Small space and streaming pattern matching with k edits

Tomasz Kociumaka, Ely Porat, Tatiana Starikovskaya

Small space and streaming pattern matching with k edits

Tomasz Kociumaka*[1], Ely Porat[2], and Tatiana Starikovskaya†[3]

[1]University of California, Berkeley, USA
kociumaka@berkeley.edu
[2]Bar-Ilan University, Israel
porately@cs.biu.ac.il
[3]DI/ENS, PSL Research University, France
tat.starikovskaya@gmail.com

Abstract

In this work, we revisit the fundamental and well-studied problem of approximate pattern matching under edit distance. Given an integer k, a pattern P of length m, and a text T of length $n \geq m$, the task is to find substrings of T that are within edit distance k from P. Our main result is a streaming algorithm that solves the problem in $\tilde{\mathcal{O}}(k^5)$ space and $\tilde{\mathcal{O}}(k^8)$ amortised time per character of the text, providing answers correct with high probability. (Hereafter, $\tilde{\mathcal{O}}(\cdot)$ hides a $\mathrm{poly}(\log n)$ factor.) This answers a decade-old question: since the discovery of a $\mathrm{poly}(k \log n)$-space streaming algorithm for pattern matching under Hamming distance by Porat and Porat [FOCS 2009], the existence of an analogous result for edit distance remained open. Up to this work, no $\mathrm{poly}(k \log n)$-space algorithm was known even in the simpler semi-streaming model, where T comes as a stream but P is available for read-only access. In this model, we give a deterministic algorithm that achieves slightly better complexity.

Our central technical contribution is a new space-efficient deterministic encoding of two strings, called the greedy encoding, which encodes a set of all alignments of cost $\leq k$ with a certain property (we call such alignments *greedy*). On strings of length at most n, the encoding occupies $\tilde{\mathcal{O}}(k^2)$ space. We use the encoding to compress substrings of the text that are close to the pattern. In order to do so, we compute the encoding for substrings of the text and of the pattern, which requires read-only access to the latter.

In order to develop the fully streaming algorithm, we further introduce a new edit distance sketch parametrised by integers $n \geq k$. For any string of length at most n, the sketch is of size $\tilde{\mathcal{O}}(k^2)$ and it can be computed with an $\tilde{\mathcal{O}}(k^2)$-space streaming algorithm. Given the sketches of two strings, in $\tilde{\mathcal{O}}(k^3)$ time we can compute their edit distance or certify that it is larger than k. This result improves upon $\tilde{\mathcal{O}}(k^8)$-size sketches of Belazzougui and Zhu [FOCS 2016] and very recent $\tilde{\mathcal{O}}(k^3)$-size sketches of Jin, Nelson, and Wu [STACS 2021].

1 Introduction

In the pattern matching problem, given two strings, a *pattern* P of length m and a *text* T of length n, one must find all substrings of the text equal to the pattern. This is a fundamental problem of string processing with a myriad of applications in such fields as computational biology, information retrieval, and signal processing, to mention just a few. However, in many

*Partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

†Partly supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).

applications, retrieving substrings that are exactly equal to the pattern is not enough, and one must search for substrings merely *similar* to the pattern. This task, which is often referred to as *approximate pattern matching problem*, can be formalised in the following way: for each position i in the text, compute the smallest distance d_i between P and any substring of T that ends at position i. In string processing, the two most popular distances are the Hamming distance and the edit distance. Recall that the Hamming distance between two equal-length strings is the number of mismatching pairs of characters of the strings. The edit distance of two strings, not necessarily of equal lengths, is the smallest number of edits (character insertions, deletions, and substitutions) needed to transform one string into the other. Due to its practical importance, the approximate pattern matching problem has been extensively studied in the literature, originally in the classic setting with input strings explicitly stored in memory.

In general, computing Hamming distance is easier and can be considered as a preliminary step towards tackling edit distance. The first solution for approximate pattern matching under the Hamming distance was given by Abrahamson [2] and, independently, Kosaraju [40]; based on the fast Fourier transform, it spends $\mathcal{O}(n\sqrt{m\log m})$ time to compute the Hamming distance between the pattern and all the length-m substrings of the text. Up to date, no algorithms improve upon this time complexity for the general version of approximate pattern matching under Hamming distance, but there are better solutions when one is interested only in the distances not exceeding a given k, a variant known as the *k-mismatch problem*. The first algorithm for the k-mismatch problem was given by Landau and Vishkin [42], who improved the running time to $\mathcal{O}(kn)$ via so-called "kangaroo jumps", a technique utilising the suffix tree to compute the longest common prefix of two suffixes of a string in constant time. This bound was further improved by Amir et al. [3] who showed two solutions, one with running time $\mathcal{O}(n\sqrt{k\log k})$ and one with running time $\tilde{\mathcal{O}}(n+k^3n/m)$-time algorithm[1]. Continuing this line of research, Clifford et al. [14] presented an $\tilde{\mathcal{O}}(n+k^2n/m)$-time algorithm, while Gawrychowski and Uznański [31] demonstrated a smooth trade-off between the latter and the solution of Amir et al. by designing an $\tilde{\mathcal{O}}(n+kn/\sqrt{m})$-time algorithm. Very recently, Chan et al. [11] shaved off most of the polylogarithmic factors and achieved the running time of $\mathcal{O}(n+\min(k^2n/m, kn\sqrt{\log m}/\sqrt{m}))$ at the cost of Monte-Carlo randomization.

For the edit distance, a detailed survey of previous solutions can be found in [49], and here we only discuss the landmark results of the theoretical landscape. For the general variant of the problem, the first algorithm was given by Sellers [56]. The algorithm was based on dynamic programming and used $\mathcal{O}(nm)$ time. Masek and Paterson [46] improved the running time of the algorithm to $\mathcal{O}(nm/\log n)$ via the Four Russians technique. On the lower bound side, it is known that there is no solution with strongly subquadratic time complexity unless the Strong Exponential Time hypothesis [36] is false, even for the binary alphabet [5, 8]. Abboud et al. [1] gave a more precise bound under a weaker assumption: Namely, they showed that even shaving an arbitrarily large polylog factor would imply that NEXP does not have non-uniform NC^1 circuits. Finally, Clifford et al. [15] showed that, in the cell-probe model with the word size $w=1$, any randomised algorithm that computes the edit distances between the pattern and the text online must spend $\Omega(\frac{\sqrt{\log n}}{(\log\log n)^{3/2}})$ expected amortised time per character of the text.

Similarly to the Hamming distance, one can define the threshold variant of approximate pattern matching, which we refer to as approximate pattern matching with k edits. The first algorithm for this variant of the problem was developed by Landau and Vishkin [43]; this by-now classical algorithm solves the problem in $\mathcal{O}(nk)$ time. The current best result was achieved by a series of work [55, 17] with the running time (for some range of parameters) $\mathcal{O}(n+k^4n/m)$. Very recently, Charalampopoulos et al. [12] studied the problem for both

[1]Hereafter, $\tilde{\mathcal{O}}(\cdot)$ hides a factor of $\mathrm{poly}(\log n)$.

distances in the grammar-compressed setting. Their result, in particular, implies an $\mathcal{O}(k^4)$-space and $\mathcal{O}(nk^4)$-time algorithm for the read-only model, where random access to characters in P and T is allowed and one accounts only for the "extra" space, the space required beyond the space needed to store the pattern and the text.

In this work, we focus on developing algorithms for approximate pattern matching with k edits that use as little space as possible. In particular, we consider the streaming model of computation. In this model, we assume that the input arrives as a stream, one character at a time. We define the space complexity of the algorithm to be all the space used, in other words, we cannot store any information about the input without accounting for it.

The field of streaming algorithms for string processing is relatively recent but, because of its practical interest, it has received a lot of attention in the literature. It started with a seminal paper of Porat and Porat in FOCS 2009 [52], who showed streaming algorithms for exact pattern matching and for the k-mismatches problem. The result of Porat and Porat was followed by a series of works on streaming pattern matching [7, 13, 34, 14, 57, 33, 30, 16, 32, 53], search of repetitions in streams [20, 18, 19, 28, 48, 47, 29], and recognising formal languages in streams [45, 24, 25, 22, 23, 27, 4, 21, 26].

All known streaming algorithms for approximate pattern matching under the Hamming distance [52, 14, 16] are based on some rolling hash — a hash on strings of fixed length that can be efficiently updated when we delete the first character of a string and append a new one to the end of the string, such as, for example, the famous Karp–Rabin fingerprint, which allows computing the Hamming distance between two strings or certify that it exceeds k. The current best algorithm was demonstrated by Clifford et al. in SODA 2019 [16]. The algorithm uses only $\mathcal{O}(k \log \frac{m}{k})$ space, which is optimal up to a logarithmic factor, and spends $\mathcal{O}(\log \frac{m}{k}(\sqrt{k \log k} + \log^3 m))$ time per character. The algorithm is necessarily randomised and can err with high probability[2]. In other words, approximate pattern matching under the Hamming distance in the streaming model is essentially fully understood.

On the other hand, for the edit distance there are no small-space solutions, in particular, because there are no rolling hashes that allow to compute the edit distance between strings. When k is small, the state-of-the-art solution [57] uses $\mathcal{O}(k^8 \sqrt{m} \log^6 m)$ space and $\mathcal{O}((k^2\sqrt{m} + k^{13}) \cdot \log^4 m)$ worst-case time per symbol. Again, the algorithm is randomised and outputs all substrings at edit distance at most k from the pattern with high probability. Another interesting result is that of Chakraborty et al. [9], who developed an algorithm for the general version of approximate pattern matching under edit distance in the model where the text is streaming, but the pattern is read-only. They showed a randomised algorithm that, for every position i of the text T, computes the smallest edit distance d_i between P and a suffix of $T[1\,..\,i]$ with constant multiplicative and $m^{8/9}$-additive approximation (in other words, the algorithm returns a number between d_i and $c \cdot d_i + m^{8/9}$, where $c \geq 1$ is a predetermined constant). The algorithm receives the text online and uses $\mathcal{O}(m^{1-1/54})$ extra space, in addition to the space required to store the pattern.

Naturally, a question arises: is the true complexity of streaming approximate pattern matching under edit distance is on par with that of the Hamming distance? In this work, we answer this question affirmatively.

1.1 Our results

The main result of our work is a fully streaming algorithm for approximate pattern matching under the edit distance that uses $\tilde{\mathcal{O}}(k^5)$ space and $\tilde{\mathcal{O}}(k^8)$ amortized time per character of the text (Theorem 7.6). The algorithm is randomised and its answers are correct with high probability.

[2]With high probability means with probability at least $1 - 1/n^c$ for any predefined constant $c > 1$.

