Locally consistent decomposition of strings with applications to edit distance sketching

Sudatta Bhattacharya*[1] and Michal Koucký†[1]

[1]Computer Science Institute of Charles University, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic

Abstract

In this paper we provide a new locally consistent decomposition of strings. Each string x is decomposed into blocks that can be described by grammars of size $\widetilde{O}(k)$ (using some amount of randomness). If we take two strings x and y of edit distance at most k then their block decomposition uses the same number of grammars and the i-th grammar of x is the same as the i-th grammar of y except for at most k indexes i. The edit distance of x and y equals to the sum of edit distances of pairs of blocks where x and y differ. Our decomposition can be used to design a sketch of size $\widetilde{O}(k^2)$ for edit distance, and also a rolling sketch for edit distance of size $\widetilde{O}(k^2)$. The rolling sketch allows to update the sketched string by appending a symbol or removing a symbol from the beginning of the string.

1 Introduction

Edit distance is a measure of similarity of two strings. It measures how many symbols one has to insert, delete or substitute in a string x to get a string y. The measure has many applications from text processing to bioinformatics. The edit distance $\mathrm{ED}(x, y)$ of two strings x and y can be computed in time $O(n^2)$ by a classic dynamic programming algorithm [WF74]. Save for poly-log improvements in the running time [MP80, Gra16], the best known running time for edit distance computation is $O(n + k^2)$ [LMS98], where $k = \mathrm{ED}(x, y)$. Assuming Strong Exponential Time Hypothesis (SETH) this running time cannot be substantially improved [BI15]. The conditional lower bound does not exclude some approximation algorithms, though, and there was a recent progress on computing edit distance in almost-linear time to within some constant factor approximation [CDG+18, KS20, BR20, AN20].

Another problem for edit distance that saw a major progress in recent years is sketching. In sketching we want to map a string x to a short sketch $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x)$ so that from sketches $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x)$ and $\mathrm{sk}^{\mathrm{ED}}_{n,k}(y)$ of two strings x and y we can compute their edit distance, either exactly or approximately. Apriori it is not even obvious that short sketches for edit distance exist. In a surprising construction, Belazzougui and Zhang [BZ16] gave an exact edit distance sketch of size $O(k^8 \log^5 n)$ bits. The sketch size was then improved to $O(k^3 \log^2(\frac{n}{\delta}) \log n)$ bits by Jin, Nelson and Wu [JNW21], where the $\mathrm{ED}(x, y)$ was computed

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

†Email: koucky@iuuk.mff.cuni.cz. Partially supported by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 823748 (H2020-MSCA-RISE project CoSP).

exactly from the sketches with probability at least $1-\delta$, if $\mathrm{ED}(x,y) \le k$. The current best sketch is of size $O(k^2 \log^3 n)$ bits and was given by Kociumaka, Porat and Starikovskaya [KPS21]. [JNW21] gives a lower bound $\Omega(k)$ on the size of a sketch for exact edit distance.

The major problem in edit distance computation as well as in sketching is how to align the matching parts of two strings x and y. Finding an optimal alignment of two strings is the crux in the computation of edit distance and its sketching. In sketching finding a good alignment is even more challenging as we do not have both strings in our hands simultaneously to look for the matching. To the best of our knowledge, to resolve this issue all edit distance sketches use *CGK random walk* on strings [CGK16] which allows to embed the edit distance metrics into Hamming distance metrics with distortion $O(k)$. The walk implicitly fixes some reasonably good matching between the two strings. Going from the CGK random walk to a sketch is non-trivial undertaking and all three sketch results rely on sophisticated machinery to achieve it.

In this paper we provide a new technique to align two strings x and y in oblivious manner. In nutshell, we provide a decomposition procedure that breaks x and y into the same number of "short" blocks so that at most k pairs of blocks in the decomposition of x and y differ, and all other pairs of blocks are matching in an optimal alignment. So the edit distance of x and y is the sum of edit distances of the differing blocks. To be more specific our blocks are not short in their length but they are short in the sense that each of them can be described by a context-free grammar of size $\widetilde{O}(k)$. Our decomposition algorithm constructs the grammars. Our decomposition is based on *locally consistent parsing* of strings a technique similar to the one used in [SV94, BES06, Jow12, BGP20]. Our main technical result is:

Theorem 1.1 (String decomposition)**.** *There is an algorithm running in time $\widetilde{O}(nk)$ that for each string x of length at most n produces grammars $G^x_1, \dots, G^x_s$ such that with probability at least $1 - O(1/\sqrt{n})$, $x = \mathrm{eval}(G^x_1) \cdots \mathrm{eval}(G^x_s)$ and each of the grammars is of size $\widetilde{O}(k)$. Furthermore, for any two strings x and y of edit distance at most k with grammars $G^x_1, \dots, G^x_s$ and $G^y_1, \dots, G^y_{s'}$, resp., that are produced by the algorithm using the same randomness, the following is true simultaneously with probability at least $4/5$:*

1. *$s = s'$,*
2. *$G^x_i = G^y_i$, for all $i \in \{1, \dots, s\}$ except for at most k indices i, and*
3. *$\mathrm{ED}(x,y) = \sum_i \mathrm{ED}(\mathrm{eval}(G^x_i), \mathrm{eval}(G^y_i))$.*

Here, for a grammar G, $\mathrm{eval}(G)$ denotes its evaluation. Our decomposition can be used immediately to give an embedding of edit distance into Hamming distance with distortion $O(k)$. It also readily yields a sketch for exact edit distance of size $\widetilde{O}(k^2)$:

Theorem 1.2 (Sketch for edit distance)**.** *There is a randomized sketching algorithm $\mathrm{sk}^{\mathrm{ED}}_{n,k}$ that on an input string x of length at most n produces a sketch $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x)$ of size $\widetilde{O}(k^2)$ in time $\widetilde{O}(nk)$, and a comparison algorithm running in time $\widetilde{O}(k^2)$ such that given two sketches $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x)$ and $\mathrm{sk}^{\mathrm{ED}}_{n,k}(y)$ for two strings x and y of length at most n obtained using the same randomness of the sketching algorithm outputs with probability at least $1 - 1/n$ (over the randomness of the sketching and comparison algorithms) the edit distance of x and y if it is less than k and ∞ otherwise.*

Furthermore, we can also provide a *rolling sketch*, a sketch in which we can update the stored string by appending a symbol or removing its first symbol.

Theorem 1.3 (Rolling sketch for edit distance)**.** *There are algorithms $\mathrm{Append}(sk_x, a)$, $\mathrm{Remove}(sk_{ax}, a)$, and $\mathrm{Compare}(sk_x, sk_y)$ such that for integer parameters $k \le m$:*

1. *Given a sketch sk_x representing a string x and a symbol a,* $\mathrm{Append}(sk_x, a)$ *outputs a sketch sk_{xa} for the string xa in time $\widetilde{O}(k^2)$.*
2. *Given a sketch sk_{ax} representing a string ax for a symbol a,* $\mathrm{Remove}(sk_{ax}, a)$ *outputs a sketch sk_x for the string x in time $\widetilde{O}(k^2)$.*
3. *Given two sketches sk_x and sk_y representing strings x and y obtained from the same random sketch for empty string using two sequences of at most m operations* Append *and* Remove, $\mathrm{Compare}(sk_x, sk_y)$ *calculates the edit distance of x and y if it is less than k, and outputs ∞ otherwise. The algorithm* $\mathrm{Compare}(sk_x, sk_y)$ *runs in time $\widetilde{O}(k^2)$.*

All the sketches are of size $\widetilde{O}(k^2)$. The probability that any of the algorithms fails or produces incorrect output is at most $1/m$ over the initial randomness of the sketch for empty string and internal randomness of the algorithms.

We remark that we did not attempt to optimize the running time of either of our algorithms, or poly-log factors in the sketch sizes, and we believe that both parameters can be readily improved by usual amortization techniques of processing symbols in batches of size $\widetilde{O}(k)$. We believe that building the sketch in the first theorem can be done in time $\widetilde{O}(n)$ using fast multi-point polynomial evaluation for $\widetilde{O}(k)$-wise independent hash functions, the update time in the last theorem can be improved to $\widetilde{O}(1)$ by buffering $\widetilde{O}(k)$ symbols that shall be inserted or removed without affecting the other parameters of the algorithm.

Another distinguishing feature of our decomposition procedure compared to the technique of CGK random walks is its parallelizability. CGK random walk seems inherently sequential whereas our decomposition procedure can be easily parallelized. We believe that our decomposition will allow for further applications beyond our simple sketches.

1.1 Related work

The problem of embedding edit distance to other distance measures, like Hamming distance, ℓ_1, etc. has been studied extensively. In [CGK16], the authors have given a randomized embedding from edit distance to Hamming distance, where any string $x \in \{0,1\}^n$ can be mapped to a string $f(x) \in \{0,1\}^{3n}$, given a random string $r \in \{0,1\}^{\log^2 n}$, such that, $\mathrm{ED}(x,y)/2 \le \mathrm{Ham}(f(x), f(y)) \le O(\mathrm{ED}(x,y)^2)$ with probability at least $2/3$. Batu, Ergun and Sahinalp [BES06] have introduced a dimensionality reduction technique, where any string x of length n can be mapped to a string $f(x)$ of length at most n/r, for any parameter r, with a distortion of $\widetilde{O}(r)$. They used the locally consistent parsing technique for their embedding. Ostrovsky and Rabani [OR07] gave an embedding from edit distance to ℓ_1 distance with a distortion of $O(\sqrt{\log n \log\log n})$. Jowhari [Jow12] also gave a randomized embedding from edit distance to ℓ_1 distance with a distortion of $O(\log n \log^* n)$. He used the embedding given by Cormode and Muthukrishnan [CM02] who showed that any string x of length n can be mapped to a vector $f(x)$ of length $m = O(2^{n \log n})$, such that for any pair of strings x, y of length n each, $\mathrm{ED}(x,y)/2 \le \|f(x) - f(y)\|_{\ell_1} \le O(\log n \log^* n) \cdot \mathrm{ED}(x,y)$. Since the size of the vector was too large, [Jow12] used random hashing to get his final embedding.

1.2 Our techniques

We first provide the intuition for our technique. We would like to break a string x into small blocks *obliviously* so that when a string y is broken by the same procedure, the difference between x and y caused by the edit operations is confined within the corresponding blocks of x and y, and the overall decomposition is not affected by them. For random binary strings x and y this could be done fairly easily: look on all the (overlapping) windows of $\log n$ consecutive bits in each of the strings and for each window decide at random whether to make a break at that window or not. To make it consistent between x and y use some random hash function $H : \{0,1\}^{\log n} \to \{0,\dots,D-1\}$ so that if the hash function evaluates to 0 on a given window then start a next block of the decomposition. If we chose D suitably, say $D \geq 10k \log n$, then we are unlikely to start a new block in any window which is affected by the the at most k edit operations on x and y. In that case we obtain the desired decomposition. Hence, decomposing random strings x and y is easy.

The issue is what to do with non-random strings. Consider for example strings x and y that are very sparse, so they contain $\sqrt{n}$ ones sprinkled within the vast ocean of zeros. The hash function H will see mostly windows of 0's and occasionally a window of the form $0^i 10^{\log(n)-i-1}$. The decomposition will have no effect on such strings despite the fact that the string might contain $\Omega(\sqrt{n})$ bits of entropy.

However, we can compress such sparse strings: replace stretches of zeros by some binary encoded information about their length, and try to break the strings again. Still, this will fail if in our example the stretches of zeros are replaced by stretches of some repeated pattern such as $(01)^*$. So we need slightly more general compression which will compress any $\log n$ bits into $\log(n)/2$ bits. By repeating the sequence of steps: split and compress, we will eventually get the desired decomposition of each string.

Our actual algorithm mimics the above intuition. It is technically easier to work with a larger alphabet, so we extend the input alphabet Σ by adding special compression symbols into the work alphabet Γ. (Without loss of generalization we can assume that Σ is of size $O(n^3)$ otherwise we can hash each symbol of our input strings using some perfect hash function into an alphabet of size $O(n^3)$ without affecting the edit distance of a given pair of strings.) To split a string we will use a random $\widetilde{O}(k)$-wise independent hash function $H : \Gamma^2 \to \{0,\dots,D-1\}$, for $D = \Theta(k \log n)$. If the hash function is zero on a pair of consecutive symbols in a string, we start a new block of the decomposition on the first symbol in the pair.

Then in each resulting block we replace stretches of repeated symbols by a special compression symbol from Γ representing the block, and we use a pair-wise independent hash function $C : \Gamma^2 \to (\Gamma \setminus \Sigma)$ to compress non-overlapping pairs of symbols into one symbol. This latter step requires some care as we have to make sure that we select non-overlapping pairs in the same way in x and y. For the selection of non-overlapping pairs we use the locally consistent coloring of Cole and Vishkin [CV86, Lin87, Lin92] where the selection of pairs depends only on the context of $O(\log^* n)$ symbols. The compression reduces the size of each block by a factor of $2/3$. We repeat the compress and split process for $O(\log n)$ iterations until each compressed block of x is of size at most 2. *Decompression* of each block then gives us the desired decomposition of x. (See Fig. 1 for an illustration.)

It is natural and convenient to represent each of the blocks by a context-free grammar which corresponds to the compression process. We can argue that the grammars will be of size $O(D \log n)$ with high probability. So we can represent each string by a sequence of small grammars so that if x and y are at edit distance at most k then at most k pairs of their grammars will differ, and the sum of the edit distances of differing pairs is the edit distance of x and y. Note, that edit distance of two strings represented by context-free grammars can be computed efficiently [GKLS22]. These are the main ideas behind our decomposition algorithm, and we provide more details in Section 3

Building a sketch from the string decomposition is straightforward: We encode each grammar in binary using fixed number of bits, and we use off-the-shelf sketch for Hamming distance to sketch the sequence of grammars. As the Hamming distance sketch does not recover identical bits but only the mismatched bits we make sure that if two grammars differ then their binary encoding differ in every bit. Over binary alphabet this might be impossible but over large alphabets one could use error-correcting codes to achieve the desired effect of recovering the differing grammars; for simplicity we use the Karp-Rabin fingerprint of the whole grammar to encode the binary 0 and 1 distinctly. See Section 3.3 for the details of our encoding and Section 3.4 for details of the sketch for edit distance.

To design a rolling sketch for edit distance where we can extend the represented string by a new symbol or repeatedly remove the first symbol of the represented string we will employ our decomposition technique together with the rolling sketch for Hamming distance of Clifford, Kociumaka, and Porat [CKP19]. We will argue that appending a new symbol to a string affects only some fixed number of grammars in the decomposition of a string. There is a certain threshold T so that except for the last T grammars the decomposition of a string stays the same regardless of how many other symbols are appended. Hence, we will keep a buffer of at most T *active* grammars corresponding to the recently added symbols, and upon addition of a new symbol we will only update those grammars. We are guaranteed that the grammars before this threshold will stay the same forever, so we can *commit* them into the rolling Hamming sketch (in the form of their binary encoding.) Similarly, we will keep a buffer of up-to T *active* grammars that capture the symbols that were deleted from the sketch most recently. Once they become "mature" enough we can commit them by removing their binary encoding from the rolling Hamming sketch. (See Fig. 3 for an illustration.) This allows to maintain a rolling sketch for edit distance.

Evaluation of an edit distance query on two rolling sketches will use their Hamming sketch to recover differing committed grammars. Together with the active grammars of inserted and deleted symbols this provides enough information for evaluating the edit distance query. Technical details are explained in Section 4. In Section 6 we give a table of parameters used throughout the paper.

2 Notations and preliminaries

For any string $x = x_1x_2x_2\dots x_n$ and integers p, q, $x[p]$ denotes x_p, $x[p,q]$ represents substring $x' = x_p \dots x_q$ of x, and $x[p,q) = x[p,q-1]$. If $q < p$, then $x[p,q]$ is the empty string ε. $x[p,\dots]$ represents $x[p,|x|]$, where $|x|$ is the length of x. "$\cdot$"-operator is used to denote concatenation, e.g $x \cdot y$ is the concatenation of two strings x and y. $\mathrm{Dict}(x) = \{x[i,i+1], i \in [n-1]\}$, is the dictionary of string x, which stores all pairs of consecutive symbols that appear in x. For strings x and y, $\mathrm{ED}(x,y)$ is the minimum number of modifications (*edit operations*) required to change x into y, where a single modification can be adding a character, deleting a character or substituting a character in x. All logarithms are based-2 unless stated otherwise. For integers $p > q$, $\sum_{i=p}^{q} a_i = 0$ by definition regardless of a_i's.

2.1 Grammars

Let $\Sigma \subseteq \Gamma$ be two alphabets and $\# \notin \Gamma$. A *grammar* G is a set of *rules* of the type $c \to ab$ or $c \to a^r$, where $c \in (\Gamma \cup \{\#\}) \setminus \Sigma$, $a, b \in \Gamma$ and $r \in \mathbb{N}$. c is the *left hand side* of the rule, and ab or a^r is the *right hand side* of the rule. $\#$ is the starting symbol. The size $|G|$ of the grammar is the number of rules in G. We only consider grammars where each $a \in \Gamma \cup \{\#\}$ appears on the left hand side of at most one rule of G, we call such grammars *deterministic*. (We assume that rules of the form $c \to a^r$ are stored in implicit (compressed) form.) The $\mathrm{eval}(G)$ is the string from Σ^* obtained from $\#$ by iterative rewriting of the intermediate results by the rules from G. If the rewriting process never stops or stops with a string not from Σ^*, $\mathrm{eval}(G)$ is undefined. Observe, that we can replace each rule of the type $c \to a^r$ by a collection of at most $2\lceil \log r \rceil$ new rules of the other type using some auxiliary symbols. Hence, for each grammar G there is another grammar G' using only the first type of the rules such that $\mathrm{eval}(G) = \mathrm{eval}(G')$ and $|G'| \leq |G| \cdot 2\lceil \log |\mathrm{eval}(G)| \rceil$. Using a depth-first traversal of a deterministic grammar G we can calculate its *evaluation size* $|\mathrm{eval}(G)|$ in time $O(|G|)$. Given a deterministic grammar G and an integer m less or equal to its evaluation size, we can construct in time $O(|G|)$ another grammar G' of size $O(|G|)$ such that $\mathrm{eval}(G') = \mathrm{eval}(G)[m, \dots]$. G' will use some new auxiliary symbols. Given a deterministic grammar G, using a depth-first traversal on symbols reachable from the starting symbol $\#$ we can identify in time $O(|G|)$ the smallest sub-grammar $G' \subseteq G$ with the same evaluation.

We will use the following observation of Ganesh, Kociumaka, Lincoln and Saha [GKLS22]:

Proposition 2.1 ([GKLS22])**.** *There is an algorithm that on input of two grammars G_x and G_y of size at most m computes the edit distance k of $\mathrm{eval}(G_x)$ and $\mathrm{eval}(G_y)$ in time $O((m + k^2) \cdot \mathrm{poly}(\log m + n))$, where $n = |\mathrm{eval}(G_x)| + |\mathrm{eval}(G_y)|$.*

2.2 Rolling Hamming distance sketch

For two strings x and y of the same length, we define their *mismatch information* $\mathrm{MIS}(x, y) = \{(i, x[i], y[i]);\ i \in \{1, \dots, |x|\} \text{ and } x[i] \neq y[i]\}$. The Hamming distance of x and y is $\mathrm{Ham}(x, y) = |\mathrm{MIS}(x, y)|$.

