}_{v_{\leq i}}(x)$ is defined for all $i < j$ then $\mathbf{found\text{-}start}_v(x)$ will be defined. $\mathbf{found\text{-}size}_{v_{\leq i}}(x)$ is defined from the run of HMR-RECOVER on the HMR-sketch of ($\mathbf{left\text{-}size}^i$, $T(\{0, 1\}^i), \lambda^{\leq i}, 4ds, 1/n^4$) produced by LOCATION-CONDENSE. The mismatch triples of this run correspond to nodes w at level i for which $\mathbf{left\text{-}size}_w^i(x) \neq \mathbf{left\text{-}size}_w^i(y)$ or $\mathbf{fingerprints}_w^*(x) \neq \mathbf{fingerprints}_w^*(y)$. Since we are assuming a normal execution, $\mathbf{fingerprints}_w^*(x) \neq \mathbf{fingerprints}_w^*(y)$ if and only if $\mathbf{substrs}_w^*(x) \neq \mathbf{substrs}_w^*(y)$. Since $\mathbf{substrs}_v(x) \neq \mathbf{substrs}_v(y)$ by hypothesis, it follows that $\mathbf{substrs}_w(x) \neq \mathbf{substrs}_w(y)$ for all ancestors w of v. (If $\mathbf{substrs}_w(x) = \mathbf{substrs}_w(y)$ then BASIC-DECOMP acts identically on both and so the $\mathbf{substrs}(x)$ and $\mathbf{substrs}(y)$ would be identical below w.) Therefore for $i \leq j$, there is a mismatch triple corresponding to $v_{\leq i}$. Thus in the HMR-recovery problem associated to ($\mathbf{left\text{-}size}^i, T(\{0, 1\}^i), \lambda^{\leq i}, 4ds, 1/n^4$), if every node on the path from $v_{\leq i}$ to the root (including $v_{\leq i}$) is $\frac{1}{4ds}$-underloaded then the mismatch triple associated to $v_{\leq i}$ will be recovered. The hypothesis of the second part ensures that $v_{\leq 0}, \ldots, v_{\leq i-1}$ are all $\frac{1}{4ds}$-underloaded. $v_{\leq i}$ is also because its load is just 1. Therefore $\mathbf{found\text{-}size}_{v_{\leq i}}(x)$ and $\mathbf{found\text{-}size}_{v_{\leq i}}(y)$ will both be defined, as required to conclude that $\mathbf{found\text{-}start}_v(x)$ and $\mathbf{found\text{-}start}_v(y)$ to both be defined. $\quad \square$

It remains to prove Lemmas 8.6 and 8.7.

8.1 Proof of Lemma 8.6

For i, j with $0 \leq i \leq j \leq d$ define the following event:

$\mathbf{F}(i, j)$: $w(i)$ is $\frac{1}{4(d+1)}$-overloaded with respect to the HMR-recovery problem associated with ($\mathbf{grams}^j, T(W^j \times \{1, \ldots, N\}), \kappa^{\leq j}, 4(d+1), 1/n^4$), which means the load of node $w(i)$ in $T(W^j \times \{1, \ldots, N\})$ satisfies $\widehat{\kappa}_{w(i)}^{\leq j} \geq \frac{1}{4(d+1)}\kappa_i^{\leq j}$.

Note that by the first part of Proposition 8.8:

$$\mathbf{Pr}[\neg\mathbf{STR}(w(q+1)) \wedge \mathbf{Normal} \wedge (q < d) \wedge \mathbf{C}(\leq q+1)]$$

$$\leq \mathbf{Pr}[(\mathbf{OR\text{-}prints}_{w(q+1)}(x) = \mathbf{OR\text{-}prints}_{w(q+1)}(y)) \wedge (q < d) \wedge \mathbf{Normal}\,]$$

$$+ \mathbf{Pr}[\bigvee_{0 \leq i \leq q} (\mathbf{F}(i, q+1) \wedge \mathbf{Normal} \wedge (q < d) \wedge \mathbf{C}(\leq i) \wedge T(\leq i))]$$

$$= \mathbf{Pr}[\bigvee_{0 \leq i \leq q} (\mathbf{F}(i, q+1) \wedge \mathbf{Normal} \wedge (q < d) \wedge \mathbf{C}(\leq i) \wedge T(\leq i))]$$

$$\leq \mathbf{Pr}[\bigvee_{0 \leq i < j \leq d} (\mathbf{F}(i, j) \wedge \mathbf{Normal} \wedge \mathbf{C}(\leq i) \wedge T(\leq i))] \quad (8)$$

The equality holds since $\mathbf{ED}_{w(q+1)} \geq t_{q+1}$ by definition of q and so a normal execution implies $\mathbf{OR\text{-}prints}_{w(q+1)}(x) \neq \mathbf{OR\text{-}prints}_{w(q+1)}(y)$.