As a stepping stone, we also consider a simpler semi-streaming model introduced in [9]. In this model, we assume that the text arrives in a stream, but the pattern is read-only, which means that, at any moment, the algorithm can access any character of the pattern in constant time (but re-writing characters is prohibited). The space complexity of the algorithm is defined as the total space used on top of the read-only memory holding the pattern. In this setting, we show a *deterministic* algorithm for approximate pattern matching under the edit distance that uses $\tilde{\mathcal{O}}(k^5)$ space and $\tilde{\mathcal{O}}(k^6)$ amortized time per character of the text (Theorem 7.5).

Additionally, we design a new sketch for retrieving the exact edit distance (capped with a threshold k) between strings of length at most n (Theorem 3.17). The sketch is of size $\tilde{\mathcal{O}}(k^2)$, and it can be built using a streaming algorithm that costs $\tilde{\mathcal{O}}(nk)$ total time and uses $\tilde{\mathcal{O}}(k^2)$ space. Given the sketches of two strings X, Y, in $\tilde{\mathcal{O}}(k^3)$ time and $\tilde{\mathcal{O}}(k^2)$ space, we can compute the edit distance between X, Y or certify that it is larger than k. The answer is correct with a large constant probability (with standard amplification, we then achieve high probability of success). This improves upon the $\tilde{\mathcal{O}}(k^8)$-size sketches of Belazzougui and Zhang [6] and the $\tilde{\mathcal{O}}(k^3)$-size sketches of Jin, Nelson, and Wu [37] developed independently.

The conceptual contribution of our work is described in the technical overview (Section 3).

2 Preliminaries

We assume an integer alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$ with σ *characters*. A *string* Y is a sequence of characters numbered from 1 to $n = |Y|$. By $Y[i]$ we denote the i-th symbol of Y. For a string Y of length n, we denote its *reverse* $Y[n]Y[n-1]\ldots Y[1]$ by $\overline{Y}$. We define $Y[i\,..\,j]$ to be equal to $Y[i]\ldots Y[j]$ which we call a *fragment* of Y if $i \le j$ and to the empty string ε otherwise. We also use notations $Y[i\,..\,j)$ and $Y(i\,..\,j]$ which naturally stand for $Y[i]\ldots Y[j-1]$ and $Y[i+1]\ldots Y[j]$, respectively. We call a fragment $Y[1]\ldots Y[j]$ *a prefix* of Y and use a simplified notation $Y[..\,i]$, and a fragment $Y[i]\ldots Y[n]$ *a suffix* of Y denoted by $Y[i\,..]$. We say that X is a *substring* of Y if $X = Y[i\,..\,j]$ for some $1 \le i \le j \le |Y|$. The fragment $Y[i\,..\,j]$ is called an *occurrence* of X.

For a string Y, we define Y^m to be the concatenation of m copies of Y. We also define Y^∞ to be an infinite string obtained by concatenating infinitely many of copies of Y. We say that a string X of length x is a *period* of a string T if $X = T[1\,..\,x]$ and $T[i] = T[i+x]$ for all $i = 1, \ldots, |T| - x$. By $\mathsf{per}(T)$ we denote the length of the shortest period of T. The string T is called *periodic* if $2\,\mathsf{per}(T) \le |T|$. For a string $Y \in \Sigma^n$, we define a *forward rotation* $\mathsf{rot}(Y) = Y[2]\cdots Y[n]Y[1]$. In general, a *cyclic rotation* $\mathsf{rot}^s(Y)$ with *shift* $s \in \mathbb{Z}$ is obtained by iterating rot or the inverse operation rot^{-1}. A non-empty string $X \in \Sigma^n$ is *primitive* if it is distinct from its non-trivial rotations, i.e., if $X = \mathsf{rot}^s(X)$ holds only when s is a multiple of n.

We say that a fragment $X[i\,..\,i+\ell)$ is a *previous factor* if $X[i\,..\,i+\ell) = X[i'\,..\,i'+\ell)$ holds for some $i' \in [1\,..\,i)$. The *LZ77 factorization* of X is a factorization $X = F_1 \cdots F_z$ into non-empty *phrases* such tht the jth phrase F_j is the longest previous factor starting at position $1 + |F_1 \cdots F_{j-1}|$; if no previous factor starts there, then F_j consists of a single character. In the underlying *LZ77 representation*, every phrase $F_j = T[j\,..\,j+\ell)$ that is a previous fragment is encoded as (i', ℓ), where $i' \in [1\,..\,i)$ satisfies $X[i\,..\,i+\ell) = X[i'\,..\,i'+\ell)$. The remaining length-1 phrases are represented by the underlying character. We use $\mathsf{LZ}(X)$ to denote the underlying *LZ77 representation* and $|\mathsf{LZ}(X)|$ to denote its size (the number of phrases).

2.1 Edit Distance Alignments

The *edit distance* $\mathsf{ed}(X, Y)$ between two strings X and Y is defined as the smallest number of character insertions, deletions, and substitutions required to transform X to Y. The *Hamming distance* $\mathsf{hd}(X, Y)$ allows substitutions only (and we assume $\mathsf{hd}(X, Y) = \infty$ if $|X| \neq |Y|$).

Definition 2.1. *A sequence* $(x_t, y_t)_{t=1}^m$ *is an* alignment *of* $X, Y \in \Sigma^*$ *if* $(x_1, y_1) = (1, 1)$, $(x_m, y_m) = (|X| + 1, |Y| + 1)$, *and* $(x_{t+1}, y_{t+1}) \in \{(x_t + 1, y_t + 1), (x_t + 1, y_t), (x_t, y_t + 1)\}$ *for* $t \in [1\,..\,m)$.[3]

Given an alignment $\mathcal{A} = (x_t, y_t)_{t=1}^m$ of strings $X, Y \in \Sigma^*$, for every $t \in [1\,..\,m)$:

- If $(x_{t+1}, y_{t+1}) = (x_t + 1, y_t)$, we say that $\mathcal{A}$ *deletes* $X[x_t]$,
- If $(x_{t+1}, y_{t+1}) = (x_t, y_t + 1)$, we say that $\mathcal{A}$ *deletes* $Y[y_t]$,
- If $(x_{t+1}, y_{t+1}) = (x_t + 1, y_t + 1)$, we say that $\mathcal{A}$ *aligns* $X[x_t]$ and $Y[y_t]$, denoted $X[x_t] \sim_{\mathcal{A}} Y[y_t]$. If additionally $X[x_t] = Y[y_t]$, we say that $\mathcal{A}$ *matches* $X[x_t]$ and $Y[y_t]$, denoted $X[x_t] \simeq_{\mathcal{A}} Y[y_t]$. Otherwise, we say that $\mathcal{A}$ *substitutes* $X[x_t]$ for $Y[y_t]$.

The *cost* of an edit distance alignment $\mathcal{A}$ is the total number characters that $\mathcal{A}$ deletes or substitutes. We denote the cost by $\mathsf{cost}_{X,Y}(\mathcal{A})$, omitting the subscript if X, Y are clear from context. The cost of an alignment $\mathcal{A} = (x_t, y_t)_{t=1}^m$ is at least its width $\mathsf{width}(\mathcal{A}) = \max_{t=1}^m |x_t - y_t|$. Observe that $\mathsf{ed}(X, Y)$ can be defined as the minimum cost of an alignment of X and Y. An alignment of X and Y is *optimal* if its cost is equal to $\mathsf{ed}(X, Y)$.

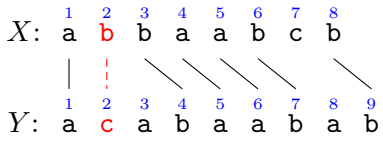
Given an alignment $\mathcal{A} = (x_t, y_t)_{t=1}^m$ of $X, Y \in \Sigma^+$, we partition the elements (x_t, y_t) of $\mathcal{A}$ into *matches* (for which $X[x_t] \simeq_{\mathcal{A}} Y[y_t]$) and *breakpoints* (the remaining elements). We denote the set of matches and breakpoints by $\mathcal{M}_{X,Y}(\mathcal{A})$ and $\mathcal{B}_{X,Y}(\mathcal{A})$, respectively, omitting the subscripts if the strings X, Y are clear from context. Observe that $|\mathcal{B}_{X,Y}(\mathcal{A})| = 1 + \mathsf{cost}(\mathcal{A})$.

We call $M \subseteq [1\,..\,|X|] \times [1\,..\,|Y|]$ a *non-crossing matching* of $X, Y \in \Sigma^*$ if $X[x] = Y[y]$ holds for all $(x, y) \in M$ and there are no distinct pairs $(x, y), (x', y') \in M$ with $x \le x'$ and $y \ge y'$. Note that, for every alignment $\mathcal{A}$ of X, Y, the set $\mathcal{M}(\mathcal{A})$ is a non-crossing matching of X, Y.

Example 2.2. Consider strings $X = \mathtt{abbaabcb}$ and $Y = \mathtt{acabaabab}$ and a cost-4 alignment

$$\mathcal{A} : (1,1), (2,2), (3,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,8), (8,9), (9,10).$$

The breakpoints are $\mathcal{B}(\mathcal{A}) = \{(2,2), (3,3), (7,8), (8,8), (9,10)\}$; the first 4 breakpoints correspond to a substitution of $X[2]$ for $Y[2]$, a deletion of $Y[3]$, a deletion of $X[7]$, and a deletion of $Y[8]$, respectively. Graphically, the alignment is depicted on the right; the aligned pairs of characters are connected with an edge, and the substituted pair is highlighted.



Given an alignment $\mathcal{A} = (x_t, y_t)_{t=1}^m$ of X and Y, for every $\ell, r \in [1\,..\,m]$ with $\ell \le r$, we say that $\mathcal{A}$ *aligns* $X[x_\ell\,..\,x_r)$ and $Y[y_\ell\,..\,y_r)$, denoted $X[x_\ell\,..\,x_r) \sim_{\mathcal{A}} Y[y_\ell\,..\,y_r)$. If there is no breakpoint (x_t, y_t) with $t \in [\ell\,..\,r)$, we further say that $\mathcal{A}$ *matches* $X[x_\ell\,..\,x_r)$ and $Y[y_\ell\,..\,y_r)$, denoted $X[x_\ell\,..\,x_r) \simeq_{\mathcal{A}} Y[y_\ell\,..\,y_r)$.

An alignment $\mathcal{A} = (x_t, y_t)_{t=1}^m$ of $X, Y \in \Sigma^*$ naturally induces a unique alignment of any two fragments $X[x\,..\,x')$ and $Y[y\,..\,y')$. Formally, the *induced alignment* $\mathcal{A}_{[x\,..\,x'),[y\,..\,y')}$ is obtained by removing repeated entries from $(\max(x, \min(x', x_t)) - x + 1, \max(y, \min(y', y_t)) - y + 1)_{t=1}^m$.