There exist various sketches for Hamming distance, which allow to compute Hamming distance with low error probability [KOR98, FIM$^+$06]. Moreover, [PL07, CKP19] also allow to retrieve the mismatch information. For our purposes we will use the sketch given by Clifford, Kociumaka, and Porat [CKP19].

Let $k \leq n$ be integers and $p \geq n^3$ be a prime. [CKP19] give a randomized sketch for Hamming distance $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p} : \{1, \dots, p-1\}^* \to \{0, \dots, p-1\}^{k+4}$ computable in time $\widetilde{O}(n)$ with the following properties.[1]

Proposition 2.2 ([CKP19])**.** *There is a randomized algorithm working in time $O(k \log^3 p)$ that given sketches $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(x)$ and $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(y)$ of two strings x and y of length $\ell \leq n$ constructed using the same randomness decides whether $\mathrm{Ham}(x, y) \leq k$, and if so returns $\mathrm{MIS}(x, y)$, with probability of error at most $1/n$ over the randomness of the sketches and the internal randomness of the algorithm.*

They also construct the following update procedures for their sketch. We will use them to construct a rolling sketch for edit distance.

Proposition 2.3 (Lemma 2.3 of [CKP19])**.** *For $x \in \{1, \dots, p\}^*$ of length less than n and $a \in \{1, \dots, p\}$, in time $O(k \log p)$ we can compute:*

1. $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(xa)$ *and* $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(ax)$*, given* $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(x)$ *and* a*.*
2. $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(x)$ *given* $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(xa)$ *or* $\mathrm{sk}^{\mathrm{Ham}}_{n,k,p}(ax)$*, and* a*.*

Corollary 2.5 of [CKP19] states that appending a character to a sketch of x can be done even faster namely in amortized time $O(\log p)$.

2.3 Locally consistent coloring

The following color reduction procedure allows for locally consistent parsing of strings. The technique was originally proposed by Cole and Vishkin [CV86] and further studied by Linial [Lin87, Lin92].

Proposition 2.4 ([CV86, Lin87, Lin92])**.** *There exists a function $F_{\mathrm{CVL}} : \Gamma^* \to \{1, 2, 3\}^*$ with the following properties. Let $R = \log^* |\Gamma| + 20$. For each string $x \in \Gamma^*$ in which no two consecutive symbols are the same:*

[1]Clifford, Kociumaka and Porat have the sketch size only $k + 3$ elements but we include as an extra item the randomness of the sketch, which is a single element from $\{0, \dots, p-1\}$ used to compute Karp-Rabin fingerprint.

1. *$|F_{\mathrm{CVL}}(x)| = |x|$ and $F_{\mathrm{CVL}}(x)$ can be computed in time $O(R \cdot |x|)$.*

2. *For $i \in \{1, \ldots, |x|\}$, the i-th symbol of $F_{\mathrm{CVL}}(x)$ is a function of symbols of x only in positions $\{i - R, i - R + 1 \ldots, i + R\}$.*

3. *No two consecutive symbols of $F_{\mathrm{CVL}}(x)$ are the same.*

4. *Out of every three consecutive symbols of $F_{\mathrm{CVL}}(x)$ at least one of them is 1.*

5. *If $|x| = 1$ then $F_{\mathrm{CVL}}(x) = 3$, and otherwise $F_{\mathrm{CVL}}(x)$ starts by 1 and ends by either 2 or 3.*

The first three items are standard for $R = \log^* |\Gamma| + 10$. The other two can be obtained by a simple modification of the output of the standard function. In the output, replace first in parallel each sequence 232 by 212, and then each sequence 323 by 313. This guarantees the fourth condition. To satisfy the fifth condition, if $|x| = 1$, set $F_{\mathrm{CVL}}(x) = 3$, if $|x| = 2$, set $F_{\mathrm{CVL}}(x) = 12$, if $|x| = 3$, set $F_{\mathrm{CVL}}(x) = 123$, and if $|x| = 4$, set $F_{\mathrm{CVL}}(x) = 1212$. If $|x| > 4$ then replace the sequence at the beginning of the output as follows: if it starts by a word from $\{2, 3\}\{2, 3\}1$ replace it by 121, if it starts by $\{2, 3\}1\{2, 3\}\{2, 3\}$ replace it by 1212, if it starts by $\{2, 3\}1\{2, 3\}1$ replace it by 1231. Then at the end of the sequence, replace $1\{2, 3\}1$ by 123, and $1\{2, 3\}\{2, 3\}1$ by 1212. This will increase the local dependency to at most $R = \log^* |\Gamma| + 20$.

3 Decomposition algorithm

In this section we describe our main technical tool that we have developed. It is a randomized procedure that splits a string x into blocks $B_1^x, B_2^x, \ldots, B_s^x$ and for each block it produces a grammar of size at most $S = \widetilde{O}(k)$. Furthermore, if $B_1^x, B_2^x, \ldots, B_s^x$ is the decomposition for a string x and $B_1^y, B_2^y, \ldots, B_{s'}^x$ is the decomposition for a string y, obtained using the same randomness, where $\mathrm{ED}(x, y) \leq k$ then with good probability, $s = s'$ and $B_i^x = B_i^y$ for all but k indices i. The edit distance of x and y can be calculated as $\mathrm{ED}(x, y) = \sum_i \mathrm{ED}(B_i^x, B_i^y)$ where i ranges over the differing blocks.

First we provide an overview of the algorithm, specific details are given in the next sub-section. The decomposition procedure proceeds in $O(\log n)$ rounds. In each round, the algorithm maintains a decomposition of x into *compressed* blocks. In each round each block of size at least two is first *compressed* and then *split*. The compression is done by compressing pairs of consecutive symbols into one using a randomly chosen pair-wise independent hash function $C_\ell : \Gamma^2 \to \Gamma$, where ℓ is the round number (*level*). Non-overlapping pairs of symbols are chosen for compression using a *locally consistent coloring* so that every three symbols shrink to at most two. Prior to the compression of pairs we replace each repeated sequence a^r of a symbol a, $r \geq 2$, by a special character $\mathtt{r}_{a,r}$.

The splitting procedure uses a $\widetilde{O}(k)$-wise independent hash function $H_\ell : \Gamma^2 \to \{0, \ldots, D - 1\}$ to select places where to subdivide each block into sub-blocks, where $D = \widetilde{O}(k)$ is a suitable parameter. We start a new block at each consecutive pair of symbols ab, where $H_\ell(ab) = 0$.

After $O(\log n)$ rounds, each block is compressed into at most two symbols and we output a grammar that can generate the block.

For the correctness of the algorithm we will need to establish several properties of the algorithm. Some of these properties are related to behaviour on a single string x, others analyze the behaviour of the procedure on a pair of strings x and y of edit distance at most k.

The properties we want from the algorithm when it runs on x are the following: In each round, each block should be compressed by factor at least $2/3$ while the size of the required grammar capturing the compression should be $\widetilde{O}(k)$. The former is achieved by the design of the compression procedure. The latter goal is provided by the property of the splitting procedure which makes sure that each block $B = b_1 b_2 \cdot b_m$ resulting from a split has small dictionary $\mathrm{Dict}(B) = \{b_i b_{i+1}, i = 1, \ldots, m-1\}$. In particular, we require $|\mathrm{Dict}(B)| = \widetilde{O}(k)$. The grammar size will be proportional to this dictionary.

For the compression procedure we require that it preserves information so the function C_ℓ is one-to-one on each $\mathrm{Dict}(B)$. Since the total size of all dictionaries is bounded by $\widetilde{O}(n)$ this can be easily achieved by picking C_ℓ at random provided that its range size is $\Omega(n^3)$.

Additionally, we need the following property to hold on a pair of strings x and y of edit distance at most k with good probability: The splitting procedure should never split x or y in a *region* which is affected by edit operations that transform x to y (for some canonical choice of those operations.) The total size of those regions will be again $\widetilde{O}(k)$ so we can satisfy this property if each pair of symbols has probability at most $1/\widetilde{O}(k)$ to start a new block. This constrains the choice of the range size for the splitting function H_ℓ.

In the next section we describe the decomposition algorithm fully, and then we establish its properties.

3.1 Algorithm description

Let n be an upper bound on the length of the input string and $k \leq n$ be given. Set $L = \lceil \log_{3/2} n \rceil + 3$ to be an upper bound on the decomposition depth. Let Σ be an input alphabet of size at most n^3, $\Sigma_c = \{\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_{Ln^3}\}$ and $\Sigma_r = \{\mathtt{r}_{a,r}, a \in \Sigma \cup \Sigma_c, r \in \{2, 3, \ldots, n\}\}$ be auxiliary pair-wise disjoint alphabets. Let $\Gamma = \Sigma \cup \Sigma_c \cup \Sigma_r$ be the working alphabet, and $\#$ be a symbol not in Γ. Notice $|\Gamma| = O(n^5 + |\Sigma|)$. We call symbols from $\Sigma_c^0 = \Sigma$ *level-0 compression symbols*, and for $\ell \geq 1$, symbols from $\Sigma_c^\ell = \{\mathtt{c}_i, (\ell-1)n^3 < i \leq \ell n^3\}$ are *level-ℓ compression symbols*. Additionally, symbols from $\Sigma_r^\ell = \{\mathtt{r}_{a,r} \in \Sigma_r, a$ is a level-$(\ell-1)$ compression symbol$\}$ are also *level-ℓ compression symbols*.

Let $R = \log^* |\Gamma| + 20$, $D = 110R(L+1)k$ and $S = 30DL \log n + 6$ be parameters. The algorithm is a recursive algorithm of depth at most L. It starts by selecting at random several hash functions: For $\ell = 1, \ldots, L$, it selects at random a compression hash function $C_\ell : \Gamma^2 \to \Sigma_c^\ell$ from a pair-wise independent hash family, and for $\ell = 0, \ldots, L$, it selects at random a splitting function $H_\ell : \Gamma^2 \to \{0, \ldots, D-1\}$ from a $(5D \log n)$-wise independent hash family.

Main building blocks of the algorithm are two functions, Compress and Split. The first one compresses strings by a factor of $2/3$, and the other splits strings at random points. Their pseudo-code is provided as Algorithm 1 and 2. We describe them next.

Compress. The function $\mathrm{Compress}(B, \ell)$ takes as input a string B over alphabet Γ of length at least two, and an integer $\ell \geq 1$, which denotes the level number. Divide B into minimum number of blocks $B_1, \ldots, B_m$, $B = B_1 B_2 B_3 \ldots B_m$, so that in each B_i either all the characters are the same, i.e. $B_i = a^r$ for some $a \in \Gamma$ and $r \geq 2$, or no two adjacent characters are the same. The first step is to compress the B_i's which contain repeated characters by simply replacing the whole B_i with the symbol $\mathtt{r}_{a,|B_i|}$, where a is the repeated character. Then for the remaining blocks, the following compression is applied: Let B_i be an uncompressed block. Each character of B_i is colored by applying $F_{\mathrm{CVL}}(B_i)$. Divide B_i into blocks $B_i = B'_1 B'_2 \ldots B'_s$, such that for each B'_j only the first character is colored 1. Now, according to Proposition 2.4, length of each B'_j is either 2 or 3. If $B'_j = ab$, replace it with $C_\ell(ab)$ else if $B'_j = abc$, replace it with $C_\ell(ab) \cdot c$, where $a, b, c \in \Gamma$. The actual pseudo-code given below performs the compression of blocks of repeats in two stages, where in the first stage we replace the repeated sequence a^r by $\mathtt{r}_{a,r} \cdot \#$, and then in the next stage we remove the extra symbol $\#$. This simplifies analysis in Lemma 3.10. Assuming that C_ℓ can be evaluated in time $O(1)$, the running time of $\mathrm{Compress}(B, \ell)$ is dominated by the time needed to compute F_{CVL}-coloring of blocks which is $O(R \cdot |B|)$ in total.

Algorithm 1 $\mathrm{Compress}(B, \ell)$

Input: String B over alphabet Γ of length at least two, and level number ℓ.
Output: String B'' over alphabet Γ.

```
1  Divide $B = B_1 B_2 B_3 \dots B_m$ into minimum number of blocks so that each maximal subword $a^r$ of $B$, for
     $a \in \Gamma$ and $r \geq 2$, is one of the blocks.
2  for each $i \in \{1, \dots, m\}$ do
3      if $B_i = a^r$, where $r \geq 2$ then Set $B'_i = \mathtt{r}_{a,r} \cdot \#$ and color $\mathtt{r}_{a,r}$ by 1 and $\#$ by 2.[2];
4      else Set $B'_i = B_i$ and color each symbol of $B'_i$ according to $F_{\mathrm{CVL}}(B_i)$;
5  end
6  Set $B' = B'_1 B'_2 \cdots B'_m$, $B'' = \varepsilon$, and $i = 1$.
7  while $i < |B'|$ do
8      if $B'[i+1] = \#$ then $B'' = B'' \cdot B'[i]$;
9      else $B'' = B'' \cdot C_\ell(B'[i, i+1])$;
10     $i = i + 2$.
11     if $i \leq |B'|$ and $B'[i]$ is not colored 1 then $B'' = B'' \cdot B'[i]$, $i = i + 1$ ;
12 end
13 Return $B''$.
```

Split. The function takes as input a string B over alphabet Γ of length at least two, and an integer $\ell \geq 1$. The function splits the string B into smaller blocks. The algorithm works as follows: For each $i \in \{2, \dots, |B| - 1\}$, if $H_\ell(B[i, i+1]) = 0$, start a new block at position i. The running time of $\mathrm{Split}(B, \ell)$ is dominated by the time to evaluate H_ℓ at $|B| - 2$ points.

Algorithm 2 $\mathrm{Split}(B, \ell)$

Input: String B over alphabet Γ of length at least two, and level number ℓ.
Output: A sequence of strings $(B_0, B_1, \dots, B_s)$ over alphabet Γ.

```
14 Let $i_1 < \cdots < i_s$ be all $i \in \{2, \dots, |B| - 1\}$ where $H_\ell(B[i, i+1]) = 0$. Set $s = 0$ if no such $i$ exists.
15 Let $i_0 = 1$ and $i_{s+1} = |B| + 1$.
16 For $j = 0, \dots, s$, set $B_j = B[i_j, i_{j+1})$.
17 Return $(B_0, B_1, \dots, B_s)$.
```

The main recursive step of the algorithm is encompassed in function $\mathrm{Process}$. The function gets a block $B \in \Gamma^*$ as its input. The block might have already been compressed previously, so the function also gets dictionaries that allow decompression of the block. If the block is already of length at most two, then the function outputs the block. Otherwise it compresses the block B using $\mathrm{Compress}$, then it subdivides the compressed block using Split, and invokes itself recursively on each sub-block. For the output, each block is represented by a grammar. The grammar is reconstructed from the compressed block and its dictionaries by a simple bread-first search algorithm provided in the function $\mathrm{Grammar}$.

[2]If $a = \mathtt{r}_{b,s}$ for some $b \in \Gamma$ and $s \in \mathbb{N}$, then set $B'_i = \mathtt{r}_{b,rs} \cdot \#$. However, such a situation should never happen during the execution of the algorithm as level-ℓ compression symbol can be introduced only at level ℓ.

Algorithm 3 $\text{Process}(B, (D_1, D_2, \ldots, D_{\ell-1}), \ell)$

```
Input: String $B \in \Gamma^*$, a sequence of dictionaries $D_i \subseteq \Gamma^2$ for decompressing $B$, and level number $\ell$.
Output: A sequence of blocks of $B$ each encoded by a grammar.
18 if $|B| \leq 2$ then Output $\text{Grammar}(B, (D_1, D_2, \ldots, D_{\ell-1}), \ell - 1)$ and return ;
19 $A = \text{Compress}(B, \ell)$.
20 $(B_0, B_1, \ldots, B_s) = \text{Split}(A, \ell)$.
21 For $i = 0, \ldots, s$, $\text{Process}(B_i, (D_1, \ldots, D_{\ell-1}, \text{Dict}(B)), \ell + 1)$.
```

To decompose an input string x into blocks, we first apply function $\text{Split}(x, 0)$ to x and then invoke $\text{Process}(B, (), 1)$ on each of the obtained blocks B. Breaking the string x into sub-blocks guarantees that each block passed to Process has small dictionary whereas the dictionary of x could have been arbitrarily large.

Algorithm 4 $\text{Grammar}(B, (D_1, D_2, \ldots, D_\ell), \ell)$

```
Input: String $B \in \Gamma^*$, a sequence of dictionaries $D_i \subseteq \Gamma^2$ for decompressing $B$.
Output: The smallest grammar $G$ for $B$ based on the dictionaries $D_i$ and hash functions $C_1, \ldots, C_\ell$.
22 Let $C = \{c \in \Sigma_c : c$ appears in $B$ or $\mathtt{r}_{c,r}$ appears in $B$ for some $r\}$.   // Symbols needed to decompress B
23 $G = \{\# \to B\}$.
24 for $j = \ell, \ldots, 1$ do
25     for each $ab \in D_j$ do
26         if $C_j(ab) \in C$ then $G = G \cup \{C_j(ab) \to ab\}$,
27             $C = C \cup \{c \in \Sigma_c; c \in \{a, b\}$ or $\mathtt{r}_{c,r} \in \{a, b\}$ for some $r\}$ ;
28     end
29 end
30 For each $\mathtt{r}_{a,r}$ appearing in any of the rules in $G$, add $\mathtt{r}_{a,r} \to a^r$ to $G$.
31 Return $G$.
```

3.2 Correctness of the decomposition algorithm

Our goal is to establish the following theorem which is a stronger version of Theorem 1.1:

Theorem 3.1. *Let x and y be a pair of strings of length at most n with $\text{ED}(x, y) \leq k$. Let $G^x_1, \ldots, G^x_s$ and $G^y_1, \ldots, G^y_{s'}$ be the sequence of grammars output by the decomposition algorithm on input x and y respectively, using the same choice of random functions $C_1, \ldots, C_L$ and $H_0, \ldots, H_L$. The following is true for n large enough:*

1. *With probability at least $1 - 2/n$, $x = \text{eval}(G^x_1) \cdots \text{eval}(G^x_s)$ and $y = \text{eval}(G^y_1) \cdots \text{eval}(G^y_{s'})$.*
2. *With probability at least $1 - 2/\sqrt{n}$, for all $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, s'\}$, $|G^x_i|, |G^y_j| \leq S$.*
3. *With probability at least $9/10$, $s = s'$, $G^x_i = G^y_i$, for all $i \in \{1, \ldots, s\}$ except for at most k indices i, and $\text{ED}(x, y) = \sum_i \text{ED}(\text{eval}(G^x_i), \text{eval}(G^y_i))$.*

By union bound, all three parts happen simultaneously with probability at least $9/10 - 2/n - 1/\sqrt{n}$ which is $\geq 4/5$ for n large enough.

To prove the theorem we make some simple observations about the algorithm, first.