To prove an upper bound on the right hand side of (8) we will prove an upper bound on $\mathbb{E}[\widehat{\kappa}_{w(i)}^{\leq j}]$ and use Markov's inequality. To obtain such an upper bound, we introduce an auxiliary integer labeling $\mathbf{load\text{-}ub}^j$ on the internal nodes of $T(W^j \times \{1, \ldots, N\})$ which depends on the execution of ED-SKETCH on x and y and is an upper bound on $\widehat{\kappa}^{\leq j}$. (Below, $s_{\mathrm{E} \to \mathrm{H}}$ is the parameter appearing in Theorem 4.3.)

$$\mathbf{load\text{-}ub}_v^j = \begin{cases} \min(s_{\mathrm{E} \to \mathrm{H}} \mathbf{ED}_v, \kappa_j) & \text{if } |v| = j \text{ and } \mathbf{OR\text{-}prints}_v(x) = \mathbf{OR\text{-}prints}_v(y) \\ 2k_j & \text{if } |v| = j \text{ and } \mathbf{OR\text{-}prints}_v(x) \neq \mathbf{OR\text{-}prints}_v(y) \\ \sum_{w \in \mathbf{child}(v)} \mathbf{load\text{-}ub}_w^j & \text{if } |v| < j \text{ and } v \text{ is compatibly split} \\ \kappa_{|v|} & \text{if } |v| < j \text{ and } v \text{ is not compatibly split} \end{cases}$$

Proposition 8.9. *Suppose $j \in \{1, \ldots, d\}$. For all internal nodes v of $T(W^j \times \{1, \ldots, N\})$, $\widehat{\kappa}_v^{\leq j} \leq$* ***load-ub****$_v^j$.*

Proof. Fix $j \in \{1, \ldots, d\}$ and let v be an internal node of $T(W^j \times \{1, \ldots, N\})$. We proceed by reverse induction on $|v|$.

The base case is $|v| = j$. $\widehat{\kappa}^{\leq j}$ is defined on the nodes of $T(W^j \times \{1, \ldots, N\})$. The leaves of this tree are pairs $(v, i) \in W^j \times \{1, \ldots, N\}$ and by definition, $\widehat{\kappa}_{v \circ i}^{\leq j}$ is 1 if and only if $\mathbf{grams}_{v \circ i}^j(x) \neq \mathbf{grams}_{v \circ i}^j(y)$. Thus for v with $|v| = j$ in $T(W^j \times \{1, \ldots, N\})$, by definition $\widehat{\kappa}_v^{\leq j} = \min(\kappa_j, \sum_{i=1}^N \widehat{\kappa}_{v \circ i}^{\leq j}) = \min(\kappa_j, \mathbf{Ham}(\mathbf{grams}_v^j(x), \mathbf{grams}_v^j(y)))$.

$\mathbf{Ham}(\mathbf{grams}_v^j(x), \mathbf{grams}_v^j(y)) = \mathbf{Ham}(\mathbf{grams}_v(x) \times_{\mathbb{F}} \mathbf{OR\text{-}prints}_v(x), \mathbf{grams}_v(y) \times_{\mathbb{F}} \mathbf{OR\text{-}prints}_v(y))$. If $\mathbf{OR\text{-}prints}_v(x) = \mathbf{OR\text{-}prints}_v(y)$, by Theorem 4.3, $\mathbf{Ham}(\mathbf{grams}_v(x), \mathbf{grams}_v(y)) \leq s_{\mathrm{E} \to \mathrm{H}} \mathbf{ED}_v$, and so $\widehat{\kappa}_v^{\leq j} \leq \min(s_{\mathrm{E} \to \mathrm{H}} \mathbf{ED}_v, \kappa_j)$. If $\mathbf{OR\text{-}prints}_v(x) \neq \mathbf{OR\text{-}prints}_v(y)$, this is at most the total number of nonzero entries of $\mathbf{grams}_v(x)$ and $\mathbf{grams}_v(y)$ which is at most $2k_j$.

Now suppose v is a node of depth $i < j$. By definition:

$$\widehat{\kappa}_v^{\leq j} = \min(\kappa_{|v|}, \sum_{w \in \mathbf{child}(v)} \widehat{\kappa}_w^{\leq j}).$$

If v is not compatibly split, then $\mathbf{load\text{-}ub}_v^j = \kappa_{|v|} \geq \widehat{\kappa}_v^{\leq j}$, as required.