Fact 2.3. *If an alignment* $\mathcal{A}$ *satisfies* $X[x\,..\,x') \sim_{\mathcal{A}} Y[y\,..\,y')$, *then* $|x - x'|, |y - y'| \le \mathsf{width}(\mathcal{A})$ *and* $|(x' - x) - (y' - y)| \le \mathsf{ed}(X[x\,..\,x'), Y[y\,..\,y')) \le \mathsf{cost}(\mathcal{A}_{[x\,..\,x'),[y\,..\,y')}) \le \mathsf{cost}(\mathcal{A})$.

3 Technical Overview

In this section, we provide an overview of our conceptual and technical contribution. Let us start with a formal statement of the pattern matching with k edits problem. We say that

[3] This definition is rather complex, but it is equivalent to the standard definition given in the textbooks. We chose this particular formulation as it allowed us to introduce notions essential for this work in a rigorous way.

$T[\ell\,..\,r]$ is a *k-edit occurrence* of P if $\mathsf{ed}(P, T[\ell\,..\,r]) \leq k$, and we denote the set of the *right* endpoints of the k-edit occurrences of P in T by $\mathsf{OCC}_k^E(P, T)$.

Problem 3.1 (Pattern matching with k edits). Given a pattern P of length m over an alphabet Σ, a text T of length n over Σ, and an integer k, compute $\mathsf{OCC}_k^E(P, T)$.

We solve an *online* version of the problem, where the text arrives in a stream (character by character) and the algorithm must decide whether $r \in \mathsf{OCC}_k^E(P, T)$ while processing $T[r]$. The pattern is preprocessed in advance (consistently with [52], in the current version of this paper we do not account for this preprocessing in the complexity analysis). We consider two settings:

1. In the *streaming* setting, the algorithm can no longer access P or $T[1\,..\,r)$ while processing $T[r]$. In other words, all the information regarding these strings needs to be stored explicitly and accounted for in the space complexity of the algorithm.
2. In the *semi-streaming* setting, the algorithm can no longer access $T[1\,..\,r)$ while processing $T[r]$, but it is given an oracle providing read-only constant-time access to individual characters of P. This oracle is not counted towards the space complexity of the algorithm.

For the semi-streaming setting, we provide a deterministic solution, whereas our solution for the streaming setting is Monte-Carlo randomized. Both algorithms are designed for the w-bit word RAM model, where $w = \Omega(\log n)$, and integer alphabets $\Sigma = [0\,..\,n^{\mathcal{O}(1)})$.

3.1 (Semi-)Streaming Algorithm for Pattern Matching with k Edits

Our algorithms solve a slightly stronger problem: every element $r \in \mathsf{OCC}_k^E(P, T)$ is augmented with the smallest integer $k' \in [0\,..\,k]$ such that $r \in \mathsf{OCC}_{k'}^E(P, T)$. At a very high-level, we reuse the structure of existing streaming algorithms for exact pattern matching and the k-mismatch problem [52, 7, 14, 16]. Namely, we consider $\mathcal{O}(\log m)$ prefixes $P_i = P[1\,..\,\ell_i]$ of exponentially increasing lengths ℓ_i. The algorithms are logically decomposed into $\mathcal{O}(\log m)$ levels, with the ith level receiving $\mathsf{OCC}_k^E(P_{i-1}, T)$ and producing $\mathsf{OCC}_k^E(P_i, T)$. In other words, the task of the ith level is determine which k-edit occurrences of P_{i-1} can be extended to k-edit occurrences of P_i. When the algorithm processes $T[r]$, the relevant positions $p \in \mathsf{OCC}_k^E(P_{i-1}, T)$ are those satisfying $|r - p - (\ell_i - \ell_{i-1})| \leq k$. Since each $p \in \mathsf{OCC}_k^E(P_{i-1}, T)$ is reported when the algorithm processes $T[p]$, we need a buffer storing the *active* k-edit occurrences of P_{i-1}. We implement it using a recent combinatorial characterization of k-edit occurrences [12], which classifies strings based on the following notion of approximate periodicity:

Definition 3.2 (k-periodic string)**.** *A string X is* k-periodic *if there exists a primitive string Q with $|Q| \leq |X|/128k$ such that the edit distance between X and a prefix of Q^∞ is at most $2k$. We call Q a* k-period *of X.*

The main message of [12] is that only k-periodic strings may have many k-edit occurrences.

Corollary 3.3 (of [12, Theorem 5.1])**.** *Let $X \in \Sigma^m$, $k \in [1\,..\,m]$, and $Y \in \Sigma^n$ with $n \leq 2m$. If X is not k-periodic, then $|\mathsf{OCC}_k^E(X, Y)| = \mathcal{O}(k^2)$.*

In particular, if P_{i-1} is not k-periodic, then it has $\mathcal{O}(k^2)$ active k-edit occurrences. For each active occurrence $p \in \mathsf{OCC}_k^E(P_{i-1}, T)$, we maintain an edit-distance sketch $\mathsf{sk}_k^E(T(p\,..\,r])$ and combine it with a sketch $\mathsf{sk}_k^E(P(\ell_{i-1}\,..\,\ell_i])$ (constructed at preprocessing) in order to derive $\mathsf{ed}(T(p\,..\,r], P(\ell_{i-1}\,..\,\ell_i])$ or certify that this distance exceeds k. Since we have stored the smallest $k' \in [0\,..\,k]$ such that $p \in \mathsf{OCC}_{k'}^E(P, T)$, this lets us check whether any k-edit occurrence of P_{i-1} ending at position p extends to a k-edit occurrence of P_i ending at position r. With existing k-edit sketches [6, 37], this already yields an $\mathrm{poly}(k \log n)$-space implementation in this case.

The difficulty lies in k-periodic strings whose occurrences form *chains*.

Definition 3.4 (Chain of occurrences)**.** *Consider strings $X, Y \in \Sigma^*$ and an integer $k \in \mathbb{Z}_{\geq 0}$. An increasing sequence of positions $p_1, \ldots, p_c$ forms a* chain *of k-edit occurrences of X in Y if:*

1. *there is a* difference *string $D \in \Sigma^*$ such that $D = Y(p_j \,..\, p_{j+1}]$ for $j \in [1\,..\,c)$, and*
2. *there is an integer $k' \in [0\,..\,k]$ such that $p_j \in \mathsf{OCC}^E_{k'}(X, Y) \setminus \mathsf{OCC}^E_{k'-1}(X, Y)$ for $j \in [1\,..\,c]$.*

Corollary 3.5 (of [12, Theorem 5.2, Claim 5.16, Claim 5.17])**.** *Let $X \in \Sigma^m$, $k \in [1\,..\,m]$, and $Y \in \Sigma^n$ with $n \leq 2m$. If X is k-periodic with period Q, then $\mathsf{OCC}^E_k(X, Y)$ can be decomposed into $\mathcal{O}(k^3)$ chains whose differences are of the form $\mathsf{rot}^s(Q)$ with $|m - s| \leq 10k$.*

In the following discussion, assume that P_{i-1} is k-periodic with period Q_{i-1}. Compared to the previous algorithm, we cannot afford maintaining a sketch $\mathsf{sk}^E_k(T(p\,..\,r])$ for all active $p \in \mathsf{OCC}^E_k(P, T)$. If sk^E_k were a rolling sketch (like the k-mismatch sketches of [16]), we would compute $\mathsf{sk}^E_k(D)$ at preprocessing time for all $\mathcal{O}(k)$ feasible chain differences D and then, for any two subsequent positions p_j, p_{j+1} in a chain with difference D, we could use $\mathsf{sk}^E_k(D) = \mathsf{sk}^E_k(T(p_j \,..\, p_{j+1}])$ to transform $\mathsf{sk}^E_k(T(p_j \,..\, r])$ into $\mathsf{sk}^E_k(T(p_{j+1} \,..\, r])$. However, despite extensive research, no rolling edit distance sketch is known, which remains the main obstacle in designing streaming algorithms for approximate pattern matching with k edits.

Our workaround relies on a novel *encoding* $\mathsf{qGR}(X, Y)$ that, for a pair of strings $X, Y \in \Sigma^*$, represents a large class of low-distance edit distance alignments between X, Y. In the preprocessing phase of our algorithm, we build $\mathsf{qGR}(P(\ell_{i-1} \,..\, \ell_i], D^\infty[1\,..\,\ell_i - \ell_{i-1}])$ for every feasible chain difference D. In the main phase, for subsequent positions $p_j \in \mathsf{OCC}^E_k(P_{i-1}, T)$ in a chain with difference D, we aim to build $\mathsf{qGR}(T(p_j \,..\, r], D^\infty[1\,..\,\ell_i - \ell_{i-1}])$ when necessary, i.e., $|r - p_j - (\ell_i - \ell_{i-1})| \leq k$. We then combine the two encodings to derive $\mathsf{qGR}(P(\ell_{i-1} \,..\, \ell_i], T(p_j \,..\, r])$ and $\mathsf{ed}(P(\ell_{i-1} \,..\, \ell_i], T(p_j \,..\, r])$. Except for such *products* (transitive compositions), our encoding supports *concatenations*, i.e., $\mathsf{qGR}(X_1, Y_1)$ and $\mathsf{qGR}(X_2, Y_2)$ can be combined into $\mathsf{qGR}(X_1X_2, Y_1Y_2)$. Consequently, it suffices to maintain $\mathsf{qGR}(T(p_c \,..\, r], D^\infty[1\,..\,r - p_c - k])$ (where p_c is the rightmost element of the chain). When necessary, we prepend $(j - c)$ copies of $\mathsf{qGR}(D, D)$ (merged by doubling) and append $\mathsf{qGR}(\varepsilon, D^\infty(r - p_j - k \,..\, \ell_i - \ell_{i-1}])$ to derive $\mathsf{qGR}(P(\ell_{i-1} \,..\, \ell_i], D^\infty[1\,..\,\ell_i - \ell_{i-1}])$.