Lemma 3.2. *For any string B of length at least two, and $\ell \geq 1$, $|\mathrm{Compress}(B,\ell)| \leq \frac{2}{3}|B| + 1$ and $|\mathrm{Compress}(B,\ell)| < |B|$.*

Proof. Let $B = B_1B_2B_3\dots B_m$ be as in the procedure. Every block B_i that equals to a^r, for some a and $r \geq 2$, is reduced to one symbol by the compression. The other blocks are colored using $F_{\mathrm{CVL}}(\cdot)$ and compressed. Unless a block B_i is of size one, the coloring induces division of the block B_i into subwords of size two or three, where the former is compressed into one symbol and the latter into two symbols. Hence, each such a block is compressed to at most $2/3$ of its size. So the only blocks B_i that do not shrink are of size one, and are sandwiched between blocks of repeated symbols (that shrink by a factor of at least two). The worst-case situation is when m is odd, blocks B_i are of size one for odd i, and of size two for even i. In that case the original string B shrinks to size $\lfloor \frac{2}{3}|B| \rfloor + 1$. This proves the first inequality. The second inequality is also clear from the analysis above: The only time the string does not shrink is if it is of size one. □

Corollary 3.3. *On a string B of length at most n, the depth of the recursive calls of* Process *is at most L.*

Indeed, from the previous lemma it follows that each block after ℓ compressions and splits is of size at most $(2/3)^{\ell}|B| + 3$. Hence, after $L = \lceil \log_{3/2} n \rceil + 3$ recursive calls Process must stop the recursion.

Lemma 3.4. *Let $B \in \Gamma^*$ be of length at most n, and $\ell \in \{0,\dots,L\}$. Let $(B_0, B_1, \dots, B_s) = \mathrm{Split}(B,\ell)$ where $H_\ell : \Gamma^2 \to \{0,\dots,D-1\}$ is chosen at random from $(5D\log n)$-wise independent hash family. Then with probability at least $1 - 1/n^3$, for all $j \in \{0,\dots,s\}$, $|\mathrm{Dict}(B_j)| \leq 5D\log n$.*

Proof. If for some $j \in \{0,\dots,s\}$, $|\mathrm{Dict}(B_j)| > 5D\log n$, then there exists $1 < r < t \leq |B|$ such that $|\mathrm{Dict}(B[r,t])| = 5D\log n$ and for all $i \in \{r,\dots,t-1\}$, $H_\ell(B[i,i+1]) \neq 0$. (Pick r to be the position in B of the second symbol of B_j and r some later position in B_j.) For a fixed r and t with $|\mathrm{Dict}(B[r,t])| = 5D\log n$, $\Pr_{H_\ell}[\forall i \in \{r,\dots,t-1\}, H_\ell(B[i,i+1]) \neq 0] \leq \left(1 - \frac{1}{D}\right)^{5D\log n}$ by the $(5D\log n)$-wise independence of H_ℓ. Hence, $\Pr_{H_\ell}[\exists 1 < r < t \leq |B|, |\mathrm{Dict}(B[r,t])| = 5D\log n$ and $\forall i \in \{r,\dots,t-1\}, H_\ell(B[i,i+1]) \neq 0] \leq |B|^2 \left(1 - \frac{1}{D}\right)^{5D\log n} \leq n^2 e^{-5\log n} \leq 1/n^3$. □

Lemma 3.5. *For $B \in \Gamma^*$, $\ell \leq L$, $D_1, D_2, \dots, D_\ell \subseteq \Gamma^2$, $\mathrm{Grammar}(B, (D_1,\dots,D_\ell), \ell)$ outputs a grammar G of size at most $3|B| + 6\sum_i |D_i|$, and runs in time $\widetilde{O}(|B| + \sum_i |D_i|)$.*

Proof. The main loop of the algorithm iterates over all the pairs from D_j. In each iteration we can add a rule of the type $c \to ab$ to G. Hence, the number of such rules in G is at most $|B| + 2\sum_i |D_i|$. Last, we add to G rules for symbols from Σ_r that appear on right hand sides of rules in G. This increases the size of G by at most factor of 3. If C is stored using some efficient data structure such as binary search trees or hash tables, each iteration takes $\widetilde{O}(1)$ time. (We assume that evaluation of $C_j(\cdot)$ takes $O(1)$.) Hence, the total running time is bounded by claimed bound. □

During processing of a string x, there are at most Ln calls to the function Split. (The actual number of calls is $O(n)$ as the strings shrink exponentially but our simple upper bound suffices.) The probability that any one of them would produce a block with dictionary larger than $5D\log n$ is at most Ln/n^3. We can conclude the next corollary which implies the second item of Theorem 3.1.

Corollary 3.6. *For n large enough, on a string x of length at most n, processing the string x produces a sequence of grammars each of size at most $S = 30DL\log n + 6$ with probability at least $1 - 1/n$.*

For the grammars produced by the algorithm to be deterministic, we need that each C_ℓ is one-to-one on $\mathrm{Dict}(B)$ for each block B on which $\mathrm{Compress}(B,\ell)$ is invoked. That will happen with high probability by a standard argument:

Lemma 3.7. *Let $B \in \Gamma^*$ be of length at most n and $\ell \in \{1,\dots,L\}$. Let $C_\ell : \Gamma^2 \to \{\mathtt{c}_i,\ (\ell-1)n^3 < i \leq \ell n^3\}$ be chosen at random from a pair-wise independent family of hash functions. Then with probability at least $1 - |B|/n^2$, C_ℓ is one-to-one on $\mathrm{Dict}(B)$.*

Proof. For two distinct elements from $\mathrm{Dict}(B)$, the probability of a collision for randomly chosen C_ℓ is at most $1/n^3$. By the union bound, the probability that C_ℓ is not one-to-one on $\mathrm{Dict}(B_j)$ is at most $|\mathrm{Dict}(B)|^2/n^3 \leq |B|/n^2$ as $|\mathrm{Dict}(B)| \leq |B| \leq n$. □

During processing of a string x, there are at most Ln calls to the function Compress. For a fixed level $\ell \in \{1,\dots,L\}$, the total size of blocks B for which $\mathrm{Compress}(B,\ell)$ is invoked is at most n. By the previous lemma and the union bound, the probability that during any of those calls $\mathrm{Compress}(B,\ell)$ uses a function C_ℓ that is not one-to-one on $\mathrm{Dict}(B)$ is at most $1/n$. If all the hash functions $C_1, C_2, \dots, C_L$ that are used to compress blocks of x are one-to-one on their respective blocks then the grammars that Grammar produces will be deterministic, and they will evaluate to their respective blocks of x. (We can actually conclude a stronger statement that each C_ℓ will be one-to-one on the union of all blocks at level ℓ with high probability.) We can conclude the next corollary which implies the first item of Theorem 3.1.

Corollary 3.8. *For n large enough, on a string x of length at most n, with probability at least $1 - L/n$, processing the string x produces a sequence of grammars $G_1, G_2, \dots, G_s$ such that $x = \mathrm{eval}(G_1)\cdots\mathrm{eval}(G_s)$.*

At this point we can estimate the running time of the decomposition algorithm. We can let the algorithm fail, and produce some trivial decomposition of x, whenever Split produces a block with dictionary larger than $5D\log n$. If it does not fail, then all grammars are of size at most S which is $\widetilde{O}(k)$. There are at most n of them so time spent in $\mathrm{Grammar}(\dots)$ is bounded by $\widetilde{O}(nk)$. The total time spent in $\mathrm{Compress}(\dots)$ is proportional to the sum of sizes of all non-trivial blocks over all levels of recursion which is $O(nL) = \widetilde{O}(n)$. (A more accurate estimate on the total size of blocks is $O(n)$ since the blocks are shrinking geometrically in each iteration.) This means that the time to execute all calls to Compress is $O(nLR) = \widetilde{O}(n)$. The time spent in $\mathrm{Split}(\dots)$ is dominated by the time needed to evaluate H_ℓ. The number of evaluation points at a given level ℓ is proportional to the total size of all blocks at that level. Since H_ℓ can be evaluated at a single point in time $O(D\log n) = \widetilde{O}(k)$, we get a trivial upper bound $O(nLD\log n) = \widetilde{O}(nk)$ on time spent in Split. Hence, in total the decomposition procedure runs in time $\widetilde{O}(nk)$. (We believe that the total running time can be improved to $\widetilde{O}(n)$ on average. One could argue that in expectation the number of grammars the procedure produces is $\widetilde{O}(n/k)$ as the average block size a string x is decomposed into should be at least $\Omega(D/\log n)$. So we believe that the total running time of calls to Grammar is $\widetilde{O}(n)$. Using multi-point evaluation of $(5D\log n)$-wise independent hash functions we could reduce the time for evaluation of H_ℓ on a given level to $\widetilde{O}(n)$.)

Proposition 3.9. *Given $k \leq n$, the running time of the decomposition algorithm on a string x of length at most n is $\widetilde{O}(nk)$ with probability at least $1 - 1/n$.*

It remains to address the properties of the algorithm run on a pair of strings x and y of edit distance at most k to establish Theorem 3.1. For the pair of strings x and y we fix a *canonical decomposition of x and y* to be a sequence of words $w_0, w_1, \dots, w_k, u_i, \dots, u_k, v_1, \dots, v_k \in \Gamma^*$ such that $x = w_0u_1w_1u_2w_2\cdots u_kw_k$, $y = w_0v_1w_1v_2w_2\cdots v_kw_k$ and $|u_i|, |v_i| \leq 1$ for all i. By the definition of edit distance such a decomposition exists: each pair (u_i, v_i) represents one edit operation, and we fix one such decomposition to be *canonical*. Observe, if we now partition x into blocks $B^x_1, \dots, B^x_s$ so that each B^x_i starts within one of the w_j's, and we partition y into blocks $B^y_1, \dots, B^y_s$ so that each block B^y_i starts at the corresponding location in w_j as B^x_i, then $\mathrm{ED}(x, y) = \sum_i \mathrm{ED}(B^x_i, B^y_i)$.

We need to understand what happens with the decomposition of x and y when we apply the Compress function. Let $x = uwv$ and $x' = \mathrm{Compress}(x, \ell) = u'w'v'$, for some $u, w, v, u'w'v' \in \Gamma^*$. We say that a symbol c in w' *comes from the compression of* w if either it is directly copied from w by Compress, or it is the image $c = C_\ell(ab)$ of a pair of symbols ab where a belongs to w, or $c = \mathtt{r}_{a,r}$ replaced a block a^r where the first symbol of a^r belongs to w. w' *is the compression of* w if it consists precisely of the symbols that come from the compression of w. Furthermore, we say a symbol c in w' *comes weakly from the compression of* w if either it is directly copied from w by Compress, or it is the image $c = C_\ell(ab)$ of a pair of symbols ab where a or b belong to w, or $c = \mathtt{r}_{a,r}$ replaced a block a^r where some symbol of a^r belongs to w. w' *is the weak compression of* w if it consists precisely of the symbols that come weakly from the compression of w. Notice, a weak compression of w might contain and extra symbol at the beginning compared to the compression of w.

The following lemma captures what compression does to the canonical decomposition of x and y. (See Fig. 2 for illustration.)

Lemma 3.10. *Let x and y be strings over Γ, and let $x' = \mathrm{Compress}(x, \ell)$ and $y' = \mathrm{Compress}(y, \ell)$. Let $x = w_0u_1w_1u_2w_2 \cdots u_qw_q$ and $y = w_0v_1w_1v_2w_2 \cdots v_qw_q$ for some strings w_i, u_i and v_i where for $i \in \{1, \dots, q\}$, $|u_i|, |v_i| \le 4R + 24$. Then there are $w'_0, w'_1, \dots, w'_q, u'_1, \dots, u'_q, v'_1, \dots, v'_q \in \Gamma^*$ such that for $i \in \{1, \dots, q\}$, $|u'_i|, |v'_i| \le 4R + 24$, $x' = w'_0u'_1w'_1u'_2w'_2 \cdots u'_qw'_q$ and $y' = w'_0v'_1w'_1v'_2w'_2 \cdots v'_qw'_q$. Moreover, each w'_i is the compression of the same subword of w_i in both x and y.*

For each $x = w_0u_1w_1u_2w_2 \cdots u_qw_q$, $y = w_0v_1w_1v_2w_2 \cdots v_qw_q$ and ℓ we fix one choice of $w'_0, \dots, w'_q$, $u'_0, \dots, u'_q$, $v'_0, \dots, v'_q$ satisfying the lemma. We will refer to it as the *canonical decomposition* of x' and y' induced by the decomposition of x and y as given by the lemma.

Proof. The first stage of Compress replaces maximal blocks of repeated symbols by shortcuts. To simplify our analysis first we will reassign blocks of repeated symbols among neighboring blocks of w_i, u_i and v_i, resp., so each maximal block of symbols in x and y is fully contained in one of the words w_i, u_i or v_i.

For $i = 1, \dots, q-1$ we define words $w^{(1)}_i$ and parameters $a_i, b_i \in \Gamma$ and $k_i, k'_i \in \mathbb{N}$ as follows: If w_i contains at least two distinct symbols let $w_i = a_i^{k_i} w^{(1)}_i b_i^{k'_i}$ so that k_i and k'_i are maximum possible, otherwise $w_i = a_i^{k_i}$ for some a_i and k_i (k_i might be zero), and we set $w^{(1)}_i = \varepsilon$, $b_i = a_i$ and $k'_i = 0$. Let $w_0 = w^{(1)}_0 b_0^{k'_0}$ for maximum possible k'_0 and some symbol b_0. Let $w_q = a_q^{k_q} w^{(1)}_q$ for maximum possible k_q and some symbol a_q. For $i = 1, \dots, q$, we let $u^{(1)}_i = b_{i-1}^{k'_{i-1}} u_i a_i^{k_i}$. Similarly, $v^{(1)}_i = b_{i-1}^{k'_{i-1}} v_i a_i^{k_i}$. Hence, $x = w^{(1)}_0 u^{(1)}_1 w^{(1)}_1 \cdots u^{(1)}_q w^{(1)}_q$ and $y = w^{(1)}_0 v^{(1)}_1 w^{(1)}_1 \cdots v^{(1)}_q w^{(1)}_q$.

Next, if there is a maximal block of symbols a^r contained in $u^{(1)}_s w^{(1)}_s \cdots u^{(1)}_t$ starting in $u^{(1)}_s$ and ending in $u^{(1)}_t$, $s \ne t$, we add all the symbols of the a^r to the end of $u^{(1)}_s$ and remove them from the other $u^{(1)}_i$, $i = s+1, \dots, t$. (Notice, $w^{(1)}_i = \varepsilon$ for $s < i < t$ because of the definition of $w^{(1)}_i$, and $u^{(1)}_i$ will become empty for $s < i < t$.) We do this for all maximal blocks of repeated symbols that span multiple $u^{(1)}_i$. We perform similar moves on $v^{(1)}_i$'s. After all of those moves we denote the resulting subwords by $w^{(2)}_i$, $u^{(2)}_i$, and $v^{(2)}_i$. (Notice, $w^{(2)}_i = w^{(1)}_i$ for all i.) We have: $x = w^{(2)}_0 u^{(2)}_1 w^{(2)}_1 \cdots u^{(2)}_q w^{(2)}_q$ and $y = w^{(2)}_0 v^{(2)}_1 w^{(2)}_1 \cdots v^{(2)}_q w^{(2)}_q$. At this stage, each maximal block of repeated symbols in x or y is contained in one of the subwords $w^{(2)}_i$, $u^{(2)}_i$, and $v^{(2)}_i$.

The first stage of Compress replaces each maximal block a^r, $r \geq 2$, by a sequence $\mathtt{r}_{a,r}\#$, and we apply this procedure on each subword $w_i^{(2)}$, $u_i^{(2)}$, and $v_i^{(2)}$ to obtain corresponding subwords $w_i^{(3)}$, $u_i^{(3)}$, and $v_i^{(3)}$. Observe, for $i = 1, \ldots, q$, $|u_i^{(3)}|, |v_i^{(3)}| \leq 4R + 28$. This is because every u_i is transformed into $u_i^{(3)}$ by appending or prepending possibly empty block of repeated symbols, i.e., $u_i^{(3)} = a^r u_i b^{r'}$ for some a, b, r, r', or removing its content entirely. Each block of repeats is reduced to two symbols so each $u_i^{(3)}$ is longer than the original by at most 4 symbols. Similarly for $v_i^{(3)}$.

Next, coloring function F_{CVL} is used on parts of x and y that are not obtained from repeated symbols; the two symbols replacing each repeated block are colored by 1 and 2, resp. We refer to this as $\{1, 2, 3\}$-coloring. At most R first and last symbols of each $w_i^{(3)}$ might be colored differently in x and y as the color of each symbol depends on the context of at most R symbols on either side of the symbol, and that context might differ in x and y. Hence, only symbols near the border of $w_i^{(3)}$ that are in vicinity of $u_i^{(3)}$'s and $v_i^{(3)}$'s, resp., might get different colors. All the other symbols of $w_i^{(3)}$ are colored the same in both x and y. The coloring is then used to make decisions on which pairs of symbols are compressed into one.

We will let u'_i be the symbols that come from the compression of symbols in $u_i^{(3)}$, the first up-to $R + 2$ symbols of $w_i^{(3)}$, and the last up-to $R+3$ symbols of $w_{i-1}^{(3)}$. Next we specify precisely which symbols of $w_i^{(3)}$ and $w_{i-1}^{(3)}$ are considered to be compressed into symbols belonging to u'_i. For $i = 0, \ldots, q$, if $|w_i^{(3)}| \geq R+3$, let s_i^x be the position of the first symbol in $w_i^{(3)}$ among positions $R + 1, R + 2, R + 3$ which is colored 1 in x by the $\{1, 2, 3\}$-coloring. If $|w_i^{(3)}| < R + 3$, let $s_i^x = 1$. Next, if $|w_i^{(3)}| \geq 2R + 3$ set t_i^x to be the first position from left colored 1 among the symbols of $w_i^{(3)}$ at positions $R+1, R+2, R+3$ counting from right. If $|w_i^{(3)}| < 2R + 3$, set t_i^x to be equal to s_i^x. For $i = 0$, if $|w_0^{(3)}| \geq R + 3$ then redefine $s_0^x = 1$. For $i = q$, redefine $t_q^x = |w_q^{(3)}| + 1$ and if $|w_q^{(3)}| < R + 3$ then redefine s_q^x to t_q^x. Similarly, define s_i^y and t_i^y based on the $\{1, 2, 3\}$-coloring of y.

Notice, $s_i^x \neq t_i^x$ iff $s_i^y \neq t_i^y$. Furthermore, if $s_i^x \neq t_i^x$ then either $i \in \{q, 0\}$ or $|w_i^{(3)}| \geq 2R + 3$ so $s_i^x = s_i^y$ and $t_i^x = t_i^y$ as the symbols R-away from either end of $w_i^{(3)}$ are colored the same in x and y. We let u'_i to be the compression of $w_{i-1}^{(3)}[t_{i-1}^x, |w_{i-1}^{(3)}|] \cdot u_i^{(3)} \cdot w_i^{(3)}[1, s_i^x)$ and similarly, v'_i to be the compression of $w_{i-1}^{(3)}[t_{i-1}^y, |w_{i-1}^{(3)}|] \cdot v_i^{(3)} \cdot w_i^{(3)}[1, s_i^y)$. We let w'_i be the compression of $w_i^{(3)}[s_i^y, t_i^y)$.