If v is compatibly split, then by the induction hypothesis

47

$$\textbf{load-ub}_v^j \;=\; \sum_{w\in\textbf{child}(v)} \textbf{load-ub}_w^j$$

$$\geq \sum_{w\in\textbf{child}(v)} \widehat{\kappa}_w^{\leq j}$$

$$\geq \widehat{\kappa}_v^{\leq j}.$$

$\square$

Let $i \leq d$. For the analysis we will fix the outcome of the decomposition tree up-to level i, and analyze the decomposition conditioned on the fixed part. Let $g^{\leq i}(x), g^{\leq i}(y)$ be possible outcomes for the random variables $\textbf{substrs}^{\leq i}(x)$ and $\textbf{substrs}^{\leq i}(y)$ and let $A_i = A(g^{\leq i}(x), g^{\leq i}(y))$ denote the event that $\textbf{substrs}^{\leq i}(x) = g^{\leq i}(x)$ and $\textbf{substrs}^{\leq i}(y) = g^{\leq i}(y)$. The event A_i determines the value for $\textbf{ED}_v$ for all v at depth at most i, and we denote this value by $\textbf{ED}_v(A_i)$. A_i also determines whether $\textbf{C}(\leq i)$ and $\textbf{T}(\leq i)$ hold. We now will show:

Proposition 8.10. *Let $i \leq j \leq d$. Let $g^{\leq i}(x)$ and $g^{\leq i}(y)$ be a possible outcome for the first i levels of $\textbf{substrs}(x)$ and $\textbf{substrs}(y)$. Let v be an internal node in $T(W^{\leq j} \times \{1,\ldots,N\})$ at depth i such that under the event A_i, v is compatible and $\textbf{ED}_v < t_v$. Then*

1. *$\mathbb{E}[\textbf{load-ub}_v^j | A_i] \leq s_{\text{load}} \, \textbf{ED}_v(A_i) \times (j - i + 2)$.*

2. *For any R, $\textbf{Pr}[\widehat{\kappa}_v^{\leq j} \geq \frac{1}{R}\kappa_i | A_i] \leq \frac{R}{4\log^5 n}$.*

Note that the event $\widehat{\kappa}_v^{\leq j} \geq \frac{1}{R}\kappa_i^{\leq j}$ is precisely the event that v is R-overloaded, and this will allow us to prove an upper bound on the right hand side of (8).

Proof. The main thing to prove is the first part. The second part follows easily from the first part:

$$
\begin{aligned}
\textbf{Pr}[\widehat{\kappa}_v^{\leq j} \geq \frac{1}{R}\kappa_i | A_i] \;&\leq\; \frac{R\,\mathbb{E}[\widehat{\kappa}_v^{\leq j}|A_i]}{\kappa_i} \\
&\leq\; \frac{R\,\mathbb{E}[\textbf{load-ub}_v^j|A_i]}{\kappa_i} \\
&\leq\; \frac{R\,s_{\text{load}}\,\textbf{ED}_v(A_i)(2\log n)}{\kappa_i} \\
&\leq\; \frac{R\,s_{\text{load}}\,t_i(2\log n)}{\kappa_i} \\
&\leq\; \frac{R}{4\log^5 n}.
\end{aligned}
$$

Here the first inequality comes from Markov's inequality, the second comes from Proposition 8.9, the third comes from the first part of this proposition, the fourth comes from the hypothesis that $\textbf{ED}_v(A_i) < t_i$ and the final inequality follows from the definitions of t_i and κ_i.

We prove the first part by reverse induction on i. The base case is $i = j$. In this case, using Theorem 4.2 we have:

48

$$\mathbb{E}[\textbf{load-ub}_v^j|A_j] = \quad \mathbf{Pr}[\textbf{OR-prints}_v(x) = \textbf{OR-prints}_v(y)|A_j] \cdot s_{\text{E}\to\text{H}} \cdot \mathbf{ED}_v(A_j)$$
$$+ \mathbf{Pr}[\textbf{OR-prints}_v(x) \neq \textbf{OR-prints}_v(y)|A_j] \cdot 2k_j$$
$$\leq \quad s_{\text{E}\to\text{H}}\, \mathbf{ED}_v(A_j) + \frac{\mathbf{ED}_v(A_j) \cdot s_{\text{OR}}}{t_j} \cdot 2k_j$$
$$\leq \quad (s_{\text{E}\to\text{H}} + s_{\text{load}}) \cdot \mathbf{ED}_v(A_j) \leq 2s_{\text{load}} \cdot \mathbf{ED}_v(A_j). \tag{9}$$

Now suppose $i < j$ and condition on A_i. Suppose v is a node at depth i.
Let CS be the set of nodes that are compatibly split

$$\mathbb{E}[\textbf{load-ub}_v^j|A_i] = \quad \mathbf{Pr}[v \notin CS|A_i] \cdot \kappa_i + \mathbf{Pr}[v \in CS|A_i] \cdot \mathbb{E}[\textbf{load-ub}_v^j|A_i \wedge v \in CS]$$
$$\leq \quad \mathbf{Pr}[v \notin CS|A_i] \cdot \kappa_i + \mathbb{E}[\textbf{load-ub}_v^j|A_i \wedge v \in CS].$$

By Theorem 4.3:

$$\mathbf{Pr}[v \notin CS|A_i] \cdot \kappa_i \leq s_{\text{split}} \frac{\mathbf{ED}_v(A_i)}{k_{i+1}}\kappa_i \leq \frac{s_{\text{load}}}{2}\mathbf{ED}_v(A_i). \tag{10}$$

To bound the second summand we condition further on the $(i+1)$-st level of $\textbf{substrs}(x)$ and $\textbf{substrs}(y)$. This extension determines whether v is compatibly split, and we let $g^{\leq i+1}(x), g^{\leq i+1}(y)$ denote a possible extension of $g^{\leq i}(x), g^{\leq i}(y)$ to level $i+1$ such that v is compatibly split. Let $A_{i+1} = A(g^{\leq i+1}(x), g^{\leq i+1}(y))$.