In the semi-streaming setting, we extend $\mathsf{qGR}(T(p_c \,..\, r], D^\infty[1\,..\,r - p_c - k])$ one character at a time using read-only random access to D^∞. In the streaming setting, we cannot afford storing D, so we append the entire difference D in a single step and utilize a new edit-distance sketch sk^q that allows retrieving $\mathsf{qGR}(T(r - |D| \,..\, r], D)$ from $\mathsf{sk}^\mathsf{q}(T(r - |D| \,..\, r])$ and $\mathsf{sk}^\mathsf{q}(D)$. The sketch $\mathsf{sk}^\mathsf{q}(D)$ is constructed in the preprocessing phase, whereas $\mathsf{sk}^\mathsf{q}(T(r - |D| \,..\, r])$ is built as the algorithm scans T. Similarly, we can (temporarily) append any of the $\mathcal{O}(k)$ prefixes that of D may arise when $\mathsf{qGR}(T(p_c \,..\, r], D^\infty[1\,..\,r - p_c - k])$ is necessary. A complete presentation of our algorithms is provided in Section 7. Below, we outline the ideas behind our two main conceptual and technical contributions: the encoding $\mathsf{qGR}(\cdot, \cdot)$ and the sketch $\mathsf{sk}^\mathsf{q}(\cdot)$.

3.2 Greedy Alignments and Encodings

Recall that the encoding $\mathsf{qGR}(\cdot, \cdot)$ needs to support the following three operations:

Capped edit distance: given $\mathsf{qGR}(X, Y)$, compute $\mathsf{ed}(X, Y)$ or certify that $\mathsf{ed}(X, Y)$ is large;
Product: given $\mathsf{qGR}(X, Y)$ and $\mathsf{qGR}(Y, Z)$, retrieve $\mathsf{qGR}(X, Z)$;
Concatenation: given $\mathsf{qGR}(X_1, Y_1)$ and $\mathsf{qGR}(X_2, Y_2)$, retrieve $\mathsf{qGR}(X_1X_2, Y_1Y_2)$.

Our encoding is parameterized with a threshold $k \in \mathbb{Z}_+$ such that $\mathsf{ed}(\cdot, \cdot) > k$ is considered large, and the goal is to achieve $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)})$ encoding size. In fact, whenever $\mathsf{ed}(X, Y) > k$, we shall simply assume that $\mathsf{qGR}_k(X, Y)$ is undefined (formally, $\mathsf{qGR}_k(X, Y) = \bot$). Consequently, products and concatenations will require sufficiently large thresholds in the input encodings so that if either of them is undefined, the output encoding is also undefined.

In order to support concatenations alone, we could use so-called *semi-local* edit distances. For now, suppose that we only need to encode pairs of equal-length strings.[4] Through a sequence of concatenations, we may only extend $\mathsf{qGR}_k(X,Y)$ to $\mathsf{qGR}_k(X',Y')$ so that $X = X'[\ell\,..\,r)$ and $Y = Y'[\ell\,..\,r)$. For any alignment $\mathcal{A}'$ of X', Y' with $\mathsf{cost}(\mathcal{A}') \le k$, consider the induced alignment $\mathcal{A} := \mathcal{A}'_{[\ell\,..\,r),[\ell\,..\,r)}$. Note that $\mathcal{A}$ mimics the behavior of $\mathcal{A}'$ except that it deletes some characters at the extremes of X and Y (which $\mathcal{A}'$ aligns outside Y and X, respectively). By Fact 2.3, we have $\mathsf{cost}(\mathcal{A}) \le k$ and, in particular, $\mathcal{A}$ deletes at most k characters at the extremes of X and Y. If, after performing these deletions, we replace $\mathcal{A}$ with an optimal alignment between the remaining fragments of X and Y, this modification may only decrease $\mathsf{cost}(\mathcal{A})$ and $\mathsf{cost}(\mathcal{A}')$. Consequently, it suffices to store the $\mathcal{O}(k)$ characters at the extremes of X, Y and the $\mathcal{O}(k^4)$ edit distances[5] between long fragments of X and Y. In a sense, this encoding represents $\mathcal{O}(k^4)$ alignments between X, Y that are sufficient to derive an optimal alignment of any extension.

The main challenge is to handle products, for which we develop a *greedy encoding* $\mathsf{GR}_k(X,Y)$ that compactly represents the following family $\mathsf{GA}_k(X,Y)$ of *greedy alignments* of X, Y.

Definition 3.6 (Greedy alignment)**.** *We say that an alignment $\mathcal{A}$ of two strings $X, Y \in \Sigma^*$ is* greedy *if $X[x] \neq Y[y]$ holds for every $(x,y) \in \mathcal{B}(\mathcal{A}) \cap ([1\,..\,|X|] \times [1\,..\,|Y|])$. Given $k \ge 0$, we denote by $\mathsf{GA}_k(X,Y)$ the set of all greedy alignments $\mathcal{A}$ of X, Y satisfying $\mathsf{cost}(\mathcal{A}) \le k$.*

Intuitively, whenever a greedy alignment encounters a pair of matching characters $X[x]$ and $Y[y]$, it must (greedily) match these characters (it cannot delete $X[x]$ or $Y[y]$). As stated below, this restriction does not affect the optimal cost. (All claims are proved in Section 5.)

Fact 3.7. *For any two strings $X, Y \in \Sigma^*$, there is an optimal greedy alignment of X, Y.*

For strings $X, Y \in \Sigma^*$ and an integer $k \ge \mathsf{ed}(X,Y)$, we define a set $\mathcal{M}_k(X,Y)$ of *common matches* of all alignments $\mathcal{A} \in \mathsf{GA}_k(X,Y)$; formally $\mathcal{M}_k(X,Y) = \bigcap_{\mathcal{A} \in \mathsf{GA}_k(X,Y)} \mathcal{M}_{X,Y}(\mathcal{A})$. In our greedy encoding, we shall mask out all the characters involved in the common matches. Below, this transformation is defined for an arbitrary non-crossing matching of X, Y.

Definition 3.8. *Let M be a non-crossing matching of strings $X, Y \in \Sigma^*$. We define X^M, Y^M to be the strings obtained from X, Y by replacing $X[x]$ and $Y[y]$ with $\# \notin \Sigma$ for every $(x,y) \in M$. We refer to $\#$ as a* dummy symbol *and to maximal blocks of $\#$'s as* dummy segments*.*

The following lemma proves that masking out common matches does not affect $\mathsf{ed}(X,Y)$ or $\mathcal{M}_k(X,Y)$ provided that we *enumerate* the dummy symbols, that is, any string Z is transformed to $\mathsf{num}(Z)$ by replacing the ith leftmost occurrence of $\#$ with a unique symbol $\#_i \notin \Sigma$.

Lemma 3.9. *Consider strings $X, Y \in \Sigma^*$, an integer $k \ge \mathsf{ed}(X,Y)$, and a set $M \subseteq \mathcal{M}_k(X,Y)$. Then, $\mathsf{ed}(X,Y) = \mathsf{ed}(\mathsf{num}(X^M), \mathsf{num}(Y^M))$ and $\mathcal{M}_k(X,Y) = \mathcal{M}_k(\mathsf{num}(X^M), \mathsf{num}(Y^M))$.*

At the same time, after masking out the common matches, the strings become compressible. Intuitively, this is because once two greedy alignments converge, they stay together until they encounter a mismatch. Moreover, when two alignments proceed in parallel without any mismatch, this incurs a small period (at most $2k$) that is captured by the LZ factorization.

Lemma 3.10. *Let $M = \mathcal{M}_k(X,Y)$ for strings $X, Y \in \Sigma^*$ and a positive integer $k \ge \mathsf{ed}(X,Y)$. Then, $|\mathsf{LZ}(X^M)|, |\mathsf{LZ}(Y^M)| = \mathcal{O}(k^2)$, and X^M, Y^M contain $\mathcal{O}(k)$ dummy segments.*

[4] Pairs of strings of any lengths can be supported in the same way provided that concatenations require larger input thresholds (compared to the output threshold) to accommodate length differences.

[5] In fact, $\mathcal{O}(k^2)$ edit distances suffice and they can be encoded in $\mathcal{O}(k)$ space using techniques of Tiskin [58].

Consequently, for $k \geq \mathsf{ed}(X,Y)$, we could define the *greedy encoding* $\mathsf{GR}_k(X,Y)$ so that it consists of $\mathsf{LZ}(X^M)$ and $\mathsf{LZ}(Y^M)$. Instead, we use a more powerful compressed representation (developed in Section 4) that supports more efficient queries concerning X^M and Y^M.

Even though $\mathsf{GR}_k(X,Y)$ is small, $\mathsf{GA}_k(X,Y)$ may consist of $2^{\Theta(k)}$ alignments, which is why constructing $\mathsf{GR}_k(X,Y)$ in $\mathrm{poly}(k)$ time is far from trivial. The following combinatorial lemma lets us obtain an $\mathcal{O}(k^5)$-time algorithm in Section 5.2. Intuitively, the alignments in $\mathsf{GA}_k(X,Y)$ can be interpreted as paths in a directed acyclic graph with $\mathcal{O}(k^5)$ branching vertices.

Lemma 3.11. *For all $X,Y \in \Sigma^*$ and $k \in \mathbb{Z}_+$, the set $\mathcal{B}_k(X,Y) = \bigcup_{\mathcal{A}\in\mathsf{GA}_k(X,Y)} \mathcal{B}(\mathcal{A})$ is of size $\mathcal{O}(k^5)$.*

The reason why $\mathsf{GR}_k(X,Y)$ supports products is that every greedy alignment of X,Z can be interpreted as a product of a greedy alignment of X,Y and a greedy alignment of Y,Z.

Definition 3.12. *Consider strings $X,Y,Z \in \Sigma^*$, an alignment $\mathcal{A}^{X,Y}$ of X,Y, an alignment $\mathcal{A}^{Y,Z}$ of Y,Z, and an alignment $\mathcal{A}^{X,Z}$ of X,Z. We say that $\mathcal{A}^{X,Z}$ is a* product *of $\mathcal{A}^{X,Y}$ and $\mathcal{A}^{Y,Z}$ if, for every $(x,z) \in \mathcal{A}^{X,Z}$, there is $y \in [1\,..\,|Y|+1]$ such that $(x,y) \in \mathcal{A}^{X,Y}$ and $(y,z) \in \mathcal{A}^{Y,Z}$.*

Lemma 3.13. *Consider strings $X,Y,Z \in \Sigma^*$ and $k \in \mathbb{Z}_{\geq 0}$. Every alignment $\mathcal{A}^{X,Z} \in \mathsf{GA}_k(X,Z)$ is a product of alignments $\mathcal{A}^{X,Y} \in \mathsf{GA}_d(X,Y)$ and $\mathcal{A}^{Y,Z} \in \mathsf{GA}_d(Y,Z)$, where $d = 2k + \mathsf{ed}(X,Y)$.*

As a result, $\mathsf{GR}_d(X,Y)$ and $\mathsf{GR}_d(Y,Z)$ contain enough information to derive $\mathsf{GR}_k(X,Z)$. The underlying algorithm propagates the characters of Y stored in $\mathsf{GR}_d(X,Y)$ and $\mathsf{GR}_d(Y,Z)$ along the matchings $\mathcal{M}_d(Y,Z)$ and $\mathcal{M}_d(X,Y)$, respectively. Then, $\mathsf{GR}_k(X,Z)$ is obtained by masking out all the characters corresponding to $\mathcal{M}_k(X,Y)$ (see Section 5.4 for details).