Hence, u'_i comes from the compression of at most $|u_i^{(3)}| + 2R + 5 \leq 6R + 33$ symbols. Since each symbol after a symbol colored 1 is *removed* by the compression, and each consecutive triple of symbols contains at least one symbol colored by 1, the at most $6R + 27$ symbols are compressed into at most $(6R + 33) \cdot 2/3 + 2 = 4R + 24$ symbols. So u'_i is of length at most $4R + 24$. Similarly for v'_i. □

The following generalization of the previous lemma will be useful to design a rolling sketch. It considers situation where x and y are prefixed by some strings u and v, resp., that we want to ignore from the analysis. The proof of the lemma is a straightforward modification of the above proof.

Lemma 3.11. *Let $x, y, u, v \in \Gamma^*$, and let $u'x' = \mathrm{Compress}(ux, \ell)$ and $v'y' = \mathrm{Compress}(vy, \ell)$, where x' is the weak compression of x, and y' is the weak compression of y. Let $x = u_0w_0u_1w_1u_2w_2 \cdots u_qw_q$ and $y = v_0w_0v_1w_1v_2w_2 \cdots v_qw_q$ for some strings w_i, u_i and v_i where for $i \in \{0, \ldots, q\}$, $|u_i|, |v_i| \leq 4R + 24$. Then there are $w'_0, w'_1, \ldots, w'_q, u'_0, u'_1, \ldots, u'_q, v'_0, v'_1, \ldots, v'_q \in \Gamma^*$ such that for $i \in \{0, \ldots, q\}$, $|u'_i|, |v'_i| \leq 4R + 24$, $x' = u'_0w'_0u'_1w'_1u'_2w'_2 \cdots u'_qw'_q$ and $y' = v'_0w'_0v'_1w'_1v'_2w'_2 \cdots v'_qw'_q$. Moreover, each w'_i is the compression of the same subword of w_i in both x and y.*

Let $x \in \Sigma^*$. Let $H_0, H_1, \ldots, H_L, C_1, C_2, \ldots, C_L$ be chosen. We define inductively the *trace* of the algorithm on x at level $\ell \geq 0$ to consist of sequences $B^x(\ell,1),\dots,B^x(\ell,s^x_\ell) \in \Gamma^*$, of auxiliary sequences $A^x(\ell,1),\dots,A^x(\ell,s^x_\ell) \in \Gamma^*$ and $t^x_{\ell,1},\dots,t^x_{\ell,s^x_\ell+1} \in \mathbb{N}$. Their meaning is: $B^x(\ell,i)$ is compressed into $A^x(\ell,i)$ and that is split into blocks $B^x(\ell+1,j)$ for $t^x_{\ell+1,i} \leq j < t^x_{\ell+1,i+1}$. (See Fig. 1 for illustration.)[3]

Set

$$B^x(0,1),\dots,B^x(0,s^x_0) = \mathrm{Split}(x,0).$$

For $\ell = 1,\dots,L$ we define $B^x(\ell,1),\dots,B^x(\ell,s^x_\ell)$ inductively. Set $t^x_{\ell,1} = 1$. For $i = 1,\dots,s^x_{\ell-1}$, if $|B^x(\ell-1,i)| \geq 2$, then

$$A^x(\ell-1,i) = \mathrm{Compress}(B^x(\ell-1,i),\ell),$$

and for $(B_0,B_1,\dots,B_s) = \mathrm{Split}(A^x(\ell-1,i),\ell)$ set

$$B^x(\ell,t^x_{\ell,i}) = B_0,\ \ B^x(\ell,t^x_{\ell,i}+1) = B_1,\ \ \dots,\ \ B^x(\ell,t^x_{\ell,i}+s) = B_s$$

and $t^x_{\ell,i+1} = t^x_{\ell,i} + s + 1$. If $|B^x(\ell-1,i)| < 3$, then set $B^x(\ell,t^x_{\ell,i})$ and $A^x(\ell-1,i)$ to $B^x(\ell-1,i)$, and $t^x_{\ell,i+1} = t^x_{\ell,i} + 1$. For $j = s^x_{\ell-1}$, set $s^x_\ell = t^x_{\ell,j+1}$.

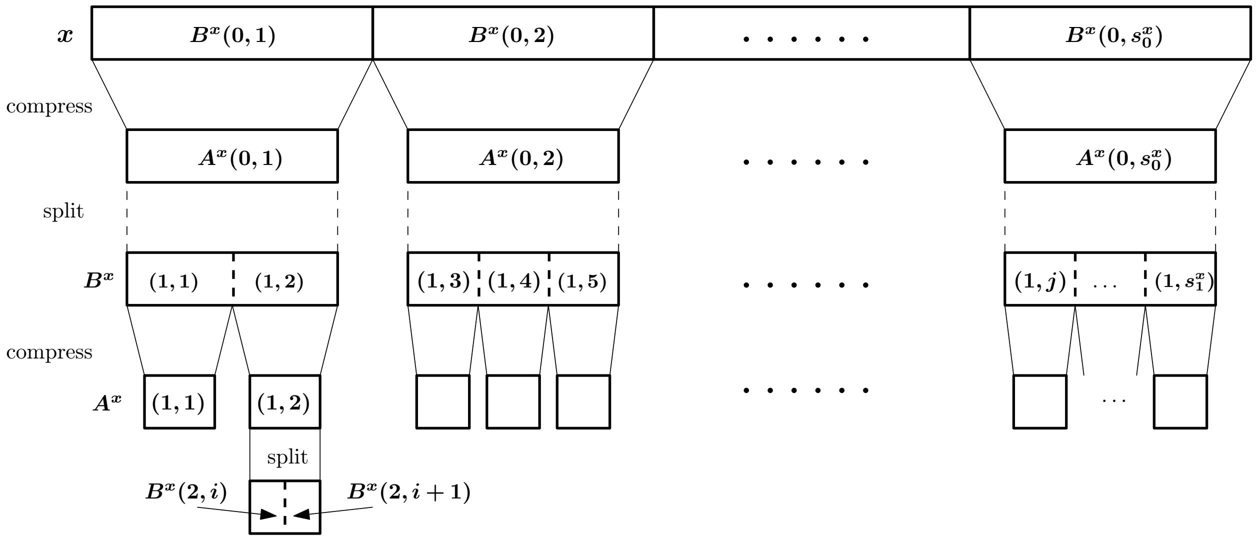

Figure 1: The hierachical decomposition of x.

Furthermore, for x and $y \in \Sigma^*$, $\ell, i \geq 0$, define a canonical decomposition of blocks $A^x(\ell,i)$, $B^x(\ell,i)$, $A^y(\ell,i)$, $B^y(\ell,i)$ inductively as follows. Let $A^x(-1,1) = x$ and $A^y(-1,1) = y$. Let $t^x_{-1,1} = 1, t^x_{-1,2} = 2$, $s^x_{-1} = 1$, $t^y_{-1,1} = 1$, $t^y_{-1,2} = 2$, and $s^y_{-1} = 1$. Let

$$A^x(-1,1) = w_0u_1w_1u_2w_2\cdots u_kw_k \ \ \& \ \ A^y(-1,1) = w_0v_1w_1v_2w_2\cdots v_kw_k$$

be the canonical decomposition of the pair x and y.

For $\ell \geq 0$ and $j \in \{1,\dots,s^x_\ell\}$, let i be such that $t^x_{\ell-1,i} \leq j < t^x_{\ell-1,i+1}$ and $m = j - t^x_{\ell-1,i}$. Then $B^x(\ell,j)$ is the m-th block of $\mathrm{Split}(A^x(\ell-1,i),\ell)$. If the decomposition of $A^x(\ell-1,i)$ is defined and is equal to $w_0u_1w_1u_2w_2\cdots u_qw_q$, for some $u_i, w_i \in \Gamma^*$, then the decomposition of $B^x(\ell,j)$ is the restriction of the decomposition of $A^x(\ell-1,i)$ to symbols of the m-th block of $\mathrm{Split}(A^x(\ell-1,i),\ell)$. Otherwise the decomposition of $B^x(\ell,j)$ is undefined. Similarly for $B^y(\ell,j)$. (See Fig. 2.)

[3]To avoid double and triple indexes we use our notation $B^x(\ell,i)$ and $A^x(\ell,i)$ instead of the usual $B^x_{\ell,i}$ and $A^x_{\ell,i}$.

For $\ell \geq 0$ and $j \in \{1, \ldots, s^x_\ell\}$, if $B^x(\ell, j)$ and $B^y(\ell, j)$ have defined decompositions $B^x(\ell, j) = w_0 u_1 w_1 u_2 w_2 \cdots u_q w_q$ and $B^y(\ell, j) = w_0 v_1 w_1 v_2 w_2 \cdots v_q w_q$ for some $u_i, v_i, w_i \in \Gamma^*$, then we let $A^x(\ell, j) = w'_0 u'_1 w'_1 u'_2 \cdots w'_q$ and $A^y(\ell, j) = w'_0 v'_1 w'_1 v'_2 \cdots w'_q$ be their canonical decomposition induced by $B^x(\ell, j)$ and $B^x(\ell, j)$ as given by Lemma 3.10.

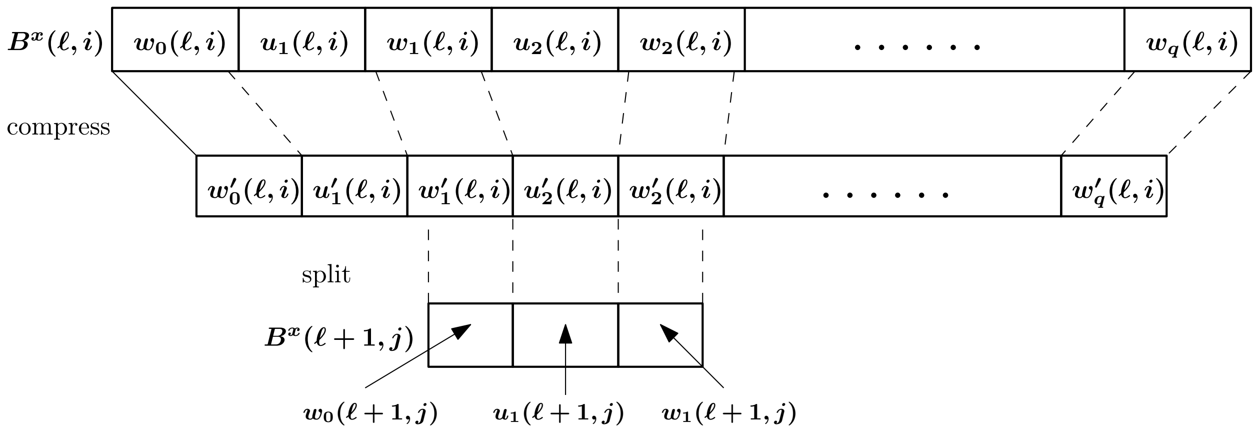

Figure 2: Decomposition of $B^x(\ell, i)$ after compression and split.

To conclude item 3 of Theorem 3.1 we want to argue that x and y are recursively split into sub-blocks that respect their canonical decomposition. So we want all splits of blocks to occur in matching parts of x and y. For $A^x(\ell - 1, i)$ with canonical decomposition $w_0 u_1 w_1 u_2 w_2 \cdots u_q w_q$ we say that $\text{Split}(A^x(\ell - 1, i), \ell)$ makes *undesirable split* if it starts a new block at a position j that either belongs to one of the $u_1, u_2, \ldots, u_q$ or is the first or last symbol of one of the $w_0, w_1, \ldots, w_q$. Recall, $\text{Split}(A^x(\ell - 1, i), \ell)$ starts a new block at each position j such that $H_\ell(A^x(\ell - 1, i)[j, j + 1]) = 0$. Since H_ℓ is chosen at random a given position starts a new block with probability $1/D$.

Similarly, for $A^y(\ell - 1, i)$ with canonical decomposition $w'_0 v_1 w'_1 v_2 \cdots v_{q'} w'_{q'}$ we say that $\text{Split}(A^y(\ell - 1, i), \ell)$ makes *undesirable split* if it starts a new block at a position j that either belongs to one of the $v_1, v_2, \ldots, v_{q'}$ or is the first or last symbol of one of the $w'_0, w'_1, \ldots, w'_{q'}$. If $A^x(\ell - 1, i)$ and $A^y(\ell - 1, i)$ have *matching* canonical decomposition (that is $q = q'$ and each $w_j = w'_j$) and both $\text{Split}(A^x(\ell - 1, i), \ell)$ and $\text{Split}(A^y(\ell - 1, i), \ell)$ make no undesirable split then $A^x(\ell - 1, i)$ and $A^y(\ell - 1, i)$ are split in the same number of blocks with matching canonical decomposition as they are split at the same positions in the corresponding w_j's.

For given $\ell \in \{0, \ldots, L\}$, if no undesirable split happens during $\text{Split}(A^x(\ell' - 1, i), \ell')$ and $\text{Split}(A^y(\ell' - 1, i), \ell')$, for any $\ell' < \ell$ and i, then for each $\ell' < \ell$, the number of blocks $B^x(\ell', i)$ and $B^y(\ell', i)$ will be the same, i.e., $s^x_{\ell'} = s^y_{\ell'}$, and blocks $B^x(\ell', i)$ and $B^y(\ell', i)$ will have matching canonical decomposition. The total number of u_j's in canonical decomposition of all $B^x(\ell', i)$, $i = 1, \ldots, t^x_{\ell'}$, will be at most k, and similarly for v_j's. Thus, there will be at most $(4R + 24 + 2)k + 2$ positions where an undesirable split can happen in $\text{Split}(A^x(\ell - 1, i), \ell)$ for any i. Similarly, there are at most $(4R + 26)k + 2$ positions where an undesirable split can happen in $\text{Split}(A^y(\ell - 1, i), \ell)$. By union bound, the probability that an undesirable split happens in some $\text{Split}(A^y(\ell - 1, i), \ell)$ or $\text{Split}(A^y(\ell - 1, i), \ell)$, for some ℓ and i, is at most $2(4R + 28)k(L + 1)/D \leq 11Rk(L + 1)/D \leq 1/10$.

Thus, if no undesirable split happens there are at most k indices i for which the canonical decomposition of $B^x(\ell, i)$ contains some u_j. All other blocks $B^x(\ell, i)$ have a canonical decomposition consisting of a single block w_0, for various w_0 depending on ℓ and i. Similarly, the canonical decomposition of $B^y(\ell, i)$ contains v_j if and only if $B^x(\ell, i)$ contains u_j. Blocks $B^y(\ell, i)$ that do not contain v_j are identical to $B^x(\ell, i)$ so they have the same grammar.

Hence, if no undesirable split happens, item 3 of Theorem 3.1 will be satisfied.

The following theorem generalizes item 3 of Theorem 3.1 and it will be useful to construct the rolling sketch in Section 4.

Theorem 3.12. *Let $u, v, x, y \in \Sigma^*$ be strings such that $|ux|, |vy| \le n$ and $\mathrm{ED}(x, y) \le k$. Let $G^x_1, \dots, G^x_s$ and $G^y_1, \dots, G^y_{s'}$ be the sequence of grammars output by the decomposition algorithm on input ux and vy respectively, using the same choice of random functions $C_1, \dots, C_L$ and $H_0, \dots, H_L$. With probability at least $1 - 1/5$ the following is true: There exist integers r, r', t, t' such that $s - t = s' - t'$,*

$$x = \mathrm{eval}(G^x_t)[r, \dots] \cdot \mathrm{eval}(G^x_{t+1}) \cdots \mathrm{eval}(G^x_s) \quad \& \quad y = \mathrm{eval}(G^y_{t'})[r', \dots] \cdot \mathrm{eval}(G^y_{t'+1}) \cdots \mathrm{eval}(G^y_{s'}),$$

and

$$\mathrm{ED}(x, y) = \mathrm{ED}(\mathrm{eval}(G^x_t)[r, \dots], \mathrm{eval}(G^y_{t'})[r', \dots]) + \sum_{i>0} \mathrm{ED}(\mathrm{eval}(G^x_{t+i}), \mathrm{eval}(G^y_{t'+i})).$$

Its proof is a minor modification of the proof above. We start with the canonical decomposition of $x = w_0 u_1 w_1 \cdots u_k w_k$ and $y = w_0 v_1 w_1 \cdots v_k w_k$, form the decomposition $ux = uu_0 w_0 u_1 w_1 \cdots u_k w_k$ and $vy = vv_0 w_0 v_1 w_1 \cdots v_k w_k$ where $u_0 = v_0 = \varepsilon$, and follow the compression and split procedures. We want to argue that during each split operation, all splits occur either in w_j's and are the same on ux and vy, or they occur in u or v where we do not care for them. Again we define a split to be *undesirable* if it starts a new block at a position j that belongs to one of the $u_0, u_1, \dots, u_k, v_0, v_1, \dots, v_k$ or it is the position of the first or last symbol of $w_0, w_1, \dots$ or w_k. Inductively we maintain that whenever a block $B^{ux}(\ell, i)$ contains a descendant of the compression of u_j, its corresponding block $B^{vy}(\ell, i')$ contains a descendant of the compression of v_j. (Here, the correspondence is counting from the highest index i to the lowest and similarly for i', so $B^{ux}(\ell, i)$ corresponds to $B^{vy}(\ell, i')$ if $i - i' = s^{ux}_\ell - s^{vy}_\ell$.) If the blocks contain a descendant of u_0 and v_0, resp., then we apply Lemma 3.11 to construct a descendant decomposition after their compression. For all other blocks that contain some w_j, u_j or v_j we use Lemma 3.10 to construct its descendant decomposition. We do not care for decomposition of blocks $B^{ux}(\ell, i)$ that are descendants of u but do not contain u_0, and similarly we do not care for decomposition of blocks $B^{vy}(\ell, i)$ that are descendants of v but do not contain v_0. (They might be decomposed arbitrarily so the number of blocks that are descendants of u might differ from the number of blocks that are descendants of v.) Inductively, there are at most $2(4R+28)(k+1)$ positions where an undesirable split can happen in blocks $B^{ux}(\ell, i)$ and $B^{vy}(\ell, i)$ for given level ℓ. In total there are at most $2(4R + 28)(k + 1)(L + 1)$ positions where an undesirable split can happen. Thus, the probability of making an undesirable split during a run of the algorithm is bounded by $2(4R + 28)(k + 1)(L + 1)/D \le 22Rk(L + 1)/D \le 1/5$. If no undesirable split ever happens then the symbols that are weak compression of symbols from x and y are contained withing the corresponding blocks $B^{ux}(\ell, i)$ and $B^{yv}(\ell, i')$. For the blocks $B^{ux}(\ell, i)$ and $B^{vy}(\ell, i')$ that contain descendants of u_0 and v_0 it is fine if their prefixes that descend from u and v, resp., which are to the left of the descendants of u_0 and v_0, are split differently in $B^{ux}(\ell, i)$ and $B^{vy}(\ell, i')$. This does not affect the correspondence between blocks $B^{ux}(\ell, i)$ and $B^{vy}(\ell, i')$ that weakly come from x and y. This concludes the proof of Theorem 3.12.

3.3 Encoding a grammar

We will set a parameter $N \geq n^3$ to be a suitable integer: Let $F_{\mathrm{KR}} : \{0,1\}^* \to \{1,\ldots,N\}$ be a hash function picked at random, such as Karp-Rabin fingerprint [KR87], so for any two strings $u, v \in \{0,1\}^*$, if $u \neq v$ then $\Pr_{F_{\mathrm{KR}}}[F_{\mathrm{KR}}(u) = F_{\mathrm{KR}}(v)] \leq (|u| + |v|)/N$.