Let $CS_v(A_{i+1})$ be the set of children of v that are compatibly split and $NCS_v(A_{i+1})$ be the set of children of v that are not compatibly split, conditioned on A_{i+1}. In what follows we write $\#A$ for the cardinality of the set A instead of $|A|$ to avoid confusion with the $|$ in conditional expectation.

$$\mathbb{E}[\textbf{load-ub}_v^j|A_{i+1}] = \quad \sum_{w \in \textbf{child}(v)} \mathbb{E}[\textbf{load-ub}_w^j|A_{i+1}]$$
$$\leq \quad \sum_{w \in \textbf{child}(v)} \mathbf{Pr}[w \in NCS_v|A_{i+1}]\kappa_{i+1}$$
$$+ \sum_{w \in \textbf{child}(v)} \mathbf{Pr}[w \in CS_v|A_i+1]s_{\text{load}}\, \mathbf{ED}_w(A_{i+1}) \times (j-i+1)$$
$$\leq \quad \kappa_{i+1} \times \mathbb{E}[\#NCS(v)|A_{i+1}]$$
$$+ s_{\text{load}}\, \mathbf{ED}_v(A_i) \times (j-i+1)$$

For each child w of v, by Theorem 4.3, $\mathbf{Pr}[w \in NCS(v)|A_{i+1}] \leq s_{\text{split}} \frac{\mathbf{ED}_w(A_{i+1})}{k_{i+2}}$. Therefore $\mathbb{E}[\#NCS(v)|A_{i+1}] \leq \sum_{w \in \textbf{child}(v)} \frac{s_{\text{split}}\, \mathbf{ED}_w(A_{i+1})}{k_{i+2}} = s_{\text{split}} \frac{\mathbf{ED}_v(A_i)}{k_{i+2}}$ where the final equality uses that v is compatibly split. Therefore:

$$\kappa_{i+1} \times \mathbb{E}[\#NCS(v)|A_{i+1}] \quad \leq \quad s_{\text{split}} \frac{\kappa_{i+1}}{k_{i+2}}\mathbf{ED}_v(A_i) \leq \frac{1}{2}s_{\text{load}}\, \mathbf{ED}_v(A_i). \tag{11}$$

and so:

$$\mathbb{E}[\textbf{load-ub}_v^j | A_{i+1}] \quad \leq \quad s_{\text{load}} \, \mathbf{ED}_v(A_i) \times (j - i + \frac{3}{2})$$

This upper bound is the same for all extensions A_{i+1} of A_i for which v is compatibly split, so:

$$\mathbb{E}[\textbf{load-ub}_v^j | A_i \wedge (v \in CS)] \quad \leq \quad s_{\text{load}} \, \mathbf{ED}_v(A_i) \times (j - i + \frac{3}{2})$$

Combining with (10) yields the desired conclusion:

$$\mathbb{E}[\textbf{load-ub}_v^j | A_i] \quad \leq \quad s_{\text{load}} \, \mathbf{ED}_v(A_i) \times (j - i + 2)$$

$\square$

Using this Proposition we obtain:

Lemma 8.11. *1. For all i, j with $0 \leq i \leq j \leq d$, $\boldsymbol{Pr}[\boldsymbol{F}(i,j) \wedge \boldsymbol{Normal} \wedge \boldsymbol{C}(\leq i) \wedge T(\leq i)] \leq \frac{1}{\log^4 n}$.*

2. $\boldsymbol{Pr}[\bigvee_{0 \leq i < j \leq d}(\boldsymbol{F}(i,j) \wedge \boldsymbol{Normal} \wedge \boldsymbol{C}(\leq i) \wedge T(\leq i))] \leq \frac{1}{\log^2 n}$.

Proof. For the first part, apply Proposition 8.10(2) with $R = 4(d+1) \leq 4 \log n$ to get $\mathbf{Pr}[\mathbf{F}(i,j) \wedge \mathbf{Normal} | A_i] \leq \frac{1}{\log^4 n}$, and then average over all choices of A_i for which $\mathbf{C}(\leq i) \wedge T(\leq i)$ hold. The second part follows immediately by applying a union bound to the first part, $\square$

The second part of this lemma implies that the right hand side of (8) is at most $\frac{1}{\log n}$, as required to prove Lemma 8.6.