To support concatenations, we extend the family $\mathsf{GA}_k(X,Y)$ of greedy alignments to a family $\mathsf{qGA}_k(X,Y)$ of *quasi-greedy alignments*, which are allowed to delete a prefix of X or Y in violation of Definition 3.6. The *quasi-greedy encoding* $\mathsf{qGR}_k(X,Y)$ is defined analogously to $\mathsf{GR}_k(X,Y)$. Equivalently, $\mathsf{qGA}_k(X,Y)$ can be derived from $\mathsf{GA}_{k+1}(\$_1X,\$_2Y)$, where $\$_1 \neq \$_2$ are sentinel symbols outside Σ, by taking the alignments induced by X,Y. The latter characterization makes all our claims regarding GA and GR easily portable to qGA and qGR (see Section 5.3). In particular, this is true for the sketches, described in the following subsection for GR only.

3.3 Edit Distance Sketches

Recall that we need an edit-distance sketch sk^E allowing to retrieve $\mathsf{GR}_k(X,Y)$ from $\mathsf{sk}^E_k(X)$ and $\mathsf{sk}^E_k(Y)$ for any strings $X,Y \in \Sigma^{\leq n}$ and any threshold $k \in [0\,..\,n]$. Furthermore, we need to make sure that $\mathsf{sk}^E_k(S)$ can be computed given streaming access to $S \in \Sigma^{\leq n}$, and that the encoding and decoding procedures use $\mathrm{poly}(k, \log n)$ space. While the existing sketches [6, 37] are designed to compute the exact edit distance $\mathsf{ed}(X,Y)$ capped with k, we believe that they could be adapted to output $\mathsf{qGR}_k(X,Y)$. Nevertheless, these sketches are relatively large (taking $\tilde{\mathcal{O}}(k^8)$ and $\tilde{\mathcal{O}}(k^3)$ bits, respectively), and we would need to strengthen the bulk of their analyses to prove that, in principle, they provide enough information to retrieve $\mathsf{qGR}_k(X,Y)$. Hence, to further demonstrate the power of our techniques, we devise a novel $\tilde{\mathcal{O}}(k^2)$-size sketch specifically designed to output $\mathsf{qGR}_k(X,Y)$. We note that the $\tilde{\mathcal{O}}(k^2)$ size is optimal for $\mathsf{qGR}_k(X,Y)$, but we are not aware of a matching lower bound for retrieving $\mathsf{ed}(X,Y)$ capped with k.

Just like the sketches of [6, 37], ours relies on the embedding of Chakraborty, Goldenberg, and Koucký [10]. The CGK algorithm performs a random walk over the input string (with forward and stationary steps only). In abstract terms, such walk can be specified as follows:

Definition 3.14 (Complete walk)**.** *For a string $S \in \Sigma^*$, we say that $(s_t)_{t=1}^{m+1}$ is an m-*step complete walk over S *if $s_1 = 1$, $s_{m+1} = |S|+1$, and $s_{t+1} \in \{s_t, s_t+1\}$ for $t \in [1\,..\,m]$.*

For any two strings $X, Y \in \Sigma^*$, the two walks underlying $\mathsf{CGK}(X)$ and $\mathsf{CGK}(Y)$ can be interpreted as an edit distance alignment using the following abstract definition:

Definition 3.15 (Zip alignment)**.** *The* zip alignment *of m-step complete walks $(x_t)_{t=1}^{m+1}$ and $(y_t)_{t=1}^{m+1}$ over $X, Y \in \Sigma^*$ is obtained by removing repeated entries in $(x_t, y_t)_{t=1}^{m+1}$.*

The key result of [10] is that the cost of the zip alignment of CGK walks over $X, Y \in \Sigma^*$ is $\mathcal{O}(\mathsf{ed}(X,Y)^2)$ with good probability, which is then exploited to derive a metric embedding (mapping edit distance to Hamming distance) with quadratic distortion. In our sketch, we also need to observe that the CGK alignment is greedy and that its width is $\mathcal{O}(\mathsf{ed}(X,Y))$ with good probability. The following proposition, proved in Section 6.1, provides a complete black-box interface of the properties of the CGK algorithm utilized in our sketches. It also encapsulates Nisan's pseudorandom generator [50] that reduces the number of (shared) random bits.

Proposition 3.16. *For every constant $\delta \in (0,1)$, there exists a constant c and an algorithm W that, given an integer n, a seed r of $\mathcal{O}(\log^2 n)$ random bits, and a string $S \in \Sigma^{\le n}$, outputs a $3n$-step complete walk $\mathsf{W}(n,r,S)$ over S satisfying the following property for all $X, Y \in \Sigma^{\le n}$ and the zip alignment $\mathcal{A}_\mathsf{W}$ of $\mathsf{W}(n,r,X)$ and $\mathsf{W}(n,r,Y)$:*

$$\Pr_r\left[\mathcal{A}_\mathsf{W} \in \mathsf{GA}_{c\cdot\mathsf{ed}(X,Y)^2}(X,Y) \text{ and } \mathsf{width}(\mathcal{A}_\mathsf{W}) \le c\cdot\mathsf{ed}(X,Y)\right] \ge 1-\delta.$$

Moreover, W is an $\mathcal{O}(\log^2 n)$-bit streaming algorithm that costs $\mathcal{O}(n\log n)$ time and reports any element $s_t \in [1\,..\,|S|]$ of $\mathsf{W}(n,r,S)$ while processing the corresponding character $S[s_t]$.

Next, we analyze the structural similarity between $\mathcal{A}_\mathsf{W}$ and any alignment $\mathcal{A} \in \mathsf{GA}_k(X,Y)$. Based on Proposition 3.16, we may assume that $\mathcal{A}_\mathsf{W} \in \mathsf{GA}_{\mathcal{O}(k^2)}(X,Y)$ and $\mathsf{width}(\mathcal{A}_\mathsf{W}) = \mathcal{O}(k)$. Consider the set $M = \mathcal{M}(\mathcal{A}) \cap \mathcal{M}(\mathcal{A}_\mathsf{W})$ of the common matches of $\mathcal{A}$ and $\mathcal{A}_\mathsf{W}$ and the string X^M obtained by masking out the underlying characters of X. Whereas Lemma 3.10 immediately implies that the LZ factorization of X^M consists of $\mathcal{O}(k^4)$ phrases, a more careful application of the same technique provides a refined bound of $\mathcal{O}(k^2)$ phrases. Furthermore, there are $\mathcal{O}(k)$ dummy segments in X^M and, if $X[x]$ is not masked out in X^M (for some $x \in [1\,..\,|X|]$), then there is a breakpoint $(x', y') \in \mathcal{B}_{X,Y}(\mathcal{A}_\mathsf{W})$ with $x' \in [1\,..\,x]$ and $|\mathsf{LZ}(X[x'\,..\,x])| = \mathcal{O}(k)$. Intuitively, this means that $\mathcal{A}$ and $\mathcal{A}_\mathsf{W}$ diverge only within highly compressible regions following the breakpoints $\mathcal{B}_{X,Y}(\mathcal{A}_\mathsf{W})$. We call these regions *forward contexts* (formally, a forward context is the longest fragment starting at a given position and satisfying certain compressiblity condition). Since our choice of $\mathcal{A} \in \mathsf{GA}_k(X,Y)$ was arbitrary, any two alignments $\mathcal{A}, \mathcal{A}' \in \mathsf{GA}_k(X,Y)$ diverge only within these forward contexts. Hence, in order to reconstruct $\mathsf{GR}_k(X,Y)$ and, in particular, $X^{\mathcal{M}_k(X,Y)}$, the sketch should be powerful enough to retrieve all characters in forward contexts of breakpoints $\mathcal{B}_{X,Y}(\mathcal{A}_\mathsf{W})$. Even though $\mathcal{B}_{X,Y}(\mathcal{A}_\mathsf{W})$ could be of size $\Theta(k^2)$, due to the aforementioned bounds on $|\mathsf{LZ}(X^M)|$ and the number of dummy segments in X^M, it suffices to take $\mathcal{O}(k)$ among these forward contexts to cover the unmasked regions of X^M and $X^{\mathcal{M}_k(X,Y)}$. Each context can be encoded in $\tilde{\mathcal{O}}(k)$ bits, so this paves a way towards sketches of size $\tilde{\mathcal{O}}(k^2)$.

Nevertheless, while processing a string $X \in \Sigma^{\le n}$, we only have access to the string X and the m-complete walk $(x_t)_{t=1}^m = \mathsf{W}(n,r,X)$ over X. In particular, depending on Y, any position in X could be involved in a breakpoint. A naive strategy would be to build a *context encoding* $\mathsf{CE}(X)[1\,..\,m]$ that stores at $t \in [1\,..\,m]$ (a compressed representation of) the forward context starting at $X[x_t]$, and then post-process it using a Hamming-distance sketch. This is sufficiently powerful because $X[x_t] \ne Y[y_t]$ holds for any $(x_t, y_t) \in \mathcal{B}(\mathcal{A}_\mathsf{W})$ (recall that $\mathcal{A}_\mathsf{W}$ is greedy). Unfortunately, this construction does not guarantee any upper bound on $\mathsf{hd}(\mathsf{CE}(X), \mathsf{CE}(Y))$ in terms of k. (For example, if X is compressible, modifying its final character of X affects

the entire $\mathsf{CE}(X)$.) Hence, we sparsify $\mathsf{CE}(X)$ by placing a blank symbol $\perp$ at some positions $\mathsf{CE}(X)[t]$ so that just a few forward contexts stored in $\mathsf{CE}(X)[t]$ cover any single position in X.

This brings two further challenges. First, if $X[x]$ is involved in a breakpoint, then we are only guaranteed that it is covered by some forward context $X[p\,..\,q)$ of $\mathsf{CE}(X)[t]$ (i.e., $x \in [p\,..\,q)$). In particular, the forward context starting at position x could extend beyond $X[x\,..\,q)$. Hence, the string $\mathsf{CE}(X)[t]$ actually stores *double forward contexts* $X[p\,..\,r) = X[p\,..\,q)X[q\,..\,r)$ defined as the concatenation of the forward contexts of $X[p]$ and $X[q]$. We expect this double forward context $X[p\,..\,r)$ to cover the entire forward context of $X[x]$. Unfortunately, this is not necessarily true if we use the Lempel–Ziv factorization to quantify compressibility: we could have $|\mathsf{LZ}(X[x\,..\,r))| < |\mathsf{LZ}(X[q\,..\,r))|$ because $\mathsf{LZ}(\cdot)$ is not monotone. Instead, we use an ad-hoc compressibility measure defined as $\overline{\mathsf{maxLZ}}(S) = \max_{[\ell\,..\,r)\subseteq[1\,..\,|S|]} \mathsf{LZ}(\overline{S[\ell\,..\,r)})$. Here, maximization over substrings guarantees monotonicity whereas reversal helps designing an efficient streaming algorithm constructing contexts (beyond the scope of this overview).