Set $M = 3S \cdot \lceil 1 + \log|\Gamma| \rceil$. We will encode a grammar G over Γ of length at most S given by our decomposition algorithm by a string $\mathrm{Enc}(G)$ over alphabet $\{1,\ldots,2N\}$ of length M. The encoding is obtained as follows: First, order the rules of the grammar G lexicographically. Then encode the rules in binary one by one using $3 \cdot \lceil 1 + \log|\Gamma| \rceil$ bits for each rule. (The extra bit allows to mark unused symbols.) This gives a binary string of length at most M, which we pad by zeros to the length precisely M. We call the resulting binary string $\mathrm{Bin}(G)$. Compute $h_G = F_{\mathrm{KR}}(\mathrm{Bin}(G))$. We replace each 0 in $\mathrm{Bin}(G)$ by h_G, and each 1 in $\mathrm{Bin}(G)$ by $N + h_G$ to obtain the string $\mathrm{Enc}(G)$. Clearly, $\mathrm{Enc}(G)$ is a string over alphabet $\{1,\ldots,2N\}$ of length exactly M. The encoding can be computed in time $O(M)$. For completeness, we encode any grammar G of length more than S or that uses rules with more than two symbols on the right as $\mathrm{Enc}(G) = 1^M$.

By the property of F_{KR} the following holds.

Lemma 3.13. *Let G, G' be two grammars of size at most S output by our decomposition algorithm. Let F_{KR} be chosen at random.*

1. $\mathrm{Enc}(G) \in \{1,\ldots,2N\}^M$.
2. *If $G = G'$ then $\mathrm{Enc}(G) = \mathrm{Enc}(G')$.*
3. *If $G \neq G'$ then $\mathrm{Enc}(G) = \mathrm{Enc}(G')$ with probability at most $2M/N$.*
4. *If $\mathrm{Enc}(G) \neq \mathrm{Enc}(G')$ then $\mathrm{Ham}(\mathrm{Enc}(G), \mathrm{Enc}(G')) = M$, that is they differ in every symbol.*

3.4 Edit distance sketch

Let n and $k \leq n$ be two parameters, and $p \geq 2N + 1$ be a prime such that $p \geq (nM)^3$. For a string $x \in \Sigma^*$ of length at most n, we compute its sketch by running first the decomposition algorithm of Theorem 3.1 to get grammars $G_1, G_2, \ldots, G_s$. Encode each grammar G_i by encoding $\mathrm{Enc}(G_i)$ from Section 3.3 using the same F_{KR} picked at random. Concatenate the encoding to get a string $w = \mathrm{Enc}(G_1) \cdot \mathrm{Enc}(G_2) \cdots \mathrm{Enc}(G_s)$. Calculate the Hamming sketch $\mathrm{sk}^{\mathrm{Ham}}_{n',m',p}(w)$ on w for strings of length $n' = nM$ and Hamming distance at most $k' = kM$ from Section 2.2. Set the sketch $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x) = \mathrm{sk}^{\mathrm{Ham}}_{n',k',p}(w)$. The calculation of $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x)$ can be done in time $\widetilde{O}(nk)$ as the number of grammars is at most n and each grammar requires $\widetilde{O}(k)$ time to be encoded into binary. The Hamming sketch can be constructed in time $\widetilde{O}(nk)$. (We believe that on average we expect only $\widetilde{O}(n/k)$ grammars to be produced for a given string x so the actual running time should be $\widetilde{O}(n)$ on average.)

Theorem 3.14. *Let $x, y \in \Sigma^*$ be strings of length at most n such that $\mathrm{ED}(x,y) \leq k$. Let $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x)$ and $\mathrm{sk}^{\mathrm{ED}}_{n,k}(y)$ be obtained using the same randomness for the decomposition algorithm and the same choice of F_{KR}. With probability at least $2/3$, we can calculate $\mathrm{ED}(x,y)$ from $\mathrm{sk}^{\mathrm{ED}}_{n,k}(x)$ and $\mathrm{sk}^{\mathrm{ED}}_{n,k}(y)$.*

Assume that the output of the decomposition algorithm on x and y satisfies all the conclusions of Theorem 3.12. In particular, for x we get $\mathrm{eval}(G^x_1) \cdot \mathrm{eval}(G^x_2) \cdots \mathrm{eval}(G^x_s)$ and for y we get $\mathrm{eval}(G^y_1) \cdots \mathrm{eval}(G^y_s)$, for some $s \leq n$, each of the grammars is of size at most S, $\mathrm{ED}(x,y) = \sum_i \mathrm{ED}(\mathrm{eval}(G^x_i), \mathrm{eval}(G^y_i))$, and the number of pairs G_i^x and G_i^y where $G_i^x \neq G_i^y$ is at most k. Assume that F_{KR} is chosen so that $\mathrm{Enc}(G_i^x) \neq \mathrm{Enc}(G_i^y)$ for each of the pairs where G_i^x and G_i^y differ.

In order to determine $\mathrm{ED}(x, y)$, we recover the (Hamming) mismatch information between $\mathrm{Enc}(G_1^x) \cdot \mathrm{Enc}(G_2^x) \cdots \mathrm{Enc}(G_s^x)$ and $\mathrm{Enc}(G_1^y) \cdot \mathrm{Enc}(G_2^y) \cdots \mathrm{Enc}(G_s^y)$ from $\mathrm{sk}_{n,k}^{\mathrm{ED}}(x)$ and $\mathrm{sk}_{n,k}^{\mathrm{ED}}(y)$. That gives grammars G_i^x and G_i^y, for all i where $G_i^x \neq G_i^y$. (Whenever the two grammars differ, their encoding differ in every symbol by Lemma 3.13 so we can recover them from the Hamming mismatch information.) Calculating the edit distance of each of the pair of differing grammars using the algorithm from Proposition 2.1 we recover $\mathrm{ED}(x, y)$ as the sum of their edit distances.

The sum is correct unless some of the assumptions fail: The probability that the grammar decomposition fails (does not have properties from Theorem 3.1) for the pair x and y is at most $1/5$ for n large enough. The probability that the choice of F_{KR} fails (two distinct grammars have the same encoding) is at most $2kM/N < 1/n$ by the choice of N. The probability that the Hamming distance sketch fails to recover the mismatch information between all the grammars is at most $1/n$. So in total, the probability that the output of the algorithm is incorrect is at most $1/3$.

The running time of the comparison algorithm is $\widetilde{O}(k^2)$: The Hamming mismatch information can be recovered in time $\widetilde{O}(kM) = \widetilde{O}(k^2)$ (Proposition 2.2), then we build the $\leq k$ mismatched grammars in time $\widetilde{O}(k^2)$, and run the edit distance computation on the pairs of grammars in time $\sum_{i<k} \widetilde{O}(k + k_i^2) \leq \widetilde{O}(k^2)$, where k_i is the edit distance of the i-th pair of mismatched grammars. (We interrupt the edit distance computation if it takes more time than $\widetilde{O}(k^2)$ which would indicate $\mathrm{ED}(x, y) > k$.)

To decide whether $\mathrm{ED}(x, y) > k$ we note that on input x and y, the Hamming sketch either outputs the correct mismatched places if their number is $\leq k'$ or it outputs ∞ if there are more mismatches than that or the sequences sketched by the Hamming sketch are of different length. (We assume that the Hamming sketch knows the number of symbols it is sketching.) In the ∞-case we know that there are more than k different pairs of grammars or the decomposition of x and y failed, and we can report $\mathrm{ED}(x, y) > k$. In the other case we try to calculate the edit distance of the differing pairs of grammars. If we spend more than $\widetilde{O}(k^2)$ time on it or we get a number larger than k then we report $\mathrm{ED}(x, y) > k$. This correctly decides whether $\mathrm{ED}(x, y) > k$ with probability at least $2/3$.

To prove Theorem 1.2 we build a more robust sketch by taking $c \log n$ independent copies of the sketch $\mathrm{sk}_{n,k}^{\mathrm{ED}}$. To calculate the edit distance of two sketched strings we run the edit distance calculation on each of the corresponding pairs of copies, and output the majority answer. A usual application of Chernoff bound shows that the probability of correct answer is at least $1 - 1/n$ for suitable constant $c > 0$.

4 Rolling sketch for edit distance

In this section we will construct the rolling sketch of Theorem 1.3. We will use two claims that will be proved in Section 4.1. The first one addresses how much a compression of a string w might change depending on what is appended to it.

Lemma 4.1. *Let $\ell \in \{0, \ldots, L\}$ and $v, u, w \in \Gamma^*$. Let $w'u' = \mathrm{Compress}(wu, \ell)$ and let $w''v' = \mathrm{Compress}(wv, \ell)$, where w' is the compression of w when compressing wu and w'' is the compression of w when compressing wv. Let $t = |w'| - 3(R+1)$ or $t = |w'u'| - |u| - 3(R+1)$. Then $w'[1, t] = w''[1, t]$.*

The next lemma addresses how much the overall decomposition of a string x might change if we append a suffix z to it.

Lemma 4.2. *Let $x, z \in \Sigma^*$, $|xz| \leq n$. Let $H_0, \dots, H_L, C_1, \dots, C_L$ be given. Let $G^x_1, G^x_2, \dots, G^x_s$ be the output of the decomposition algorithm on input x, and $G^{xz}_1, G^{xz}_2, \dots, G^{xz}_{s'}$ be the output of the decomposition algorithm on input xz using the given hash functions. Let $T = L(3R+6)$.*

1. *$G^x_i = G^{xz}_i$ for all $i = 1 \dots, s-T$.*
2. *$|x| \leq \sum_{i=1}^{\min(s+T,s')} |\mathrm{eval}(G^{xz}_i)|$.*

The second part says that if x is decomposed into s grammars by itself, then it can be recovered from the first $s+T$ grammars for xz. Hence, appending extra symbols to x cannot increase the number of grammars that cover x by more than T.

Let $m \geq k$ and $n \geq 10m^3$ be integers. A rolling sketch for a string obtained by up-to m insertions (to the right end) and m deletions (from the left end) from an empty word consists of three data structures: *insertion buffer*, *deletion buffer* and a Hamming distance sketch $\mathrm{sk}^{\mathrm{Ham}}_{n',k',p}$, where $k' = (4T+1)(k+2)M$, $n' = nM$ and $p \geq n'^3$ is a chosen prime.

The insertion buffer maintains a buffer of *committed grammars* $G_{s-4T+1}, G_{s-4T+2}, \dots, G_s$ and a buffer of *active grammars* $G^i_1, \dots, G^i_t$, $t \leq T$. The deletion buffer is similar, it maintains a buffer of *committed grammars* $G_{r-4T+1}, G_{r-4T+2}, \dots, G_r$ and a buffer of *active grammars* $G^d_1, \dots, G^d_{t'}$, $t' \leq T$. The Hamming sketch is a sketch of grammars $G_{r-2T+1}, G_{r-2T+2}, \dots, G_{s-2T}$, each encoded as a string of length M over the alphabet $\{1, \dots, 2N\}$.

In addition to that, the sketch keeps track of the current value of r and s, and remembers a collection of pair-wise independent hash functions $C_1, \dots, C_L$, a collection of $(5D \log n)$-wise independent hash functions $H_0, \dots, H_L$, and randomness for Karp-Rabin fingerprint to compute binary encoding of grammars. The hash functions and the randomness of Karp-Rabin fingerprint are chosen at random when creating the sketch for empty string. This extra information requires $\widetilde{O}(k)$ bits to specify.

Initially, the committed grammars in the insertion and deletion buffers are all treated as empty sets, there are no active grammars in the insertion or deletion buffers so $t = t' = 0$ and $s = r = 0$.

For $u, x \in \Sigma^*$, if in total a string ux was inserted into the sketch then $G_1, \dots, G_s, G^i_1, \dots, G^i_t$ represents ux, that is ux is the concatenation of the evaluation of the grammars. If in total the string u was deleted from the sketch, then $G_1, \dots, G_r, G^d_1, \dots, G^d_{t^d}$ represents u. (See Fig. 3 for an illustration.)

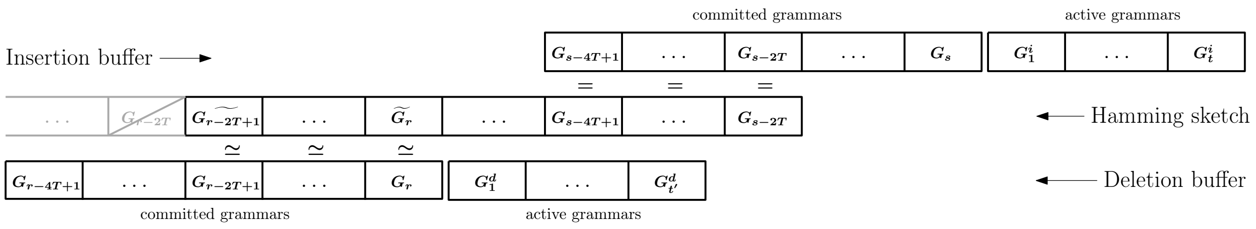

Figure 3: Rolling sketch.

Appending a symbol. When we append additional symbol a to the sketch we modify input buffers as follows: We update the active grammars $G^i_1, \dots, G^i_t$ by appending a as explained further below. Say the update produces grammars $G'^i_1, \dots, G'^i_{t'}$. If $t' \leq T$ then the produced grammars will become the active grammars, and no more changes are done to the sketch. Otherwise we commit the first $t'-T$ grammars $G'^i_1, \dots, G'^i_{t'-T}$ one-by-one into the committed buffer as grammars $G_{s+1}, \cdots, G_{s+t'-T}$ and we keep the remaining grammars as the active grammars.

Committing a grammar G_{s+1} into the committed buffer will trigger addition of G_{s-2T+1} into the Hamming sketch at the end of the represented sequence of grammars (if $s-2T+1 > 0$), and removing the grammar G_{s-4T+1} from the committed buffer. For insertion into the Hamming sketch, the grammar G_{s-2T+1} is encoded into binary as in Section 3.3 and then the binary string is encoded using the Karp-Rabin fingerprint F_{KR} of *all* the grammars $G_{s-4T+1}, \ldots, G_{s+1}$, instead of only the grammar G_{s-2T+1}. (Thus, a change in any of the neighboring grammars will trigger a recovery of also the grammar G_{s-2T+1} when calculating a mismatch information from the Hamming sketch.) We repeat this process for each grammar being committed.

By the second part of Lemma 4.2 $t' \leq t + T \leq 2T$ so we will commit at most $T = \widetilde{O}(1)$ grammars. It takes time $O(MT) = \widetilde{O}(k)$ to prepare the binary encoding of each of the committed grammars, and $\widetilde{O}(k^2)$ to insert it into the Hamming sketch. The update of the active grammars takes $\widetilde{O}(k)$ time as described below. So in total this step takes $\widetilde{O}(k^2)$ time.

Removing a symbol. Deletion buffer works in manner similar to insertion buffer, we add the removed symbol a to the active grammars, but when committing the grammar G_{r+1}, we use F_{KR}-fingerprint of all the grammars $G_{r-4T+1}, \ldots, G_{r+1}$ to encode grammar G_{r-2T+1} which is then *removed* from the beginning of the sequence of grammars represented by the Hamming sketch (if $r - 2T + 1 > 0$), i.e., we update the Hamming sketch to reflect this removal. Similarly to appending a symbol, this step takes time $\widetilde{O}(k^2)$.

Active grammar update. The update of active grammars $G^i_1, \ldots, G^i_t$ when appending a is done as follows. $G_1, \ldots, G_s, G^i_1, \ldots, G^i_t$ represents ux so we need to calculate the grammars for uxa. We claim that only the active grammars might change: At some point, G_s became committed so at that time there was T active grammars following it. If at that point the grammars together represented a string z, by appending more symbols to z we cannot change grammars $G_1, G_1, \ldots, G_s$ according to the first part of Lemma 4.2. So appending a to ux will affect only the active grammars.

From the analysis in the proof of Lemma 4.2 it follows that for $\ell \in \{0, \ldots, 1\}$ if $B^{ux}(\ell, 1), \ldots, B^{ux}(\ell, s^{xy}_\ell)$ is the trace of the decomposition algorithm on ux at level ℓ, and $B^{uxa}(\ell, 1), \ldots, B^{uxa}(\ell, s^{xya}_\ell)$ is the trace on uxa, then their difference spans at most $\ell(3R+6)$ last symbols of $B^{ux}(\ell, 1) \cdots B^{ux}(\ell, s^{xy}_\ell)$.

So instead of decompressing the active grammars completely, adding a and recompressing them back, we only decompress the necessary part of each trace $B^{ux}(\ell, 1) \cdots B^{ux}(\ell, s^{xy}_\ell)$. Let $\# \to v_i$ be the starting rule of the active grammar G_i. Starting from the string $v_1 \cdot v_2 \cdots v_t$, for each $\ell = L, \ldots, 1$, we iteratively rewrite all level-ℓ symbols in the string using the appropriate grammars while only maintaining at most T last symbols of the resulting string. (Care has to be taken to maintain information about any sequence a^r stretching from those T last symbols to the left.)

We add a to the resulting string and re-apply compress and split procedures for levels $0, 1, \ldots, \ell - 1$ to recompress only the part of the trace affected by modifications. As we perform the compression of symbols we maintain a set G of all grammar rules needed for decompression. (We initialize G with the union of all rules from the active grammars $G^i_1, \ldots, G^i_t$ minus the starting rules, and we iteratively add new rules coming from the recompression.) For the recompression we need to know the context of up-to $R + 1$ symbols preceding the modified part of the trace. On the other hand, the modification can affect the recompression of up-to $R + 1$ symbols to the left from the left-most modified symbol in the trace. Those $R + 1$ symbols all happen to be within the decompressed suffix of the trace of size at most T.

Eventually, we get a new level-L trace $B^{uxa}(L, s^{xya}_L - t' + 1), \ldots, B^{uxa}(L, s^{xya}_L)$, for some t'. Each new grammar G'^i_j is obtained by taking the grammar $G \cup \{\# \to B^{uxa}(L, s^{xya}_L - t' + j)\}$ and removing from it all useless rules. This can be done in time $O(|G|)$. (See Section 2.1).

Overall the update of active grammars on insertion of a single symbol will require $O(LT) = \widetilde{O}(1)$ evaluations of split hash functions $H_0, \ldots, H_L$, $O(LT) = \widetilde{O}(1)$ evaluations of compress hash functions

$C_1, \ldots, C_L$, and $O(T(LT + \sum_{j=1}^{t} |G_j^i|))$ time to produce the new grammars. As the total size of the grammars is $\widetilde{O}(k)$ and the time to evaluate H_ℓ at a single point is also $\widetilde{O}(k)$, the overall time for the update of active grammars is $\widetilde{O}(k)$. We provide a more detailed description of the update procedure in Section 5.

Edit distance evaluation. Consider strings x and y of length at most m and edit distance at most k. Consider the rolling sketch $\mathrm{sk}_{m,k}^{\mathrm{Rolling}}(x)$ for x obtained by inserting symbols ux and removing symbols u, for some $u \in \Sigma^*$ where $|ux| \leq m$. Consider also the rolling sketch for y obtained by inserting symbols vy and removing symbols v, for some $v \in \Sigma^*$ where $|vy| \leq m$. Both sketches should use the same randomness that is to start from the same sketch for empty string.