8.2 Proof of Lemma 8.7

The proof of this lemma is closely related to that of Lemma 8.6 and we will reuse parts of that proof. One difference is that in FIND-LOCATIONS the HMR-recovery operates on the tree $T(\{0,1\}^{ds})$ rather than $T(W^d \times \{1, \ldots, N\})$. Since the depth of this tree is up-to $ds - 1$ the overload parameter for the HMR-recovery scheme is set to $\frac{1}{4ds}$ instead of $\frac{1}{4(d+1)}$. Levels of the binary tree are denoted by ordered pairs (i, r) where $0 \leq i \leq d$ and $0 \leq r \leq s - 1$. Level (i, r) refers to level $is + r$ of $T(\{0,1\}^{ds})$. (If $i = d$ then r can only be 0.) Level $(i, 0)$ in the binary tree corresponds to level i in $T(W^d)$. The order on levels (i, r) is lexicographic. We say that v has depth (i, r) if $|v| = is + r$, and has depth at most (i, r) if $|v| \leq is + r$.

The nodes in the path of $T(\{0,1\}^{ds})$ that corresponds to the path from $w(0)$ to $w(d)$ in $T(W^d)$ are denoted by $w(i, r)$ where $(0, 0) \leq (i, r) \leq (d, 0)$.

For $m = is + r$ we abbreviate the HMR-recovery problem $(\textbf{left-size}^m, T(\{0,1\}^m), \lambda^{\leq m}, 4ds, 1/n^4)$ by $\textbf{loc}(i, r)$. We write $\lambda^{\leq(i,r)}$ for $\lambda^{\leq m}$. Recall that by definition, for a node v at depth $(i', r') \leq (i, r)$, its capacity for $\textbf{loc}(i, r)$ is $\lambda_v^{\leq(i,r)} = \kappa_{i'}$, and its load is denoted $\widehat{\lambda}_v^{\leq(i,r)}$.

The algorithm FIND-LOCATIONS, perform ds instances of HMR-recovery corresponding to the labelings $\textbf{left-size}^j$ for $0 \leq j \leq sd - 1$. This is not done for the final level sd because $\textbf{left-size}_v^*$

is not defined for leaves of the binary tree. For analysis purposes it is convenient to augment the algorithm by defining **left-size**$_v = 0$ for all $v \in$ **vis-nodes*** and performing HMR sketch-and-recover for level sd, and including it as part of the sketch. We assume that the (trivial) recovery of this level defines **found-size**$^*_v(x) = $ **found-size**$^*_v(y) = 0$ for all nodes $v \in \{0,1\}^{ds}$ where **substrs**$^*_v(x) \neq$ **substrs**$^*_v(y)$.) This additional step plays no role in recovering the canonical alignment of x, y, but makes it easier to reuse a part of the proof of Lemma 8.6. We must prove:

$\quad\quad$ **Pr**$[\neg$**LOC**$(w(q+1,0)) \wedge$ **Normal** $\wedge (q < d) \wedge$ **C**$(\leq q+1)] \leq \frac{1}{\log n}$.

$\quad\quad$ Analogous to the events **F**(i,j) defined in the previous section, for $(i,r) \leq (j,t)$, we define the bad events:

L$((i,r),(j,t))$ is the event that $w(i,r)$ is $\frac{1}{4ds}$-overloaded with respect to **loc**(j,t), i.e., $\widehat{\lambda}^{\leq(j,t)}_{w(i,r)} \geq \frac{1}{4ds}\kappa_i$.

We have:

$$
\begin{aligned}
\textbf{Pr}[\quad \neg\textbf{LOC} \quad &(w(q+1,0)) \wedge \textbf{Normal} \wedge (q < d) \wedge \textbf{C}(\leq q+1)] \\
\leq \quad &\textbf{Pr}[\bigvee_{(j,t)<(q+1,0)} \neg\textbf{SIZ}(w(j,t)) \wedge \textbf{Normal} \wedge (q<d) \wedge \textbf{C}(\leq q+1)] \\
\leq \quad &\textbf{Pr}[\bigvee_{(i,r)<(j,t)\leq(d,0)} \textbf{L}((i,r),(j,t)) \wedge \textbf{Normal} \wedge \textbf{C}(\leq i) \wedge \textbf{T}(\leq i)] \quad\quad (12)
\end{aligned}
$$

We will bound this last quantity by $\frac{1}{\log n}$. The first step is to show that it suffices to bound the probability of bad events of the form **L**$((i,0),(j,0))$:

Proposition 8.12. *For $(i,r) < (j,t) \leq (d,0)$, if $t \neq 0$, event $\boldsymbol{L}((i,r),(j,t))$ implies (is contained in) event $\boldsymbol{L}((i,0),(j+1,0))$, and if $t = 0$ event $\boldsymbol{L}((i,r),(j,t))$ implies $\boldsymbol{L}((i,0),(j,0))$, and therefore:*

$$
\begin{aligned}
\boldsymbol{Pr}[\bigvee_{(i,r)<(j,t)\leq(d,0)} \boldsymbol{L}((i,r),(j,t)) \wedge \boldsymbol{Normal} &\wedge \boldsymbol{C}(\leq i) \wedge \boldsymbol{T}(\leq i)] \\
&= \boldsymbol{Pr}[\bigvee_{i<j\leq d} \boldsymbol{L}((i,0),(j,0)) \wedge \boldsymbol{Normal} \wedge \boldsymbol{C}(\leq i) \wedge \boldsymbol{T}(\leq i)] \quad (13)
\end{aligned}
$$

Proof.