Another challenge is that the sparsification needs to be consistent between $\mathsf{CE}(X)$ and $\mathsf{CE}(Y)$: assuming $\mathsf{ed}(X,Y) \le k$, we should have $\mathsf{hd}(\mathsf{CE}(X),\mathsf{CE}(Y)) = \tilde{\mathcal{O}}(k)$, which also accounts for mismatches between $\perp$ and a stored double forward context. This rules out a naive strategy of covering X from left to right using disjoint forward contexts: any substitution at the beginning of X could then have a cascade of consequences throughout $\mathsf{CE}(X)$. Hence, we opt for a memoryless strategy that decides on $\mathsf{CE}(X)[t]$ purely based on the forward contexts $X[x_{t-1}\,..\,x'_{t-1})$ and $X[x_t\,..\,x'_t)$. For example, we could set $\mathsf{CE}(X)[t] = \perp$ unless the smallest dyadic interval containing $[x_{t-1}\,..\,x'_{t-1})$ differs from the smallest dyadic interval containing $[x_t\,..\,x'_t)$ (a dyadic interval is of the form $[i2^j\,..\,(i+1)2^j)$ for some integers $i, j \ge 0$). With this approach, each position of X is covered by at least one and at most $\mathcal{O}(\log|X|)$ forward contexts. Furthermore, substituting any character in X does not have far-reaching knock-on effects. Unfortunately, insertions and deletions are still problematic as they shift the positions x_t. Thus, the decision concerning $\mathsf{CE}(X)[t]$ should be independent of the numerical value of x_t. Consequently, instead of looking at the smallest dyadic interval containing $[x_t\,..\,x'_t)$, we choose the largest t' such that $[x_t\,..\,x'_t) = [x_t\,..\,x_{t'})$, and we look at the smallest dyadic interval containing $[t\,..\,t')$.

With each forward context $X[x_t\,..\,x'_t)$ retrieved, we also need to determine the value x_t (so that we know which fragment of X we can learn from $\mathsf{CE}(X)[t]$). To avoid knock-on effects, we actually store differences $x_t - x_u$ with respect to the previous index satisfying $\mathsf{CE}(X)[u] \neq \perp$. This completes the intuitive description of the context encoding CE studied in Section 6.2.

Our edit-distance sketch contains the Hamming-distance sketch of $\mathsf{CE}(X)$. For this, we use an existing construction [16], augmented in a black-box manner to support large alphabets (recall that the each forward contexts takes $\tilde{\mathcal{O}}(k)$ bits). Furthermore, to retrieve the starting positions x_t (rather than just the differences $x_t - x_u$), we use a hierarchical Hamming-distance sketch similar to those used in [6, 37]. This way, given sketches of X and Y, we can recover all characters that remain unmasked in $X^{\mathcal{M}_k(X,Y)}$ and $Y^{\mathcal{M}_k(X,Y)}$. The tools of Section 5 are then used to compute $\mathsf{ed}(X,Y)$ (or certify $\mathsf{ed}(X,Y) > k$) and retrieve the greedy encoding $\mathsf{GR}_k(X,Y)$ (see Section 6.4). We summarize the properties of the edit distance sketches below:

Theorem 3.17. *For every constant $\delta \in (0,1)$, there is a sketch sk^E_k (parametrized by integers $n \ge k \ge 1$, an alphabet $\Sigma = [0\,..\,n^{\mathcal{O}(1)})$, and a seed of $\mathcal{O}(\log^2 n)$ random bits) such that:*

(a) The sketch $\mathsf{sk}^E_k(S)$ of a string $\Sigma^{\le n}$ takes $\mathcal{O}(k^2\log^3 n)$ bits. Given streaming access to S, it can be constructed in $\tilde{\mathcal{O}}(nk)$ time using $\tilde{\mathcal{O}}(k^2)$ space.

(b) There exists an $\tilde{\mathcal{O}}(k^2)$-space decoding algorithm that, given $\mathsf{sk}^E_k(X), \mathsf{sk}^E_k(Y)$ for $X, Y \in \Sigma^{\le n}$, with probability at least $1-\delta$ outputs $\mathsf{GR}_k(X,Y)$ and $\min(\mathsf{ed}(X,Y), k+1)$. Retrieving $\mathsf{GR}_k(X,Y)$ costs $\tilde{\mathcal{O}}(k^5)$ time, whereas computing $\min(\mathsf{ed}(X,Y), k+1)$ costs $\tilde{\mathcal{O}}(k^3)$ time.

4 Compressed String Representation

In this section, we develop a data structure that stores a string X in $\mathcal{O}(|\mathsf{LZ}(X)| \log^2 |X|)$ space, supporting various operations in a relatively efficient way (the $\log |X|$ factors are not optimized).

Our data structure can be constructed not only from $\mathsf{LZ}(X)$, but from any *LZ-like representation* describing a factorization $X = F_1 \cdots F_f$ into non-empty phrases such that each phrase F_j with $|F_j| > 1$ is a previous factor (unlike LZ77, the phrases do not need to be a maximal).

Observation 4.1. *Every LZ-like factorization of a string $X \in \Sigma^*$ has at least $|\mathsf{LZ}(X)|$ phrases. Moreover, $|\mathsf{LZ}(X)| \le |\mathsf{LZ}(XY)| \le |\mathsf{LZ}(X)| + |\mathsf{LZ}(Y)|$ holds for all strings $X, Y \in \Sigma^*$.*

Proposition 4.2. *In the w-bit word RAM model, every string $X \in \Sigma^n$ satisfying $w = \Omega(\log n + \log |\Sigma|)$, $z = |\mathsf{LZ}(X)|$, and $\overline{z} = |\mathsf{LZ}(\overline{X})|$ has a representation $\mathsf{D}(X)$ that uses $\mathcal{O}(z \log^2 n)$ space and supports the following queries:*

(a) retrieve $n = |X|$, in $\mathcal{O}(1)$ time;
(b) given $i \in [1\,..\,n]$, retrieve $X[i]$, in $\mathcal{O}(\log n)$ time;
(c) given $i, j \in [1\,..\,n]$, compute $\mathsf{LCE}(X[i\,..\,n], X[j\,..\,n])$, in $\mathcal{O}(\log n)$ time;
(d) given $i, j \in [1\,..\,n]$, compute $\mathsf{LCE}(\overline{X}[i\,..\,n], \overline{X}[j\,..\,n])$, in $\mathcal{O}(\log n)$ time;
(e) compute $\mathsf{LZ}(X)$, in $\mathcal{O}(z \log^4 n)$ time;
(f) compute $\mathsf{LZ}(\overline{X})$, in $\mathcal{O}(\overline{z} \log^4 n)$ time;
(g) given $i, j \in [1\,..\,n]$ with $i \le j$, compute $\mathsf{D}(X[i\,..\,j])$, in $\mathcal{O}(z \log^4 n)$ time.

Moreover, $\mathsf{D}(X)$ can be constructed:

(h) in $\mathcal{O}(f \log^2 n)$ time given an f-phrase LZ-like representation of X or $\overline{X}$;
(i) in $\mathcal{O}(z \log^4 n)$ time given $\mathsf{D}(X[1\,..\,i))$ and $\mathsf{D}(X[i\,..\,n])$ for some $i \in [1\,..\,n]$.

Before proving Proposition 4.2, we need to introduce several more compression schemes. These concepts are not used any of the subsequent sections.

A *straight-line grammar* is a tuple $\mathcal{G} = (\mathcal{S}, \Sigma, \mathsf{rhs}, S)$, where $\mathcal{S}$ is a finite set of *symbols*, $\Sigma \subseteq \mathcal{S}$ is a set of *terminal symbols*, $\mathsf{rhs} : (\mathcal{S} \setminus \Sigma) \to \mathcal{S}^*$ is the *production* (or *right-hand side*) function, and $S \in \mathcal{S}$ is the start symbol. We further require existence of an order $\prec$ on $\mathcal{S}$ such that $B \prec A$ if B occurs in $\mathsf{rhs}(A)$.

The *expansion* function $\exp : \mathcal{S} \to \Sigma^+$ is defined recursively:

$$\exp(A) = \begin{cases} A & \text{if } A \in \Sigma, \\ \exp(B_1) \cdots \exp(B_k) & \text{if } \mathsf{rhs}(A) = B_1 \cdots B_k, \end{cases}$$

We say that $\mathcal{G}$ is a *grammar-compressed representation* of $\exp(S)$.

The *parse tree* $\mathcal{T}(A)$ of a symbol $A \in \mathcal{S}$ is a rooted ordered tree with each node ν associated to a symbol $s(\nu) \in \mathcal{S}$. The root of $\mathcal{T}(A)$ is a node ρ with $s(\rho) = A$. If $A \in \Sigma$, then ρ has no children. Otherwise, if $\mathsf{rhs}(A) = B_1 \cdots B_k$, then ρ has k children, and the subtree rooted at the ith child is (a copy of) $\mathcal{T}(B_i)$. The parse tree of a grammar $\mathcal{G}$ is defined as the parse tree $\mathcal{T}(S)$ of the starting symbol S, and the height of $\mathcal{G}$ is defined as the height of $\mathcal{T}(S)$.

Each node ν of $\mathcal{T}(A)$ is associated with an occurrence $\exp(\nu)$ of $\exp(s(\nu))$ in $\exp(A)$. For the root ρ, we define $\exp(\rho) = \exp(A)[1\,..\,|\exp(A)|]$ to be the whole $\exp(A)$. Moreover, if $\exp(\nu) = \exp(A)[\ell\,..\,r)$, $\mathsf{rhs}(s(\nu)) = B_1 \cdots B_k$, and $\nu_1, \ldots, \nu_k$ are the children of ν, then $\exp(\nu_i) = \exp(A)[r_{i-1}\,..\,r_i)$, where $r_i = \sum_{j=1}^{i} |\exp(B_j)|$ for $0 \le i \le k$. This way, the fragments $\exp(\nu_i)$ form a partition of $\exp(\nu)$, and $\exp(\nu_i)$ is indeed an occurrence $\exp(s(\nu_i))$ in $\exp(A)$.