The rolling sketch for x consists of the insertion buffer with committed grammars $G^x_{s^x-4T+1}, G^x_{s^x-4T+2}, \ldots, G^x_{s^x}$ and with active grammars $G_1^{ix}, \ldots, G_{t^x}^{ix}$, and the deletion buffer with committed grammars $G^x_{r^x-4T+1}, G^x_{r^x-4T+2}, \ldots, G^x_{r^x}$ and active grammars $G_1^{dx}, \ldots, G_{t'^x}^{dx}$, $t'^x \leq T$. Its Hamming sketch sketches the sequence of grammars $G^x_{r^x-2T+1}, G^x_{r^x-2T+2}, \ldots, G^x_{s^x-2T}$. Similarly for y, we have the committed insertion grammars $G^y_{s^y-4T+1}, G^y_{s^y-4T+2}, \ldots, G^y_{s^y}$, etc.

We extend the notation so for $j \in \{1, \ldots, t^x\}$, we let $G^x_{s^x+j}$ denote the active grammar G_j^{ix}, and similarly for y. Let $d^x = s^x + t^x - r^x$ and $d^y = s^y + t^y - r^y$. We assume that the hash functions used to decompose ux and vy into grammars satisfy the probabilistic conclusion of Theorem 3.12. That means that grammars $G^x_r, \ldots$ and $G^y_r, \ldots$ can be aligned from the right so G^x_j corresponds to $G^y_{j-d^x+d^y}$, for $j \geq r^x$ (they might not be identical because of the edit operations). Without loss of generality we assume that $d^x \geq d^y$.

Before proceeding with the algorithm we first observe that $d^x - d^y < 2T$. Let $p^x \geq r^x + 1$ be the index of the grammar $G^x_{p^x}$ which produces the first symbol of x when we evaluate all the grammars. Similarly, $p^y \geq r^y + 1$ is the index of $G^y_{p^y}$ which produces the first symbol of y. By Lemma 4.2 applied on $x \leftarrow u$ and $z \leftarrow x$ we get that $p^x \leq r^x + t'^x + T \leq r^x + 2T$, and similarly $p^y \leq r^y + 2T$. By our assumption on success of Theorem 3.12, $s^x + t^x - p^x = s^y - t^y - p^y$. Hence, $s^x + t^x - s^y - t^y = p^x - p^y \leq r^x + 2T - r^y - 1 \leq r^x - r^y + (2T-1)$. Thus $d^x - d^y = s^x + t^x - r^x - s^y - t^y + r^y \leq r^x - r^y + (2T-1) - r^x + r^y \leq 2T - 1$.

If $d^x < 10T$ then we can recover all the grammars $G^x_{r^x-2T+1}, G^x_{r^x-2T+2}, \ldots, G^x_{s^x-2T}$ from their Hamming sketch by constructing an auxiliary *dummy* Hamming sketch sk' for a sequence of 1's of length $(s^x - r^x)M$ and comparing the two sketches. (M is the length of the encoding of each grammar.) Their mismatch information reveals all the grammars $G^x_{r^x-2T+1}, \ldots, G^x_{s^x-2T}$ Since $d^y \leq d^x$, we can similarly recover all the grammars $G^y_{r^y-2T+1}, \ldots, G^y_{s^y-2T}$ from their Hamming sketch.

Thus we know all grammars $G^x_{r^x+1}, G^x_{r^x+2}, \ldots, G^x_{s^x+t^x}$ and $G^x_{r^y+1}, G^y_{r^y+2}, \ldots, G^y_{s^y+t^y}$. We also know grammars $G_1^{dx}, \ldots, G_{t'^x}^{dx}$ and $G_1^{dy}, \ldots, G_{t'^y}^{dy}$ that need to be *subtracted* from our grammars. As noted in Section 2.1, for each of the grammars we can calculate its evaluation size. From that information we can easily identify p^x and p^y, and shorten the grammars $G^x_{p^x}$ and $G^y_{p^y}$ to produce only symbols of x and y, respectively. We can combine all the grammars of x into one grammar G^x, and all the grammars of y into G^y, and run the algorithm of Ganesh, Kociumaka, Lincoln and Saha [GKLS22] to calculate the edit distance of x and y. Since $T = \widetilde{O}(1)$, that will take time $\widetilde{O}(|G^x| + |G^y| + k^2) = \widetilde{O}(k^2)$.

If $d^x \geq 10T$ then we proceed as follows. Clearly, $d^y \geq 8T$, so $s^y - r^y \geq 7T$ and $s^x - r^x \geq 9T$. Thus $G^x_{r^x-2T+1}, G^x_{r^x-2T+2}, \ldots, G^x_{s^x-2T}$ and $G^y_{r^y-2T+1}, G^y_{r^y-2T+2}, \ldots, G^y_{s^y-2T}$ consist of at least $7T$ grammars each, and those grammars are sketched by their Hamming sketches. Although we assume that there is a correspondence between the grammar G^x_j, for $j \geq r^x$, and $G^y_{j-d^x+d^y}$ the sequences $G^x_{r^x-2T+1}, \ldots, G^x_{s^x-2T}$ and $G^y_{r^y-2T+1}, \ldots, G^y_{s^y-2T}$ are misaligned in their Hamming sketches by $d^x - d^y$ grammars. To rectify this misalignment, we prepend $(d^x - d^y)M$ copies of symbol 1 into the sketch for $G^y_{r^y-2T+1}, \ldots, G^y_{s^y-2T}$. Furthermore, if $t^x < t^y$ then we append $(t^y - t^x)M$ ones into the sketch for $G^y_{r^y-2T+1}, \ldots, G^y_{s^y-2T}$, to rectify the difference in the number of sketched grammars. Otherwise if $t^x > t^y$ then we append $(t^x - t^y)M$

ones into the sketch for $G^x_{r^x-2T+1}, \dots, G^x_{s^x-2T}$.

Now we can calculate the mismatch information from the Hamming sketches to find out the pairs of grammars G^x_j and $G^y_{j-d^x+d^y}$, $j \geq r^x+1$, that are different.

If for some $j \in \{r^x+1, \dots, r^x+2T\}$, G^x_j and $G^y_{j-d^x+d^y}$ differ then because we use the Karp-Rabin fingerprint of the two grammars to encode also the neighboring grammars up-to distance $2T$, we recover from the sketch all the grammars G^x_j and $G^y_{j-d^x+d^y}$, for $j = r^x+1, \dots, r^x+2T$. By counting the evaluation size of each of those grammars and comparing it with the evaluation size of active grammars in deletion buffers of x and y, resp., we identify p^x and p^y, and how much the grammars $G^x_{p^x}$ and $G^y_{p^y}$ should be shortened to produce only symbols of x and y. After shortening $G^x_{p^x}$ and $G^y_{p^y}$ we calculate the edit distance of their evaluation. We sum it up with the edit distance of evaluation of each pair of grammars G^x_j and $G^y_{j-d^x+d^y}$, for $j > p^x$, that was identified as mismatch by the Hamming distance sketch or that belongs among the active grammars in insertion buffers of either x or y. There will be at most T mismatched pairs involving the active grammars, and $(4T+1)k$ pairs identified by the Hamming sketch.

In the remaining case when G^x_j and $G^y_{j-d^x+d^y}$ are identical for all $j \in \{r^x+1, \dots, r^x+2T\}$, we might not be able to recover all those grammars from the Hamming sketches, and we might not be able to identify p^x and p^y. However, since $G^x_{p^x} = G^y_{p^y}$, we know that the part of x produced by $G^x_{p^x}$ is either a prefix or suffix of the part of y produced by $G^y_{p^y}$. The difference in the size of the two parts is the edit distance of the two parts. The difference is given by the difference between the total evaluation size of active grammars in the deletion buffer of x, and the total evaluation size of active grammars in the deletion buffer of y together with grammars $G^y_{r_y-j}$, for $j = 0, \dots, d^x - d^y - 1$. The latter grammars are in the committed deletion buffer of y and they agree with $G^x_{r^x+1}, \dots, G^x_{r^x+d^x-d^y}$. Hence, the edit distance of the parts of x and y coming from $G^x_{p^x}$ and $G^y_{p^y}$ can be determined. All other mismatching pairs of grammars are identified by the Hamming sketch or are among active grammars of the insertion buffers. So we proceed as in the previous case to calculate their contribution to the edit distance of x and y. The edit distance of x and y is the sum of those edit distances.

We see that in both the cases we need the Hamming sketch to be able to recover at least T mismatched grammars at the very end caused by the dummy padding, $4T$ grammars at the beginning corresponding to $G^x_{r^x-2T+1}, G^x_{r^x-2T+2}, \dots, G^x_{r^x+2T}$, $2T$ neighbors of $G^x_{r^x+2T}$ to the right, and at most $(4T+1)k$ mismatched grammars caused by the edit operations between x and y. This is less than $M(4T+1)(k+2)$ which is the number of mismatches our Hamming sketch can recover.

The time needed to compare the sketched strings can be bounded as follows: In total the procedure generates at most $O(Tk)$ pairs of grammars of total size $\widetilde{O}(k^2)$ on which it runs edit distance computation from Proposition 2.1. If those edit distance computations take total time more than $\widetilde{O}(k^2)$ we can terminate them as we know the overall edit distance is larger than k. Recovering differing grammars from the Hamming distance sketch takes time $\widetilde{O}(k') = \widetilde{O}(k^2)$. Their follow-up processing such as counting their evaluation size and shortening them is proportional to their total size which is $\widetilde{O}(k^2)$. Hence, the time for comparing strings is $\widetilde{O}(k^2)$.

Failure probability. The update operations can fail if the grammar decomposition produces large grammars or the grammars are not deterministic (because of a collision caused by compression hash functions). This happens with probability at most $1/n$ for each update. Since we perform at most m updates, the failure probability of any update operations is at most $m/n \leq 1/10m^2$ by our choice of m and n.

When comparing two sketches for strings x and y of edit distance at most k, Theorem 3.12 might fail to align their grammar decomposition. This happens with probability at most $1/5$. With probability at most $2/n$ the Hamming sketches might fail to recover the differing pairs of grammars. There is no other source of failure for strings of edit distance at most k so the probability of the compare operation failing is at most $1/3$.

To boost the success probability of comparison from $2/3$ to $1 - 1/2m$, we again form a more robust sketch by taking $c \log m$ independent copies of the rolling edit distance sketch and operate on them simultaneously. For comparison we output the most frequent answer from the individual sketches. This multiplies the failure probability of update operations by $c \log m$, so it is still at most $1/2m$ for m large enough. The comparison will fail with probability at most $1/2m$.

For strings of edit distance more than k the comparison of an individual edit sketch will fail either because the Hamming sketch would need to recover more than k pairs of differing grammars or because the total edit distance of the differing grammars is more than k. In both these failure cases we can always output ∞ to be on the safe side.

4.1 Proofs of Lemma 4.1 and 4.2

Here we prove the remaining two lemmas.

Proof of Lemma 4.1. For the simplicity of case analysis we first compare the compression of wu and w. Consider the division of $w = B_1 \dots B_m$ when calling $\text{Compress}(w, \ell)$, and the division $wu = B'_1 \dots B'_{m'}$ when calling $\text{Compress}(wu, \ell)$. Let $w''' = \text{Compress}(w, \ell)$, from line. Let a be the last symbol of w. We consider three cases.

If $B_m = a^r$, $r \geq 1$, then $B_i = B'_i$ for all $i = 1 \dots, m-1$, and B_m is a prefix of B'_m, not necessarily proper. In this case, the compression of each B_i and B'_i, $i = 1, \dots, m-1$, is the same, so w''' equals to w' in all but possibly the last symbol.

Otherwise, B_m consists of at least two singleton symbols. If the first symbol of u is a, then $B'_m = B_m[1, |B_m| - 1]$, and $B'_{m+1} = a^r$, for some $r \geq 2$. F_{CVL} will color the same all symbols of B_m and B'_m except for at most the last R symbols. Hence, B_m and B'_m will be compressed the same except for at most the last R symbols. The last R symbols are compressed into at most R symbols in w''', and B'_{m+1} will be compressed into a single symbol. In this case we conclude that $w'[1, |w'| - R - 1] = w'''[1, |w'| - R - 1]$.

If the first symbol of u is not a then B_m is a prefix of B'_m, and the compression of B_m and $B'_m[1, |B_m|]$ will differ in at most R last symbols. So $w'[1, |w'| - R] = w'''[1, |w'| - R]$.

Hence, in all three cases $w'[1, |w'| - R - 1] = w'''[1, |w'| - R - 1]$. Moreover, $||w'| - |w'''|| \leq R + 1$.

A similar argument gives $w''[1, |w''| - R - 1] = w'''[1, |w''| - R - 1]$, and $||w''| - |w'''|| \leq R + 1$. By the triangle inequality, $||w'| - |w''|| \leq 2(R + 1)$. Hence, $w'[1, |w'| - 3(R + 1)] = w''[1, |w'| - 3(R + 1)]$. Since $|u'| \leq |u|$, $|w'u'| - |u| \leq |w'|$. The claim follows. □

Proof of Lemma 4.2. Part 1. Consider strings $B^x(\ell, i)$ from the trace of the algorithm on x given the hash functions $H_0, \dots, H_L, C_1, \dots, C_L$. (See Section 3.2) Similarly for $B^{xz}(\ell, i)$.

For $\ell = 0, \dots, L$ we will define integers i_ℓ and Δ_ℓ satisfying:

1. For all $i < i_\ell$, $B^x(\ell, i) = B^{xz}(\ell, i)$,

2. $B^x(\ell, i_\ell)[1, |B^x(\ell, i_\ell)| - \Delta_\ell] = B^{xz}(\ell, i_\ell)[1, |B^x(\ell, i_\ell)| - \Delta_\ell]$,

3. $\Delta_\ell + \sum_{i=i_\ell+1}^{s^x_\ell} |B^x(\ell, i)| \leq \ell(3R + 3) + 1$.

For $\ell = 0$, $B^x(0, 1), B^x(0, 2), \dots, B^x(0, s^x_0) = \text{Split}(x, 0)$, so we set $i_0 = s^x_0$ and $\Delta_0 = 1$. Since $B^{xz}(0, 1), B^{xz}(0, 2), \dots, B^{xz}(0, s^{xz}_0) = \text{Split}(xz, 0)$, $s^x_0 \leq s^{xz}_0$ and $B^x(0, s^x_0)$ might differ from $B^{xz}(0, s^x_0)$ by containing the last symbol of x which might be the first symbol of $B^{xz}(0, s^x_0 + 1)$. Otherwise, $B^x(0, s^x_0)$ is the prefix of $B^{xz}(0, s^x_0)$ so the properties of i_0 and Δ_0 are satisfied.

For $\ell = 1, \dots, L$, having defined $i_{\ell-1}$ and $\Delta_{\ell-1}$ we will define i_ℓ and Δ_ℓ: Define

$$A^x_{\ell-1} = \mathrm{Compress}(B^x(\ell-1, i_{\ell-1}), \ell) \quad \& \quad A^{xz}_{\ell-1} = \mathrm{Compress}(B^{xz}(\ell-1, i_{\ell-1}), \ell),$$

$$(B_0, B_1, \dots, B_m) = \mathrm{Split}(A^x_{\ell-1}, \ell) \quad \& \quad (B'_0, B'_1, \dots, B'_{m'}) = \mathrm{Split}(A^{xz}_{\ell-1}, \ell).$$

For simplicity of exposition in this proof we assume that for any $B \in \Gamma^*$ of size at most 2, $\mathrm{Compress}(B, \ell) = B$ and $\mathrm{Split}(B, \ell) = (B)$, so they both perform no action on B of size at most 2.

Let

$$\begin{aligned} w &= B^x(\ell-1, i_{\ell-1})[1, |B^x(\ell-1, i_{\ell-1})| - \Delta_{\ell-1}], \\ u &= B^x(\ell-1, i_{\ell-1})[1 + |B^x(\ell-1, i_{\ell-1})| - \Delta_{\ell-1}, \dots], \\ v &= B^{xz}(\ell-1, i_{\ell-1})[1 + |B^x(\ell-1, i_{\ell-1})| - \Delta_{\ell-1}, \dots]. \end{aligned}$$

By Lemma 4.1, $A^x_{\ell-1}$ and $A^{xz}_{\ell-1}$ agree on at least the first $|A^x_{\ell-1}| - (3R+3) - |u| = |A^x_{\ell-1}| - (3R+3) - \Delta_{\ell-1}$ symbols. (This is trivial when $|w| \le 2$, in particular, when $|B^x(\ell-1, i_{\ell-1})| \le 2$ or $|B^{xz}(\ell-1, i_{\ell-1})| \le 2$.)

Let $i \in \{0, \dots, m\}$ be the largest i such that for all $j < i$, $B_j = B'_j$. Let i_ℓ be the index of block B_i among blocks $B^x(\ell, 1), B^x(\ell, 2), \dots, B^x(\ell, s^x_\ell)$. Notice, $B^x(\ell, j) = B^{xz}(\ell, j)$, for all $j < i_\ell$. Let $\Delta_\ell \ge 0$ be the smallest integer such that $B_i[1, |B_i| - \Delta_\ell] = B'_i[1, |B_i| - \Delta_\ell]$. (So the second property holds for ℓ, as $B^x(\ell, i_\ell) = B_i$ and $B^{xz}(\ell, i_\ell) = B'_i$.) Since $B_i[|B_i| - \Delta_\ell + 1, \dots] \cdot B_{i+1} \dots B_m$ forms a part of a suffix of $A^x_{\ell-1}$ on which $A^x_{\ell-1}$ and $A^{xz}_{\ell-1}$ differ, $\Delta_\ell + \sum_{j=i+1}^m |B_j| \le (3R+3) + \Delta_{\ell-1}$.

Notice, $\sum_{j=i_\ell+1+s-i}^{s^x_\ell} |B^x(\ell, j)| \le \sum_{j=i_{\ell-1}+1}^{s^x_{\ell-1}} |B^x(\ell-1, j)|$ as each $B^x(\ell, j)$ on the left is a part of a compression of some $B^x(\ell-1, j)$ on the right. Hence,

$$\sum_{j=i_{\ell+1}}^{s^x_\ell} |B^x(\ell, j)| + \Delta_\ell \le (3R+3) + \Delta_{\ell-1} + \sum_{j=i_{\ell-1+1}}^{s^x_{\ell-1}} |B^x(\ell-1, j)| \le (3R+3) + (\ell-1)(3R+3) + 1 \le \ell(3R+3) + 1$$

Eventually, $\Delta_L + \sum_{i=i_L+1}^{s^x_L} |B^x(L, i)| \le L(3R+3)+1$. Also $B^x(L, j) = B^{xz}(L, j)$ for $j = 1, \dots, i_L - 1$. Hence, $G^x_j = G^{xz}_j$ for $j = 1, \dots, i_L - 1$. Since each $|B^x(\ell, j)| \ge 1$, $s^x_L - i_L \le L(3R+3)+1$, which implies $s^x_L - (L(3R+3)+2) \le s^x_L - T \le i_L - 1$ and the claim follows.