Claim 8.13. *For any $(i,r) < (j,t)$, $\boldsymbol{L}((i,r),(j,t))$ implies $\boldsymbol{L}((i,0),(j,t))$.*

Proof. From the recurrence relation for $\widehat{\lambda}^{\leq(j,t)}$, and the fact that λ of a node is at most λ of its parent it follows that $\widehat{\lambda}^{\leq(j,t)}$ at a node is at most the $\widehat{\lambda}^{\leq(j,t)}$ of its parent, and therefore $\widehat{\lambda}^{\leq(j,t)}$ does not decrease along paths to the root. Therefore $\widehat{\lambda}^{\leq(j,t)}_{w(i,r)} \leq \widehat{\lambda}^{\leq(j,t)}_{w(i,0)}$. Since $\lambda_{(i,0)} = \lambda_{(i,r)} = \kappa_i$, **L**$((i,r),(j,t))$ implies **L**$((i,0),(j,t))$. $\square$

Claim 8.14. *If $(i,0) \leq (j,t)$ with $t \neq 0$ then $\boldsymbol{L}((i,0),(j,t))$ implies $\boldsymbol{L}((i,0),(j+1,0))$.*

Proof. This follows from a more general claim: If v is any node of the binary tree at level $(i,r) \leq (j,t)$ and $(j,t) < (j',t')$ then $\widehat{\lambda}^{\leq(j,t)}_v \leq \widehat{\lambda}^{\leq(j',t')}_v$. We prove this by reverse induction on (i,r). For the basis, $(i,r) = (j,t)$, we have that $\widehat{\lambda}^{\leq(j,t)}_v$ is 0 or 1, and is 1 if and only if **substrs**$^*_v(x) \neq$ **substrs**$^*_v(y)$. So we need that if **substrs**$^*_v(x) \neq$ **substrs**$^*_v(y)$ then $\widehat{\lambda}^{\leq(j',t')}_v \geq 1$. Since **substrs**$^*_v(x) \neq$ **substrs**$^*_v(y)$ there

is a descendant w of v at level (j',t') such that $\mathbf{substrs}^*_w(x) \neq \mathbf{substrs}^*_w(y)$ and so $\widehat{\lambda}_w^{\leq(j',t')} = 1$. Since $\widehat{\lambda}^{\leq(j',t')}$ does not decrease along paths to the root, $\widehat{\lambda}_v^{\leq(j',t')} \geq 1$, as required for the basis step. For the induction step, if $(i,r) < (j,t)$ then by induction we have:

$$
\begin{aligned}
\widehat{\lambda}_v^{\leq(j,t)} &= \min(\lambda_v^{\leq(j,t)}, \sum_{w \text{ child of } v} \widehat{\lambda}^{\leq(j,t)}) \\
&\leq \min(\lambda_v^{\leq(j',t')}, \sum_{w \text{ child of } v} \widehat{\lambda}_v^{\leq(j',t')}) = \widehat{\lambda}_v^{\leq(j',t')},
\end{aligned}
$$

$\square$

Combining the claims we have that for $t \neq 0$, $\mathbf{L}((i,r),(j,t))$ implies $\mathbf{L}((i,0),(j,t))$ implies $\mathbf{L}((i,0),(j,0))$, and if $t = 0$ then the first claim gives $\mathbf{L}((i,r),(j,0))$ implies $\mathbf{L}((i,0),(j,0))$. $\square$

It is useful to recall the connection between the trees $T(W^d)$ and $T(\{0,1\}^{sd})$. Since $W = \{0,1\}^s$, there is a natural mapping from W^d to $\{0,1\}^{sd}$, where $v = v_1 \ldots, v_j$ maps to the binary string $v_1 \circ \cdots \circ v_j$ of length sj. This gives a 1-1 correspondence from nodes at level j in $T(W^d)$ to nodes at level $(j,0)$ in $T(\{0,1\}^{sd})$. For node v in $T(W^d)$, let v' be the associated node in $T(\{0,1\}^{sd})$. The HMR-recovery problem for $(\mathbf{grams}^j, T(W^j \times \{1,\ldots,N\}), \kappa^{\leq j}, 4ds, 1/n^4)$ analyzed in the previous subsection operated on the tree $T(W^j \times \{1,\ldots,N\})$.