A *straight-line program* (SLP) is a straight-line grammar $\mathcal{G} = (\mathcal{S}, \Sigma, \mathsf{rhs}, S)$ such that $|\mathsf{rhs}(A)| = 2$ holds for each *nonterminal* $A \in \mathcal{S} \setminus \Sigma$. In a *run-length straight-line program* (RLSLP), productions of the form $\mathsf{rhs}(A) = B^k$, with $B \in \mathcal{S}$ and $k \in \mathbb{Z}_{\ge 2}$, are also allowed.

An RLSLP $\mathcal{G}$ of size g (with g symbols) representing a string of length n can be stored in $\mathcal{O}(g)$ space ($\mathcal{O}(g \log(n+g))$ bits) with each non-terminal $A \in \mathcal{S} \setminus \Sigma$ storing $\mathsf{rhs}(A)$ (represented by (B,k) if $\mathsf{rhs}(A) = B^k$) and $|\exp(A)|$. This representation allows for efficiently traversing the parse tree $\mathcal{T}_\mathcal{G}$: given a node ν represented as a pair $(s(\nu), \exp(\nu))$, it is possible to retrieve in constant time an analogous representation of any child of ν.

Fact 4.3. *Let $\mathcal{G}$ be an RLSLP of size g representing a string X.*

(a) An LZ-like factorization of X can be constructed in $\mathcal{O}(g)$ time.

(b) For every fragment $X[i\,..\,j]$, an RLSLP representing $X[i\,..\,j]$ can be constructed in $\mathcal{O}(g)$ time.

Proof. We traverse the parse tree $\mathcal{T}(S)$ in pre-order, skipping some subtrees depending on the application.

(a) For each symbol A, we maintain an already visited node ν with symbol $s(\nu) = A$, if any. When visiting a node ν, we first check whether any node ν' with $s(\nu') = s(\nu)$ has already been visited. If so, we retrieve the expansion $\exp(\nu') = X[i'\,..\,i'+\ell)$, and we output (i', ℓ) as a part of the LZ-like representation. Otherwise, we save ν as an already visited node with symbol $s(\nu)$ and proceed as follows:

- If ν is a leaf, then we output $s(\nu) \in \Sigma$ as a part of the LZ-like representation.
- If ν has two children, we process them recursively.
- If ν has $k \geq 3$ children, we process the first child ν_1 recursively, retrieve $\exp(\nu_1) = X[i\,..\,i+\ell)$, and output $(i, (k-1)\ell)$ as a part of the LZ-like representation.

The overall running time $\mathcal{O}(g)$ is amortized by the number of distinct symbols visited.

(b) Here, an intermediate goal is to construct a sequence $(A_1, m_1), \ldots, (A_t, m_t)$ with $A_p \in \mathcal{S}$ and $m_p \in \mathbb{Z}_{\geq 1}$ such that $\exp(A_1)^{m_1} \cdots \exp(A_t)^{m_t} = X[i\,..\,j]$ and $t = \mathcal{O}(g)$. We only visit nodes ν such that $\exp(\nu)$ intersects $X[i\,..\,j]$.

- If $\exp(\nu)$ is contained in $X[i\,..\,j]$, we simply output $(s(\nu), 1)$ and skip the subtree of ν.
- Otherwise, we determine the children $\nu_\ell, \ldots, \nu_r$ of ν whose expansions intersect $X[i\,..\,j]$. Then, we recurse into ν_ℓ, output $(s(\nu_\ell), r-\ell-1)$ if $r > \ell+1$, and recurse into ν_r if $r > \ell$.

The number of visited nodes is proportional to the height of $\mathcal{G}$ and thus this traversal takes $\mathcal{O}(g)$ time. In the post-processing, for each pair (A_p, m_p), we create a new symbol B_p with $\mathsf{rhs}(B_p) = A_p^{m_p}$ (if $m_p \geq 2$) or set B_p as an alias of A_p (if $m_p = 1$). Next, we set C_1 to be an alias of B_1 and, for $p \in [2\,..\,t]$, create a new symbol C_p with $\mathsf{rhs}(C_p) = C_{p-1}B_{p-1}$. Finally, we return the extended RLSLP with C_t as the new starting symbol. □

Theorem 4.4 (I [35])**.** *Let $\mathcal{G}$ be a size-g SLP representing a string $X \in \Sigma^n$ and let g^* be the minimum size of an SLP representing X. Given $\mathcal{G}$, in $\mathcal{O}(g \log n)$ time, one can construct a size-$\mathcal{O}(g^* \log n)$ RLSLP representing X and a size-$\mathcal{O}(g^* \log n)$ data structure that answers the following queries in $\mathcal{O}(\log n)$ time:*

Access: *Given $i \in [1\,..\,n]$, retrieve $S[i]$;*
LCE queries: *Given $i, j \in [1\,..\,n]$, compute $\mathsf{LCE}(S[i\,..\,n], S[j\,..\,n])$.*

Proof of Proposition 4.2. We define $\mathsf{D}(X)$ so that it consists of two instances of the data structure of Theorem 4.4, one for X and one for $\overline{X}$, including the RLSLPs representing X and $\overline{X}$. Thus, the size of $\mathsf{D}(X)$ is $\mathcal{O}(g^* \log n)$, where g^* is the minimum size of an SLP representing X (it is the same for $\overline{X}$).

To implement (h), we first use [39, Theorem 6.1] and build a size-$\mathcal{O}(f \log n)$ SLP representing X or $\overline{X}$. By reversing the right-hand sides of all the productions, we obtain an analogous SLP representing $\overline{X}$ or X, respectively. Finally, we pass these SLPs to the construction algorithm of Theorem 4.4. Overall, the running time is $\mathcal{O}(f \log^2 n)$.

In (i), given $\mathsf{D}(X[1\,..\,i))$ and $\mathsf{D}(X[i\,..\,n])$, we convert the underlying RLSLPs to LZ-like representations (Fact 4.3), concatenate them into an $\mathcal{O}(g^* \log n)$-phrase LZ-like representation of X, and apply (h). Overall, this takes $\mathcal{O}(g^* \log^3 n) = \mathcal{O}(z \log^4 n)$ time.

As for (a), we note that $|X|$ can be trivially retrieved from the RLSLP representing X. Queries (b), (c), and (d) are implemented directly using Theorem 4.4.

In (e), we construct an index of [39, Theorem 6.11] that, given a fragment $X[i\,..\,i+\ell)$, in $\mathcal{O}(\log^3 n)$ time locates the leftmost position i' with $X[i'\,..\,i'+\ell) = X[i\,..\,i+\ell)$. This index can be constructed in $\mathcal{O}(g^* \log^3 n)$ time: it is already based on the RLSLP of Theorem 4.4, and the extra construction time is $\mathcal{O}(\log^2 n)$ per RLSLP symbol; see [39, Lemma 6.9]. This way, each phrase F_j of the LZ77 parsing of X can be constructed, by binary search, with $\mathcal{O}(\log n)$ queries to the index, for a total of $\mathcal{O}(z \log^4 n)$ time (see also [51, Section 5]). The algorithm for (f) is symmetric.

As for (g), we construct an $\mathcal{O}(g^* \log n)$-phrase LZ-like representation of $X[i\,..\,j]$ using Fact 4.3, and then apply (h). Overall, this takes $\mathcal{O}(g^* \log^3 n) = \mathcal{O}(z \log^4 n)$ time. □

5 Greedy Alignments and Encodings

Definition 3.6 (Greedy alignment)**.** *We say that an alignment $\mathcal{A}$ of two strings $X, Y \in \Sigma^*$ is* greedy *if $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}) \cap ([1\,..\,|X|] \times [1\,..\,|Y|])$. Given $k \geq 0$, we denote by $\mathsf{GA}_k(X, Y)$ the set of all greedy alignments $\mathcal{A}$ of X, Y satisfying $\mathsf{cost}(\mathcal{A}) \leq k$.*

Fact 3.7. *For any two strings $X, Y \in \Sigma^*$, there is an optimal greedy alignment of X, Y.*

Proof. Let $\mathcal{A} = (x_t, y_t)_{t=1}^m$ be an optimal alignment maximizing the sum $\sum_{t=1}^m (x_t + y_t)$. For a proof by contradiction, suppose that $\mathcal{A}$ is not greedy. Then, there exists $i \in [1\,..\,m]$ such that $(x_i, y_i) \in \mathcal{B}(A) \cap ([1\,..\,|X|] \times [1\,..\,|Y|])$ and $X[x_i] = Y[y_i]$. Observe that $x_{i+1} = x_i$ or $y_{i+1} = y_i$ and, by symmetry, assume $x_{i+1} = x_i$ without loss of generality. Let us define $j > i$ so that $x_i = \cdots = x_j < x_{j+1}$ and consider two cases depending on whether $y_j = y_{j+1}$:

- If $y_j = y_{j+1}$, we define an alignment $\mathcal{A}'$ obtained from $\mathcal{A}$ by deleting (x_j, y_j). It is easy to see that $\mathcal{A}'$ is a valid alignment. Moreover, $\mathcal{B}(\mathcal{A}') \subseteq \mathcal{B}(\mathcal{A})$ and $\mathcal{B}(\mathcal{A}) \setminus \mathcal{B}(\mathcal{A}') = \{(x_j, y_j)\}$, so $\mathsf{cost}(\mathcal{A}') < \mathsf{cost}(\mathcal{A})$, contradicting the choice of $\mathcal{A}$ as an optimal alignment.
- If $y_j < y_{j+1}$, we define an alignment $\mathcal{A}' = (x'_t, y'_t)_{t=1}^m$ obtained from $\mathcal{A}$ by incrementing x_t for $t \in (i\,..\,j]$. Moreover, $\mathcal{B}(\mathcal{A}) \setminus \mathcal{B}(\mathcal{A}') \subseteq \{(x_t, y_t) : t \in [i\,..\,j)\}$ and $\mathcal{B}(\mathcal{A}') \setminus \mathcal{B}(\mathcal{A}) = \{(x'_t, y'_t) : t \in (i\,..\,j]\}$, so $\mathsf{cost}(\mathcal{A}') \leq \mathsf{cost}(\mathcal{A})$. Furthermore, $\sum_{t=1}^m (x'_t + y'_t) = j - i + \sum_{t=1}^m (x_t + y_t)$, contradicting the choice of $\mathcal{A}$ as an optimal alignment maximizing $\sum_{t=1}^m (x_t + y_t)$. □

For strings $X, Y \in \Sigma^*$ and an integer $k \geq \mathsf{ed}(X, Y)$, we define a set

$$\mathcal{M}_k(X, Y) = \bigcap_{\mathcal{A} \in \mathsf{GA}_k(X,Y)} \mathcal{M}_{X,Y}(\mathcal{A})$$

of *common matches* of all alignments $\mathcal{A} \in \mathsf{GA}_k(X, Y)$; note that it forms a non-crossing matching of X, Y.