Part 2. The proof of this part proceeds similarly to the first part. Let $B^x(\ell, i)$ and $B^{xz}(\ell, i)$ be as above. For $\ell = 0, \dots, L$, we will define a sequence of integers $i_\ell, t_\ell, p_\ell, r_\ell$ satisfying:

1. t_ℓ is the last index i such that $B^{xz}(\ell, i)$ contains some symbol that comes from the compression of x, and r_ℓ is the length of the prefix of $B^{xz}(\ell, t_\ell)$ that comes from the compression of x.

2. $i_\ell \le t_\ell$ and for all $i < i_\ell$, $B^x(\ell, i) = B^{xz}(\ell, i)$,

3. $B^x(\ell, i_\ell)[1, p_\ell] = B^{xz}(\ell, i_\ell)[1, p_\ell]$,

4. $r_\ell - p_\ell + \sum_{j=i_\ell}^{t_\ell - 1} |B^{xz}(\ell, j)| \le \ell(3R+3)+1$.

For $\ell = 0$, $B^x(0,1), B^x(0,2), \dots, B^x(0, s^x_0) = \mathrm{Split}(x, 0)$ and $B^{xz}(0,1), B^{xz}(0,2), \dots, B^{xz}(0, s^{xz}_0) = \mathrm{Split}(xz, 0)$, so we set $i_0 = s^x_0$. If $|B^x(0, s^x_0)| \le |B^{xz}(0, s^x_0)|$ we set $t_0 = i_0$, and $r_0 = p_0 = |B^x(0, i_0)|$, otherwise the last symbol of x starts the block $B^{xz}(0, s^x_0 + 1)$ so we set $t_0 = i_0 + 1$, $p_0 = |B^x(0, s^x_0)|$ and $r_0 = 1$. (For completeness we set $i_{-1} = t_{-1} = 1$.) Clearly, the four properties are satisfied by this choice.

For $\ell = 1, \dots, L$, having defined $i_{\ell-1}, t_{\ell-1}, p_{\ell-1}, r_{\ell-1}$ we will define i_ℓ, t_ℓ, p_ℓ, and r_ℓ. As before, let

$$A^x_{\ell-1} = \text{Compress}(B^x(\ell-1, i_{\ell-1}), \ell) \quad \& \quad A^{xz}_{\ell-1} = \text{Compress}(B^{xz}(\ell-1, i_{\ell-1}), \ell).$$

Case 1. Consider the case when $i_{\ell-1} = t_{\ell-1}$. Let

$$\begin{aligned} w &= B^{xz}(\ell-1, i_{\ell-1})[1, p_{\ell-1}], \\ u_x &= B^{xz}(\ell-1, i_{\ell-1})[1+p_{\ell-1}, r_\ell], \\ u_z &= B^{xz}(\ell-1, i_{\ell-1})[1+r_{\ell-1}, \dots], \\ v &= B^{x}(\ell-1, i_{\ell-1})[1+p_{\ell-1}, \dots]. \end{aligned}$$

Let $A^{xz}_{\ell-1} = w'u'_xu'_z$ where w' comes from the compression of w, u'_x comes from the compression of u_x, and u'_z comes from the compression of u_z. Let $A^x_{\ell-1} = w''v'$ where w'' comes from the compression of w, and v' comes from the compression of v. Set $r'_\ell = |w'u'_x|$, let $p'_\ell \le r'_\ell$ be the largest integer so that $A^{xz}_{\ell-1}[1, p'_\ell] = A^x_{\ell-1}[1, p'_\ell]$. By Lemma 4.1, $p'_\ell \ge |w'| - 3(R+1)$ so $r'_\ell - p'_\ell \le |w'u'_x| - |w'| + 3(R+1) \le |u'_x| + 3(R+1)$. Furthermore, $|u'_x| \le |u_x| \le r_{\ell-1} - p_{\ell-1} \le (\ell-1)\cdot(3R+3)+1$, which follows by properties of $p_{\ell-1}$ and $r_{\ell-1}$, so $r'_\ell - p'_\ell \le \ell(3R+3)+1$.

Let $(B_0, B_1, \dots, B_m) = \text{Split}(A^{xz}_{\ell-1}, \ell)$. Let $i \ge 0$ be the smallest integer such that $p'_\ell \le \sum_{j=0}^{i}|B_j|$ and let $t \ge 0$ be the smallest such that $r'_\ell \le \sum_{j=0}^{t}|B_j|$. Set $p_\ell = p'_\ell - \sum_{j=0}^{i-1}|B_j|$ and $r_\ell = r'_\ell - \sum_{j=0}^{t-1}|B_j|$. Let i_ℓ be the index of block B_i among blocks $B^{xz}(\ell,1), B^{xz}(\ell,2), \dots, B^{xz}(\ell, s^{xz}_\ell)$, and let t_ℓ be the index of B_t among those blocks. Notice, $B^x(\ell, j) = B^{xz}(\ell, j)$, for all $j < i_\ell$. We conclude the case by observing that $r_\ell - p_\ell + \sum_{j=i}^{t-1}|B_j| = \sum_{j=i}^{t-1}|B_j| + (r'_\ell - \sum_{j=0}^{t-1}|B_j|) - (p'_\ell - \sum_{j=0}^{i-1}|B_j|) = p'_\ell - r'_\ell \le \ell(3R+3)+1$.

Case 2. The case $i_{\ell-1} < t_{\ell-1}$ is similar. In this case we let w and v to be as in the previous case and $u = B^{xz}(\ell-1, i_{\ell-1})[1+p_{\ell-1}, \dots]$. We let $p'_\ell \le |A^{xy}_{\ell-1}|$ be the largest integer so that $A^{xz}_{\ell-1}[1, p'_\ell] = A^x_{\ell-1}[1, p'_\ell]$. By Lemma 4.1, $p'_\ell \ge |A^{xy}_{\ell-1}| - 3(R+1) - |u| = |A^{xy}_{\ell-1}| - 3(R+1) - |B^{xz}(\ell-1, i_{\ell-1})| + p_{\ell-1}$. Rearranging terms: $-p'_\ell + |A^{xy}_{\ell-1}| \le -p_{\ell-1} + |B^{xz}(\ell-1, i_{\ell-1})| + 3(R+1)$.

Let $i \ge 0$ be the smallest integer such that $p'_\ell \le \sum_{j=0}^{i}|B_j|$. Let $p_\ell = p'_\ell - \sum_{j=0}^{i-1}|B_j|$, and i_ℓ be the index of the block B_i within $B^{xz}(\ell,1), B^{xz}(\ell,2), \dots, B^{xz}(\ell, s^{xz}_\ell)$. Hence, $-p_\ell + \sum_{j=i_\ell}^{i_\ell+m-i}|B^{xz}(\ell,j)| = -p_\ell + \sum_{j=i}^{m}|B_j| = -p'_\ell + \sum_{j=0}^{m}|B_j| = -p'_\ell + |A^{xy}_{\ell-1}| \le -p_{\ell-1} + |B^{xz}(\ell-1, i_{\ell-1})| + 3(R+1)$.

Let $C^{xz}_{\ell-1} = \text{Compress}(B^{xz}(\ell-1, t_{\ell-1}), \ell)$ and $(B'_0, B'_1, \dots, B'_{m'}) = \text{Split}(C^{xz}_{\ell-1}, \ell)$. Let r'_ℓ be the largest position in $C^{xz}_{\ell-1}$ of a symbol coming from compression of x, and $t \ge 0$ be the smallest integer such that $r'_\ell \le \sum_{j=0}^{t}|B_j|$, and set $r_\ell = r'_\ell - \sum_{j=0}^{t-1}|B'_j|$. Let t_ℓ be the index of the block B'_t within $B^{xz}(\ell,1), B^{xz}(\ell,2), \dots, B^{xz}(\ell, s^{xz}_\ell)$. Clearly, $r'_\ell \le r_{\ell-1}$ so $r_\ell + \sum_{j=t_\ell-t}^{t_\ell-1}|B^{xz}(\ell,j)| = r_\ell + \sum_{j=0}^{t-1}|B'_j| \le r'_\ell \le r_{\ell-1}$.

Notice, $\sum_{j=i_\ell+m-i+1}^{t_\ell-t-1}|B^{xz}(\ell,j)| \le \sum_{j=i_{\ell-1}+1}^{t_{\ell-1}-1}|B^{xz}(\ell-1,j)|$. By partitioning the sum, rearranging the terms and using the upper bounds derived so far we have: $r_\ell - p_\ell + \sum_{j=i_\ell}^{t_\ell-1}|B^{xz}(\ell,j)| = -p_\ell + \sum_{j=i_\ell}^{i_\ell+m-i}|B^{xz}(\ell,j)| + \sum_{j=i_\ell+m-i+1}^{t_\ell-t-1}|B^{xz}(\ell,j)| + \sum_{j=t_\ell-t}^{t_\ell-1}|B^{xz}(\ell,j)| + r_\ell \le -p_{\ell-1} + |B^{xz}(\ell-1, i_{\ell-1})| + 3(R+1) + \sum_{j=i_{\ell-1}+1}^{t_{\ell-1}-1}|B^{xz}(\ell-1,j)| + r_{\ell-1} = r_{\ell-1} - p_{\ell-1} + \sum_{j=i_{\ell-1}}^{t_{\ell-1}-1}|B^{xz}(\ell-1,j)| + 3(R+1) \le (\ell-1)\cdot(3R+3) + 1 + 3(R+1) \le \ell(3R+3)+1$, where the second to last inequality follows by the properties of our numbers for $\ell - 1$.

For $\ell = L$ we get: $r_L - p_L + \sum_{j=i_L}^{t_L-1}|B^{xz}(L,j)| \le L(3R+3)+1$. Since $p_L \le |B^{xz}(L, i_L)|$, and $r_L \ge 0$ we get: $\sum_{j=i_L+1}^{t_L-1}|B^{xz}(\ell,j)| \le L(3R+3)+2$. Since each $B^{xz}(L,j)$ is of non-zero size, $t_L - i_L - 1 \le L(3R+3)+2$. Thus $t_L \le i_L + L(3R+3) + 3 \le s^x_L + L(3R+3)+3$, as $i_L \le s^x_L$. The claim follows. □

5 Appending a symbol to a grammar decomposition

In this section we provide a detailed description of the process of updating the active grammars of a string x when appending a new symbol a. By Lemma 4.2 only the last T grammars of x might change when adding a new symbol a. As observed already previously, Lemma 4.2 also implies that once a grammar becomes more than $(T + 1)$-th grammar from the end it will never change, despite the fact that the number of grammars that follow it might shrink after adding more symbols. (Adding more symbols might create periodicity that will be exploited by the compression.) Our rolling sketch algorithm keeps at most T active grammars that might still change after adding more symbols. It is convenient for our implementation of the update function to have access also to the previous at most T committed grammars (to have the proper context for re-compression). Our rolling sketch algorithm has those committed grammars available in appropriate buffers. Thus we will assume that the update function is always invoked with exactly $T + 1$ grammars, unless x is decomposed into less that $T + 1$ grammars. Some of the first few grammars from the output of the update procedure should be discarded as they correspond to grammars that should stay the same. In particular, if there are t active grammars and s committed grammars then we should discard the first $\min(s, T + 1 - t)$ grammars from its output. The following statement encapsulates the properties of our update procedure $\mathrm{UpdateActiveGrammars}()$.

Theorem 5.1. *Let integers $k \leq n$ and functions $C_1, \ldots, C_L$ and $H_0, \ldots, H_L$ be given. For any $a \in \Sigma$ and $x \in \Sigma^*$ of length at most n with $G_1, \ldots, G_s$ being the grammars output by the decomposition algorithm on input x using functions $C_1, \ldots, C_L, H_0, \ldots, H_L$, $\mathrm{UpdateActiveGrammars}(G_{s-\min(s,T+1)+1}, \ldots, G_s, a)$ outputs a sequence of grammars $G'_1, \ldots, G'_{t'}$ such that $G_1, \ldots, G_{s-\min(s,T+1)}, G'_1, \ldots, G'_{t'}$ is the sequence that would be output by the decomposition algorithm on $x \cdot a$ using the functions $C_1, \ldots, C_L, H_0, \ldots, H_L$. The update algorithm runs in time $\widetilde{O}(kLT) = \widetilde{O}(k)$ and outputs $t' \leq 4TL$ grammars.*

Here we assume that the decomposition algorithm does not fail neither on x nor on $x \cdot a$ with respect to producing correct deterministic grammars so the first two parts of Theorem 3.1 are satisfied for x and $y = x \cdot a$, and the choice of functions $C_1, \ldots, C_L$ and $H_0, \ldots, H_L$. For the simplicity of our implementation, we assume a stronger property of $C_1, \ldots, C_L$, that each C_ℓ is one-to-one on the union of all blocks of x and $x \cdot a$ at level ℓ. (See remark after Lemma 3.7.)

5.1 Auxiliary functions

Our update algorithm uses several simple and straightforward auxiliary functions we describe next. Function $\mathrm{DecompressSymbol}(c, G, \ell, t)$ takes a symbol $c \in \Gamma$ and if it is a level-ℓ symbol compressed by the grammar G then it returns its decompression truncated to the length of at most t symbols. Otherwise it returns the original symbols c.

Algorithm 5 $\mathrm{DecompressSymbol}(c, G, \ell, t)$

Input: A symbol c, a grammar G, a level ℓ, maximum output size $t \geq 2$.
Output: Decompresses c if it was compressed at level ℓ. Returns at most t symbols of the decompression.

```
32 if $c \in \Sigma_c^\ell$ then let $a, b \in \Gamma$ be such that $c \to ab \in G$. Return $ab$.
33 if $c \in \Sigma_r^\ell$ then let $a \in \Gamma, r \in \mathbb{N}$ be such that $c = \mathtt{r}_{a,r}$. Return $a^{\min(t,r)}$.
34 Return $c$.
```

Function $\mathrm{DecompressString}(Z, G, \ell)$ decompresses all level-ℓ compression symbols in a string $Z \in \Gamma^*$ using the grammar G, and returns the resulting decompressed string.

```
**Algorithm 6** $\mathrm{DecompressString}(Z, G, \ell)$
**Input:** A string $Z$, a grammar $G$, and level $\ell$.
**Output:** Decompresses $z$ at level $\ell$.

$Y = \varepsilon$.
for $i = 1$ to $|Z|$ do $Y = Y \cdot \mathrm{DecompressSymbol}(Z[i], G, \ell, \infty)$.
Return $Y$.
```

Function $\mathrm{DecompressSymbolLength}(c, \ell)$ returns the length of the decompression of a symbol c at level ℓ.

```
**Algorithm 7** $\mathrm{DecompressSymbolLength}(c, \ell)$
**Input:** A symbol $c$, a level $\ell$.
**Output:** Returns the length of decompression of $c$ at level $\ell$.

if $c \in \Sigma_c^\ell$ then return 2.
if $c \in \Sigma_r^\ell$ then let $a \in \Gamma, r \in \mathbb{N}$ be such that $c = \mathtt{r}_{a,r}$. Return $r$.
Return 1.
```

```
**Algorithm 8** $\mathrm{CompressWithGrammar}(B, \ell)$
**Input:** String $B$ over alphabet $\Gamma$, and level number $\ell$.
**Output:** String $B''$ over alphabet $\Gamma$, and set of applied rules $G'$.

if $|B| \leq 1$ then return $B, \emptyset$.
Set $G' = \emptyset$.
Divide $B = B_1 B_2 B_3 \ldots B_m$ into minimum number of blocks so that each maximal subword $a^r$ of $B$, for
    $a \in \Gamma$ and $r \geq 2$, is one of the blocks.
for each $i \in \{1, \ldots, m\}$ do
    if $B_i = a^r$, where $r \geq 2$ then
        Set $B'_i = \mathtt{r}_{a,r} \cdot \#$ and color $\mathtt{r}_{a,r}$ by 1 and $\#$ by 2.
        $G' = G' \cup \{\mathtt{r}_{a,r} \to a^r\}$;
    end
    else Set $B'_i = B_i$ and color each symbol of $B'_i$ according to $F_{\mathrm{CVL}}(B_i)$
end
Set $B' = B'_1 B'_2 \cdots B'_m$, $B'' = \varepsilon$, and $i = 1$.
while $i < |B'|$ do
    if $B'[i+1] = \#$ then $B'' = B'' \cdot B'[i]$
    else
        $B'' = B'' \cdot C_\ell(B'[i, i+1])$;
        $G' = G' \cup \{C_\ell(B'[i, i+1]) \to B'[i, i+1]\}$;
    end
    $i = i + 2$.
    if $i \leq |B'|$ and $B'[i]$ is not colored 1 then $B'' = B'' \cdot B'[i]$, $i = i + 1$
end
Return $B'', G'$.
```

Function $\mathrm{CompressWithGrammar}(B, \ell)$ is an extension of $\mathrm{Compress}(B, \ell)$ that in addition to compressed block B at level ℓ returns the set of grammar rules used for the compression of B at this level.

Finally, function $\mathrm{FindCompressedPrefix}(Z, p, \ell)$ returns the length of the smallest prefix of a string Z that decompresses into at least p symbols at level ℓ.

Algorithm 9 $\mathrm{FindCompressedPrefix}(Z, p, \ell)$

Input: String Z, an integer p, level ℓ.

Output: Smallest index j such that level ℓ decompression of $Z[1, j]$ has length $\geq p$.

```
62 $q = 0$ and $j = 0$.
63 while $q < p$ do
64     $j = j + 1$;
65     $p = p + \mathrm{DecompressSymbolLength}(Z[j], \ell)$;
66 end
67 Return $j$.
```

5.2 Main functions

The core of the update function $\mathrm{UpdateActiveGrammars}((G_1, \dots, G_t), a)$ is build around the functions we describe next. The functions use globally accessible set of grammar rules G that contains all the rules from $G_1, \dots, G_t$ except for the starting rules. (This set of rules is deterministic assuming the remark after Theorem 5.1.)

The functions will build a sequence of strings $Z_L, Z_{L-1}, \dots, Z_0$ each of length at most $2T$. Z_L is the concatenation of the right-hand-sides of starting rules of $G_1, \dots, G_t$. For $\ell = L, \dots, 1$, $Z_{\ell-1}$ is then build inductively by decompressing a (suitable) largest suffix of Z_ℓ so that $Z_{\ell-1}$ would be of length at most $T + 4 \leq 2T$. The decompression is provided by function $\mathrm{PartiallyDecompress}(Z, F, \ell)$ which returns tuple Z', F', u, r'. In the case that the first symbol of the decompressed suffix of Z_ℓ is the level-ℓ repeat symbol $\mathtt{r}_{a,r}$ that would expand $Z_{\ell-1}$ beyond the limit of $T + 4$ symbols, we truncate the expansion of that symbol to the length $r_\ell = r'$. The return value u indicates how many symbols of Z were left uncompressed (which would include the partially decompressed symbol $\mathtt{r}_{a,r}$). It satisfies that if $u \neq 0$ then $|Z'| \geq T$. Strings $Z_L, \dots, Z_0$ satisfy that for $\ell = L, \dots, 1$, if $|Z_\ell| \geq T$ then $|Z_{\ell-1}| \geq T$. (In particular, if $\mathrm{UpdateActiveGrammars}()$ is invoked with at least $T + 1$ grammars, then all Z_ℓ are of length at least T. The compression of the first grammar might depend on unseen grammars in that case so we cannot re-compress it at will.)