Proposition 8.15. *For any $j \leq d$, and for any node v of $T(W^j)$, for any execution, if the corresponding node v' in $T(\{0,1\}^{sd})$ is $\frac{1}{4ds}$-overloaded for $\mathbf{loc}(j,0)$ then v is $\frac{1}{4ds}$-overloaded for the HMR-recovery problem for $(\mathbf{grams}^j, T(W^j \times \{1,\ldots,N\}), \kappa^{\leq j}, 4ds, 1/n^4)$.*

Proof. First we claim that for any non-leaf node v of $T(W^j \times \{1,\ldots,N\})$, $\widehat{\lambda}_{v'}^{\leq(j,0)} \leq \widehat{\kappa}_v^{\leq j}$. We prove this by reverse induction on $|v|$. For the basis, $|v| = j$, $\widehat{\lambda}_{v'}^{\leq(j,0)} \in \{0,1\}$ and is 1 if and only if $\mathbf{substrs}_v(x) \neq \mathbf{substrs}_v(y)$. If $\mathbf{substrs}_v(x) \neq \mathbf{substrs}_v(y)$ then $\mathbf{grams}_v(x) \neq \mathbf{grams}_v(y)$, which implies that for some $a \in \{1,\ldots,N\}$ the child (v,a) of v satisfies $\mathbf{grams}_{v \circ a}(x) \neq \mathbf{grams}_{v \circ a}(y)$ which implies that $\widehat{\kappa}_v^{\leq j} \geq 1$.

For the induction step, suppose $|v| = i < j$. For a node u at level less than $(i+1,0)$ in $T(\{0,1\}^{ds})$, let $D(u)$ be the set of its descendants at level $(i+1,0)$.

Claim 8.16. *For all u that is a descendant of v at level less than $(i+1,0)$ (including v itself):*
$\widehat{\lambda}_u^{\leq(j,0)} = \min(\kappa_i, \sum_{w \in D(u)} \widehat{\lambda}_w^{\leq(j,0)})$.

This claim follows easily by (reverse) induction using the recurrence $\widehat{\lambda}_u^{\leq(j,0)} = \min(\lambda_u, \sum_{w \in \mathbf{child}(u)} \widehat{\lambda}_w^{\leq(j,0)})$ and the fact that $\lambda_{(i,r)} = \kappa_i$ for any $r \in [0,s)$.

Observe that $D(v') = \{w' : w$ is a child of v in $T(W^d)\}$. From the previous claim using the induction, we have:

$$
\begin{aligned}
\widehat{\lambda}_{v'}^{\leq(j,0)} &= \min(\kappa_i, \sum_{w \text{ a child of } v \text{ in } T(W^d)} \widehat{\lambda}_{w'}^{\leq(j,0)}) \\
&\leq \min(\kappa_i, \sum_{w \text{ a child of } v} \widehat{\kappa}_w^{\leq j}) = \widehat{\kappa}_v^{\leq j}.
\end{aligned}
$$

Since $\lambda_{v'}^{\leq(j,0)} \leq \kappa_v^{\leq j}$, it follows that if v' is overloaded with respect to $\mathbf{loc}(j,0)$ then v is $\frac{1}{4ds}$-overloaded with respect to $(\mathbf{grams}^j, T(W^j \times \{1,\ldots,N\}), \kappa^{\leq j}, 4ds, 1/n^4)$. $\square$

We can now complete the proof of Lemma 8.7 Given these propositions we have:

$$\mathbf{Pr}[\bigvee_{(i,r)<(j,t)\leq(d,0)} \mathbf{L}((i,r),(j,t)) \wedge \mathbf{Normal} \wedge \mathbf{C}(\leq i) \wedge \mathbf{T}(\leq i)]$$

$$= \mathbf{Pr}[\bigvee_{i<j\leq d} \mathbf{L}((i,0),(j,0)) \wedge \mathbf{Normal} \wedge \mathbf{C}(\leq i) \wedge \mathbf{T}(\leq i)]$$

$$\leq \sum_{i<j\leq d} \mathbf{Pr}[\mathbf{L}((i,0),(j,0)) \wedge \mathbf{Normal} \wedge \mathbf{C}(\leq i) \wedge \mathbf{T}(\leq i)]$$

$$= \sum_{i<j\leq d} \mathbf{Pr}[\widehat{\lambda}_{w(i,0)}^{\leq(j,0)} \leq \frac{1}{4ds}\kappa_i \wedge \mathbf{Normal} \wedge \mathbf{C}(\leq i) \wedge \mathbf{T}(\leq i)]$$

$$\leq \sum_{i<j\leq d} \mathbf{Pr}[\widehat{\kappa}_{w(i)}^{\leq j} \leq \frac{1}{4ds}\kappa_i \wedge \mathbf{Normal} \wedge \mathbf{C}(\leq i) \wedge \mathbf{T}(\leq i)]. \quad (14)$$

Using the second part of Proposition 8.10, each term in the final expression is at most $\frac{ds}{\log^5 n} \leq \frac{1}{\log^3 n}$, and summing over all i,j gives an upper bound of $\frac{1}{\log n}$, as required to prove Lemma 8.7.