Strings $Z_L, \dots, Z_0$ are accompanied by strings of integers $F_L, \dots, F_0$ over the alphabet $\{0, \dots, L+1\}$. The value of $F_\ell[i]$ indicates at which level the symbol $Z_\ell[i]$ becomes the first symbol in its block. In particular, $F_\ell[i] < \ell$ indicates that a block starts at position i of Z_ℓ. This value is relevant for re-compression of updated strings $Z_0, \dots, Z_L$. The initial values of F_L are computed using $\mathrm{SplittingDepth}(G)$. Function $\mathrm{SplittingDepth}(G)$ is fairly straightforward: For a grammar G, it inductively decompresses the first two symbols of the evaluation of G. It finds the lowest level ℓ, at which the first two symbols of the decompression give zero when function H_ℓ is applied on them.

After obtaining $Z_L, \dots, Z_0$, $\mathrm{UpdateActiveGrammars}((G_1, \dots, G_t), a)$ appends a to Z_0, and then re-compresses $Z_0, \dots, Z_{L-1}$ using a function $\mathrm{Recompress}(B, Z, F, u, r, \ell)$. We provide more details on function $\mathrm{Recompress}(B, Z, F, u, r, \ell)$ further below. Invoking $\mathrm{UpdateActiveGrammars}((G_1, \dots, G_t), a)$ returns a sequence of updated grammars.

Algorithm 10 $\mathrm{PartiallyDecompress}(Z, F, \ell)$

Input: String Z, splitting depth string F, and level ℓ.

Output: Decompressed string Z', splitting depth string F', unused count u, repeat count r'.

```
68 Set $Z' = \varepsilon$ and $F' = \varepsilon$.
69 for $u = |Z|$ to $1$ do
70     if $Z[u] = \mathtt{r}_{a,r}$, where $\mathtt{r}_{a,r} \in \Sigma_r^\ell$ then
71         if $|Z'| + r \le T + 3$ then $Z' = a^r \cdot Z'$ and $F' = F[u] \cdot (L+1)^{r-1} \cdot F'$
72         else
73             $r' = T - |Z'| + 1$;
74             $Z' = a^{r'} \cdot Z'$ and $F' = (L+1)^{r'} \cdot F'$ ;
75             Return $Z', F', u, r'$.
76         end
77     end
78     else if $Z[u] = a$, where $a \in \Sigma_c^\ell$ then
79         $Z' = b \cdot c \cdot Z'$, where $a \to b \cdot c$ is in $G$;
80         $F' = F[u] \cdot (L+1) \cdot F'$;
81     end
82     else $Z' = Z[u] \cdot Z'$ and $F' = F[u] \cdot F'$
83     if $|Z'| \ge T$ then return $Z', F', u-1, 0$.
84 end
85 Return $Z', F', 0, 0$.
```

Algorithm 11 $\mathrm{SplittingDepth}(G)$

Input: Non-empty grammar G.

Output: The first level ℓ where G would be separated as a new block.

```
86 Let $v$ be such that $\# \to v \in G$.        // $v$ are the first two symbols of $\mathrm{eval}(G)$.
87 $d = L + 1$.
88 for $\ell = L, \ldots, 0$ do
89     if $|v| \ge 2$ and $H_\ell(v[1,2]) = 0$ then $d = \ell$.
90     $u = \mathrm{DecompressSymbol}(v[1], G, \ell, 2)$
91     if $|v| \ge 2$ then $u = u \cdot \mathrm{DecompressSymbol}(v[2], G, \ell, 2)$.
92     $v = u$.
93 end
94 Return $d$.
```

Algorithm 12 $\mathrm{UpdateActiveGrammars}(AG, a)$

Input: List of grammars $AG = (G_1, \ldots, G_t)$ representing a string x, and a symbol a.
Output: Updated list of grammars AG' representing string $x \cdot a$.

```
        // Construct a set of rules G, initial compressed string $Z_L$ and splitting depth string $F_L$.
For $i = 1, \ldots, t$, let $\# \to v_i$ be the starting rule in $G_i$.
Set $G = \bigcup_{i=1}^{t} G_i \setminus \{\# \to v_i\}$.
Set $Z_L = v_1$ and $F_L = 0 \cdot (L+1)^{|v_1|-1}$.
For $i = 2, \ldots, t$, set $Z_L = Z_L \cdot v_i$ and $F_L = F_L \cdot \mathrm{SplittingDepth}(G_i) \cdot (L+1)^{|v_i|-1}$.
        // Perform partial decompression
for $\ell = L$ to $1$ do
    $Z_{\ell-1}, F_{\ell-1}, u_\ell, r_\ell = \mathrm{PartiallyDecompress}(Z_\ell, F_\ell, \ell)$.
end
        // Perform re-compression
$Z_0 = Z_0 \cdot a$; $B = \mathrm{Split}(Z_0, 0)$;
for $\ell = 1$ to $L$ do
    $B', G' = \mathrm{Recompress}(B, Z_\ell, F_\ell, u_\ell, r_\ell, |Z_{\ell-1}|, \ell)$
    $G = G \cup G'$
    $B = B'$
end
Let $B = (B_1, \ldots, B_{t'})$.
$AG' = ()$.
for $i = 1$ to $t'$ do
    $G' = G \cup \{\# \to B_i\}$.
    Remove from $G'$ unnecessary rules to get $G'_i$ (as in Section 2.1).
    Append $G'_i$ to $AG'$.
end
Return $AG'$.
```

Function $\mathrm{Recompress}(B, Z, F, u, r, \ell)$ gets a sequence $B = (B_0, \ldots, B_s)$ of blocks that represent compression of the updated $Z_{\ell-1}$ (after adding a) up-to level $\ell - 1$. It also gets the original Z_ℓ, the splitting depth string F_ℓ, the number of symbols u_ℓ that were decompressed from Z_ℓ to get the original $Z_{\ell-1}$ and the parameter r_ℓ that indicates that the first r_ℓ symbols of $Z_{\ell-1}$ are a partial decompression of the repeat symbol $Z_\ell[u]$. It outputs a sequence of blocks B' that represent the updated block Z_ℓ compressed up-to level ℓ, and a set of rules G' that were used for compression at level ℓ.

Blocks $B_1, \ldots, B_s$ can be independently compressed and split at level ℓ. The block B_0 needs a special treatment though as it needs to be combined with its possible remainder in Z_ℓ. This is done in function $\mathrm{RecompressFirstBlock}(B_0, Z, F, u, r, \ell)$. Remaining blocks for the output $\mathrm{Recompress}()$ are obtained from Z_ℓ by splitting it into blocks according to F_ℓ.

Algorithm 13 Recompress(B, Z, F, u, r, z, ℓ)

Input: $B = (B_0, \ldots, B_s)$ sequence of blocks, original uncompressed string Z, splitting depth string F of Z, u number of uncompressed symbols in Z, repeat count r, $z = |Z_{\ell-1}|$, and level ℓ.

Output: B' a new sequence of blocks representing B together with $Z[1, u]$, and set of newly added rules G'.

```
119 if $z < T$ then $B' = ()$, $G' = \emptyset$, $u' = 0$, $j = 0$.          // No symbols precede $B_0$.
120 else
121     $B', G', u' = \mathrm{RecompressFirstBlock}(B_0, Z, F, u, r, \ell)$.      // Compress block $B_0$.
122     $j = 1$.
123 end
124                                       // Compress blocks $B_j, \ldots, B_s$.
125 for $i = j$ to $s$ do
126     if $|B_i| \le 2$ then $B'' = (B_i)$; $G'' = \emptyset$
127     else
128         $B'_i, G'' = \mathrm{CompressWithGrammar}(B_i, \ell)$.
129         $B'' = \mathrm{Split}(B'_i, \ell)$.
130     end
131     Append $B''$ to $B'$.
132     $G' = G' \cup G''$.
133 end
134 $i = u'$.                              // Separate remaining blocks in $Z$.
135 while $i > 0$ do
136     while $i > 1$ and $F[i] > \ell$ do $i = i - 1$.
137     Add $Z[i, u']$ as the first item of $B'$.
138     $i = i - 1$; $u' = i$.
139 end
140 Return $B', G'$.
```

Function $\mathrm{RecompressFirstBlock}(B_0, Z, F, u, r, \ell)$ is the most complicated function of the whole recompression process. The function is invoked only if $|Z_{\ell-1}| \ge T$. The function gets the first level $\ell - 1$ block B_0 that needs to be combined with its remainder in $Z = Z_\ell$. The remainder is a suffix of $Z_\ell[1, u]$, where r indicates that the first r symbols of the original $Z_{\ell-1}$ were obtained by the partial decompression of $Z_\ell[u]$. If $r \ne 0$ then the compression of the part of B_0 that follows its leading a's ($Z_\ell[u] = \mathtt{r}_{a,r'}$) is independent of the compression of the part of Z belonging to B_0 and preceding $Z_\ell[u]$, as $r' - r \ge 2$. Thus we can compress that part of B_0, combine it with an appropriate repetition symbol $\mathtt{r}_{a,r''}$ and append it to the appropriate suffix of $Z_\ell[1, u-1]$ (which is already compressed at level ℓ.) If $r = 0$ then we invoke a function $\mathrm{CrossOverBlock}(B_0, Z[u', \ldots], u - u' + 1, \ell)$, where u' is the first symbol in Z_ℓ that belongs to the block of B_0. Eventually, we split the compressed block B_0 using $\mathrm{Split}()$.

Algorithm 14 $\mathrm{RecompressFirstBlock}(B_0, Z, F, u, r, \ell)$

```
Input: Block $B_0$, an original uncompressed string $Z$, splitting depth string $F$ of $Z$, $u$ number of uncompressed symbols in $Z$, repeat count $r$, and level $\ell$.
Output: $B'$ a new sequence of blocks representing $B_0$ together with $Z[1, u]$, and set of newly added rules $G'$, number $u'$ of unused symbols in $Z$.
if $r \neq 0$ then $u = u - 1$.
$u' = u + 1$.        // Find the beginning of block $B_0$ in the uncompressed part $Z$.
while $u' > 1$ and $d[u'] \geq \ell$ do  $u' = u' - 1$.
if $r \neq 0$ then
    // Block $B_0$ starts by partially decompressed symbol $\mathtt{r}_{a,r}$.
    Let $a \in \Gamma$ and $r' \in \mathbb{N}$ be such that $Z[u + 1] = \mathtt{r}_{a,r'}$.
    for $i = 1$ to $|B_0|$ do  if $B_0[i] \neq a$ then break;

    if $B_0[i] = a$ then  $B' = \varepsilon, G'' = \emptyset, i = i + 1$.
    else $B', G'' = \mathrm{CompressWithGrammar}(B_0[i, \dots], \ell)$.
    $B' = Z[u', u] \cdot \mathtt{r}_{a,r'-r+i-1} \cdot B'$.
end
else
    $B', G'' = \mathrm{CrossOverBlock}(B_0, Z[u', \dots], u - u' + 1, \ell)$.
end
$B'' = \mathrm{Split}(B', \ell)$.
Return $B'', G'', u' - 1$.
```

Function $\mathrm{CrossOverBlock}(B, Z, u, \ell)$ gets a block B that was compressed up-to level $\ell - 1$ and needs to be combined with its remainder $Z[1, u]$ that is compressed up-to level ℓ. (The resulting block should correspond to "$Z[1, u] \cdot B$".) We know that $|Z_{\ell-1}| \geq T \geq L(3R + 3)$ otherwise $\mathrm{RecompressFirstBlock}()$ and $\mathrm{CrossOverBlock}()$ would not be called. By the three properties of $\Delta_{\ell-1}$ and $i_{\ell-1}$ defined in the proof of Part 1 of Lemma 4.2 we know that the first $3(R + 1)$ symbols of $Z_{\ell-1}$ were not modified as a result of appending the new symbol to x. Hence the first $\min(3(R + 1), |B|)$ symbols of B correspond to the decompression of $Z[u + 1, \dots]$.

In this part of B we look for any repeated symbol. If we find a repeated symbol there, we combine the compression of the part of B starting at the repeated symbol with the original part of $Z[u + 1, \dots]$ that produced the symbols of B preceding the repeated symbol (and also with $Z[1, u]$). By the properties of compression, repeated symbols break dependence between compressed symbols.

If $|B| \leq 2R + 20$ then at least $3(R + 1) - 2R - 20 > 2$ unchanged symbols follow B. Thus B ends at its original location as it was split at some level $< \ell$ and the first two symbols of the next block at all levels $< \ell$ are the same as originally.

Finally, if $|B| > 2R + 20$ and there is no repeated symbol in the first up-to $3(R + 1)$ symbols of B then we can compress B to get B', strip from B' the compression of the first $R + 10$ symbols and combine it with the original compression of those $R + 10$ symbols from Z. (The first up-to $3(R + 1)$ symbols of B consist of singletons. The compression of a singleton depends on the context of at most $R + 3$ symbols on either side.)

Algorithm 15 CrossOverBlock(B, Z, u, ℓ)

Input: Block B, an original uncompressed string Z, number u of unused symbols in Z, and level ℓ.
Output: B' and set of newly added rules G'.

```
                    // Try to find a repeated symbol in unmodified $B$.
$i = 1$.
while $i < |B|$ and $i < 3(R+1)$ and $B[i] \neq B[i+1]$ do $i = i + 1$.
if $i < |B|$ and $B[i] = B[i+1]$ then
    // $B[i]$ is a repeated symbol.
    $B', G' = \mathrm{CompressWithGrammar}(B[i, \dots], \ell)$.
    $j = \mathrm{FindCompressedPrefix}(Z[u+1, \dots], i-1, \ell)$.
    $B' = Z[1, u+j] \cdot B'$.
end
else if $|B| \leq 2R + 20$ then
    $j = \mathrm{FindCompressedPrefix}(Z[u+1, \dots], |B|, \ell)$.    // At least two unchanged symbols follow $B$.
    $B' = Z[1, u+j], G' = \emptyset$.
end
else
    $B', G' = \mathrm{CompressWithGrammar}(B, \ell)$.
    $p = \mathrm{FindCompressedPrefix}(B', R+10, \ell)$.
    $j = \mathrm{FindCompressedPrefix}(Z[u+1, \dots], R+10, \ell)$.
    $B' = Z[1, u+j-1] \cdot B'[p, \dots]$.
end
Return $B', G'$.
```

The correctness of the update algorithm follows from its description.

5.3 Time analysis

We assume that strings are represented efficiently (e.g. by balanced trees) so we can extract a sub-string, concatenate strings, etc. in time $\widetilde{O}(1)$. All strings that we will operate on will be of length $O(T)$. Similarly, we assume that grammars are represented efficiently so that we can look-up a rule with a given left-hand symbol, append two grammars, etc. in time $\widetilde{O}(1)$.

Then $\mathrm{DecompressSymbol}()$ and $\mathrm{DecompressSymbolLength}()$ takes time $\widetilde{O}(1)$. The time complexity of each of the functions $\mathrm{CompressWithGrammar}()$, $\mathrm{DecompressString}()$, $\mathrm{FindCompressedPrefix}()$, $\mathrm{PartiallyDecompress}()$ and $\mathrm{CrossOverBlock}()$ is proportional to the length of strings on which it operates so it is $\widetilde{O}(T)$. Time of $\mathrm{SplittingDepth}()$ is proportional to the depth of the grammar, which in our case is at most $\widetilde{O}(L)$. Each $\mathrm{RecompressFirstBlock}()$ executes $O(T)$ operations on strings and grammars, and $O(T)$ evaluations of H_ℓ (inside the calls to $\mathrm{Split}()$). Since H_ℓ is $\widetilde{O}(k)$-wise independent, its evaluation takes time $\widetilde{O}(k)$. So $\mathrm{RecompressFirstBlock}()$ takes time $\widetilde{O}(kT)$.

Similarly, each $\mathrm{Recompress}()$ executes up-to one call to $\mathrm{RecompressFirstBlock}()$, $O(T)$ operations on strings and grammars, and $O(T)$ evaluations of H_ℓ to split blocks. Again, its total time complexity is $\widetilde{O}(kT)$. Eventually, $\mathrm{UpdateActiveGrammars}()$ executes up-to T $\mathrm{SplittingDepth}()$, $O(T)$ string operations, L calls to $\mathrm{PartiallyDecompress}()$ and $\mathrm{Recompress}()$, and then up-to $O(LT)$ invocations of grammar minimization procedure costing $\widetilde{O}(k)$ time each. Thus, the total time for $\mathrm{UpdateActiveGrammars}()$ is $\widetilde{O}(LTk)$.

The number of grammars the algorithm outputs is at most $\sum_{\ell=0}^{L} |Z_\ell| \le 2T(L+1) \le 4TL$.

6 Table of parameters

Definition	Asymptotics	Meaning	Reference
$R = \log^* \lvert\Gamma\rvert + 20$	$\log^* n$	compression locality	Section 2.3
$L = \lceil \log_{3/2} n \rceil + 3$	$\log n$	recursion depth	Section 3, Corollary 3.3
$D = 110c - R(L+1)k$	$k \log n \log^* n$	$1/$splitting probability	Section 3, Lemma 3.4
$S = 30DL \log n + 6$	$k \log^3 n \log^* n$	maximum grammar size	Section 3, Theorem 3.1
$M = 3S \cdot \lceil 1 + \log \lvert\Gamma\rvert \rceil$	$k \log^4 n \log^* n$	grammar encoding size	Section 3.3
$T = L(3R+6)$	$\log n \log^* n$	locality of suffix changes	Section 4, Lemma 4.2
$N \ge n^3$	n^3	F_{KR} range size	Section 3.3

Acknowledgements

The authors benefited greatly from discussions with Nicole Wein who took part in the initial stages of this project. The second author also benefited from many discussions on edit distance with Mike Saks. We are grateful to Tomasz Kociumaka for providing us with a reference for Proposition 2.1. We thank anonymous reviewers for their comments.

References

[AN20] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 990–1001. IEEE, 2020.

[BES06] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 792–801, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[BGP20] Or Birenzwige, Shay Golan, and Ely Porat. Locally consistent parsing for text indexing in small space. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 607–626. SIAM, 2020.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM.

[BR20] Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *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$^+$18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 979–990, 2018.

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

[CKP19] Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1106–1125. SIAM, 2019.

[CM02] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 667–676, 2002.

[CV86] Richard Cole and Uzi Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing (STOC)*, pages 206–219, 1986.

[FIM$^+$06] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J Strauss, and Rebecca N Wright. Secure multiparty computation of approximations. *ACM transactions on Algorithms (TALG)*, 2(3):435–472, 2006.

[GKLS22] Arun Ganesh, Tomasz Kociumaka, Andrea Lincoln, and Barna Saha. How compression and approximation affect efficiency in string distance measures. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2867–2919, 2022.

[Gra16] Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 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.

[Jow12] Hossein Jowhari. Efficient communication protocols for deciding edit distance. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 648–658, 2012.

[KOR98] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 614–623, 1998.

[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 Journal of Research and Development*, 31(2):249–260, 1987.

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

[Lin87] Nathan Linial. Distributive graph algorithms-global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science,FOCS*, pages 331–335. IEEE Computer Society, 1987.

[Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

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

[MP80] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18 – 31, 1980.

[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 *Combinatorial Pattern Matching, 18th Annual Symposium, CPM*, volume 4580, pages 173–182. Springer, 2007.

[SV94] Süleyman Cenk Sahinalp and Uzi Vishkin. Symmetry breaking for suffix tree construction. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 300–309. ACM, 1994.

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