9 Constraints for the choice of our parameters

In Section 6.1 we defined various parameters for the scheme. The parameters t_i, k_i and κ_i are adjusted to satisfy various constraints that arise in the analysis. As stated in Theorem 7.1, the length of the sketch is $\tilde{O}(\kappa_0)$ so we choose the parameters to satisfy the constraints while (approximately) minimizing κ_0. For convenient reference, here we collect the constraints that these parameters must satisfy.

- $t_0 = \lceil 20ks_{\mathrm{OR}}\rceil_2$. This ensures that condition $\mathbf{C}(0)$ holds as noted prior to Proposition 8.3,

- $t_d = 1$ which is used inside the proof of Proposition 8.5.

For $j \in \{1,\ldots,d\}$:

1. $k_j \geq \kappa_{j-1}\frac{2s_{\mathrm{split}}}{s_{\mathrm{load}}}$. This is used both in (10) and (11) to bound the contribution of nodes that are not compatibly split to the expectation of $\widehat{\kappa}^{\leq j}$.

2. $t_j \geq \frac{2s_{\mathrm{OR}}}{s_{\mathrm{load}}}k_j$. This is used in (9) to bound the contribution to $\widehat{\kappa}^{\leq j}$ from leaves for which $\mathbf{OR\text{-}prints}_v(x) \neq \mathbf{OR\text{-}prints}_v(y)$.

3. $k_j \geq (s_{\mathrm{split}}\log^2 n) \cdot t_{j-1}$. This is used in (3) to bound the probability that a node v for which $\mathbf{OR\text{-}prints}_v(x) = \mathbf{OR\text{-}prints}_v(y)$ is not compatibly split by $\frac{1}{\log^2 n}$. This constraint does not need to be explicitly enforced because it follows from the combination of constraints 1 and 4

4. $\kappa_j \geq (8s_{\mathrm{load}}\log^6 n) \cdot t_j$. This allows us to use Markov's inequality to deduce the second part of Proposition 8.10 to bound the probability that a node is overloaded with respect to $\widehat{\kappa}^{\leq j}$. $\kappa_j \geq 4k_j$. When doing HMR-RECOVER on the sketches $\mathbf{str\text{-}sk}^j(x), \mathbf{str\text{-}sk}^j(y)$, every node v at level j is underloaded.

5. $\kappa_j \geq (8 \log n) \cdot k_j$. This is used in the proof of Proposition 8.8. This constraint does not need to be explicitly enforced because it follows from the combinations of constraints 3 and 4.

6. $s_{\text{load}} \geq s_{\text{E}\rightarrow\text{H}}$. This is used in the basis step of the proof of Proposition 8.10. It is implied by the constraint obtained by multiplying together 1, 2, and 4.

References

[AN20]   Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. *CoRR*, abs/2005.07678, 2020.

[AS08]   Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.

[BI18]   Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly sub-quadratic time (unless seth is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018.

[BK23]   Sudatta Bhattacharya and Michal Koucký. Locally consistent decomposition of strings with applications to edit distance sketching. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 219–232. ACM, 2023.

[BR20]   Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 685–698. ACM, 2020.

[BZ16]   Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 51–60, 2016.

[CDG+20] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *J. ACM*, 67(6):36:1–36:22, 2020.

[CEPR09] Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 778–784. SIAM, 2009.

[CGK16]   Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 712–725. ACM, 2016.

[CKP19]   Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1106–1125. SIAM, 2019.

[Die96]   Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In Claude Puech and Rüdiger Reischuk, editors, *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 569–580. Springer, 1996.

[FKSV02]   Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. An approximate l1-difference algorithm for massive data streams. *SIAM J. Comput.*, 32(1):131–151, 2002.

[FNSS92]   Amos Fiat, Moni Naor, Jeanette P. Schmidt, and Alan Siegel. Nonoblivious hashing. volume 39, pages 764–782, 1992.

[GKLS22]   Arun Ganesh, Tomasz Kociumaka, Andrea Lincoln, and Barna Saha. How compression and approximation affect efficiency in string distance measures. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2867–2919. SIAM, 2022.

[Gra16]   Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discret. Appl. Math.*, 212:96–103, 2016.

[JNW21]   Ce Jin, Jelani Nelson, and Kewen Wu. An improved sketching algorithm for edit distance. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021,*, volume 187 of *LIPIcs*, pages 45:1–45:16, 2021.

[KPS21]   Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small-space and streaming pattern matching with k edits. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 885–896, 2021.

[KR87]   Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, mar 1987.

[KS20]   Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 699–712. ACM, 2020.

[LMS98]   Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, 1998.

[MP80]   William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

[OR07]   Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *J. ACM*, 54(5):23, 2007.

[PL07]   Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, volume 4580 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2007.

[WF74]   Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.