Definition 3.8. *Let M be a non-crossing matching of strings $X, Y \in \Sigma^*$. We define X^M, Y^M to be the strings obtained from X, Y by replacing $X[x]$ and $Y[y]$ with $\# \notin \Sigma$ for every $(x, y) \in M$. We refer to $\#$ as a* dummy symbol *and to maximal blocks of $\#$'s as* dummy segments*.*

Additionally, for a string $Z \in (\Sigma \cup \#)^*$, we define $\mathsf{num}(Z)$ as a string obtained from Z by replacing the ith leftmost occurrence of $\#$ in it with a unique dummy symbol $\#_i \notin \Sigma$.

Fact 5.1. *Let $X, Y \in \Sigma^*$. For each non-crossing matching M of X, Y, we have* $\mathsf{ed}(X,Y) \leq \mathsf{ed}(\mathsf{num}(X^M), \mathsf{num}(Y^M))$.

Proof. For every $(x,y) \in [1\,..\,|X|] \times [1\,..\,|Y|]$, if $\mathsf{num}(X^M)[x] = \mathsf{num}(Y^M)[y]$, then $X[x] = Y[y]$. Thus, $\mathsf{cost}_{X,Y}(\mathcal{A}) \leq \mathsf{cost}_{\mathsf{num}(X^M),\mathsf{num}(Y^M)}(\mathcal{A})$ holds for all alignments $\mathcal{A}$ of X, Y. □

Lemma 3.9. *Consider strings $X, Y \in \Sigma^*$, an integer $k \geq \mathsf{ed}(X,Y)$, and a set $M \subseteq \mathcal{M}_k(X,Y)$. Then,* $\mathsf{ed}(X,Y) = \mathsf{ed}(\mathsf{num}(X^M), \mathsf{num}(Y^M))$ *and* $\mathcal{M}_k(X,Y) = \mathcal{M}_k(\mathsf{num}(X^M), \mathsf{num}(Y^M))$.

Proof. Denote $X' = \mathsf{num}(X^M)$ and $Y' = \mathsf{num}(Y^M)$, and let $\mathcal{A} \in \mathsf{GA}_k(X,Y)$. We shall first prove that $\mathcal{M}_{X,Y}(\mathcal{A}) \subseteq \mathcal{M}_{X',Y'}(\mathcal{A})$. Suppose that $(x,y) \in \mathcal{M}_{X,Y}(\mathcal{A})$, i.e., $X[x] \simeq_{\mathcal{A}} Y[y]$. Note that $\mathcal{M}_{X,Y}(\mathcal{A})$ is a non-crossing matching of X, Y and a superset of M, so either $(x,y) \in M$, and then $X'[x] = Y'[y]$ is a dummy symbol, or M contains no pair involving x or y, and then $X'[x] = X[x] = Y[y] = Y'[y]$. In both cases, we have $X'[x] \simeq_{\mathcal{A}} Y'[y]$, i.e., $(x,y) \in \mathcal{M}_{X',Y'}(\mathcal{A})$. This completes the proof that $\mathcal{M}_{X,Y}(\mathcal{A}) \subseteq \mathcal{M}_{X',Y'}(\mathcal{A})$, from which we derive $\mathcal{B}_{X',Y'}(\mathcal{A}) \subseteq \mathcal{B}_{X,Y}(\mathcal{A})$ and $\mathsf{cost}_{X',Y'}(\mathcal{A}) \leq \mathsf{cost}_{X,Y}(\mathcal{A}) \leq k$.

Next, we shall prove that $\mathcal{A} \in \mathsf{GA}_k(X',Y')$. Suppose that $X'[x] = Y'[y]$ holds for some $(x,y) \in \mathcal{A}$. Then, we have $X[x] = Y[y]$ because either $(x,y) \in M$ or $X[x] = X'[x] = Y'[y] = Y[y]$. Since $\mathcal{A}$ is a greedy alignment of X, Y, this implies $(x,y) \in \mathcal{M}_{X,Y}(\mathcal{A})$, i.e., $X[x] \simeq_{\mathcal{A}} Y[y]$. Due to the assumption $X'[x] = Y'[y]$, we conclude that $X'[x] \simeq_{\mathcal{A}} Y'[y]$, i.e., $(x,y) \in \mathcal{M}_{X',Y'}(\mathcal{A})$. This proves $\mathcal{A} \in \mathsf{GA}_k(X',Y')$.

Now, let $\mathcal{A} \in \mathsf{GA}_k(X',Y')$. We shall first prove that $\mathcal{M}_{X',Y'}(\mathcal{A}) \subseteq \mathcal{M}_{X,Y}(\mathcal{A})$. Suppose that $(x,y) \in \mathcal{M}_{X',Y'}(\mathcal{A})$, i.e., $X'[x] \simeq_{\mathcal{A}} Y'[y]$. In particular, $X'[x] = Y'[y]$, which implies $X[x] = Y[y]$ because either $(x,y) \in M$ or $X[x] = X'[x] = Y'[y] = Y[y]$. Hence, $X[x] \simeq_{\mathcal{A}} Y[y]$, i.e., $(x,y) \in \mathcal{M}_{X,Y}(\mathcal{A})$. This completes the proof that $\mathcal{M}_{X',Y'}(\mathcal{A}) \subseteq \mathcal{M}_{X,Y}(\mathcal{A})$, from which we derive $\mathcal{B}_{X,Y}(\mathcal{A}) \subseteq \mathcal{B}_{X',Y'}(\mathcal{A})$ and $\mathsf{cost}_{X,Y}(\mathcal{A}) \leq \mathsf{cost}_{X',Y'}(\mathcal{A}) \leq k$.

Next, we shall prove that $\mathcal{A} \in \mathsf{GA}_k(X,Y)$. For a proof by contradiction, suppose that $X[x] = Y[y]$ holds for some $(x,y) \in \mathcal{B}_{X,Y}(\mathcal{A})$; if there are several such breakpoints, let us choose the leftmost one. Note that $X[1\,..\,x) \sim_{\mathcal{A}} Y[1\,..\,y)$ and $X[x\,..\,|X|] \sim_{\mathcal{A}} Y[y\,..\,|Y|]$. Let us construct an alignment $\mathcal{A}'$ obtained from $\mathcal{A}$ by replacing the induced alignment $\mathcal{A}_{[x\,..\,|X|],[y\,..\,|Y|]}$ with an optimum greedy alignment of $X[x\,..\,|X|], Y[y\,..\,|Y|]$ (see Fact 3.7). By the choice of (x,y), the induced alignment $\mathcal{A}'_{[1\,..\,x),[1\,..\,y)} = \mathcal{A}_{[1\,..\,x),[1\,..\,y)}$ is a greedy alignment of $X[1\,..\,x), Y[1\,..\,y)$ and thus $\mathcal{A}'$ is a greedy alignment of X, Y. Due to $\mathsf{cost}_{X,Y}(\mathcal{A}') \leq \mathsf{cost}_{X,Y}(\mathcal{A}) \leq k$, this implies $\mathcal{A}' \in \mathsf{GA}_k(X,Y)$. Hence, $\mathcal{M}_{X,Y}(\mathcal{A}')$ is a non-crossing matching of X, Y and a superset of M. The construction of $\mathcal{A}'$ further guarantees $(x,y) \in \mathcal{M}_{X,Y}(\mathcal{A}')$, so either $(x,y) \in M$, and then $X'[x] = Y'[y]$ is a dummy symbol, or M contains no pair involving x or y, and then $X'[x] = X[x] = Y[y] = Y'[y]$. In both cases, we have $X'[x] = Y'[y]$ and, since $\mathcal{A}$ is a greedy alignment of X', Y', we derive $X'[x] \simeq_{\mathcal{A}} Y'[y]$ and $(x,y) \in \mathcal{M}_{X,Y}(\mathcal{A})$. This contradicts the choice of (x,y), completing the proof that $\mathcal{A} \in \mathsf{GA}_k(X,Y)$.

Overall, we conclude that $\mathsf{GA}_k(X,Y) = \mathsf{GA}_k(X',Y')$ and that every alignment $\mathcal{A}$ in this family satisfies both $\mathcal{M}_{X,Y}(\mathcal{A}) = \mathcal{M}_{X',Y'}(\mathcal{A})$ and $\mathsf{cost}_{X,Y}(\mathcal{A}) = \mathsf{cost}_{X',Y'}(\mathcal{A})$. Consequently, $\mathcal{M}_k(X,Y) = \mathcal{M}_k(X',Y')$ and, due to Fact 3.7, $\mathsf{ed}(X,Y) = \mathsf{ed}(X',Y')$. □

Proposition 5.2. *Consider a string $S \in (\Sigma \cup \{\#\})^n$ with s dummy segments. Given the sorted list of dummy segments in S, one can in $\mathcal{O}(s)$ time construct a data structure $\mathsf{RS}_{\#}(S)$ supporting the following queries in $\mathcal{O}(\log s)$ time:*

1. $\mathsf{rank}_{\#}(S,i)$*: Given $i \in [1\,..\,n+1]$, return $|\{i' \in [1\,..\,i) : S[i'] = \#\}|$.*
2. $\mathsf{select}_{\#}(S,j)$*: Given $j \in [1\,..\,\mathsf{rank}_{\#}(S,n)]$, return the jth smallest element of $\{i \in [1\,..\,n] : S[i] = \#\}$.*

Proof. For each dummy segment $S[\ell\,..\,r)$, the data structure stores a tuple $(\ell, r, \mathsf{rank}_\#(S,\ell))$. These tuples are stored in a sorted array (note that the order is the same for all coordinates). To compute $\mathsf{rank}_\#(S,i)$, we binary search for the rightmost segment $S[\ell^*\,..\,r^*)$ such that $\ell^* \le i$. If there is no such segment, we return $\mathsf{rank}_\#(S,i) = 0$. Otherwise, $\mathsf{rank}_\#(S,i) = \mathsf{rank}_\#(S,\ell^*) + \min(i, r^*) - \ell^*$.

To compute $\mathsf{select}_\#(S,j)$, we binary search for the rightmost segment $S[\ell^*\,..\,r^*)$ such that $\mathsf{rank}_\#(S,\ell^*) < j$. Then, $\mathsf{select}_\#(S,j) = \ell^* + j - 1 - \mathsf{rank}_\#(S,\ell^*)$. □

Definition 5.3. *For a non-crossing matching M of strings $X, Y \in \Sigma^*$ we set (cf. Proposition 4.2):*

$$\mathsf{E}^M(X,Y) := \big(\mathsf{D}(X^M Y^M), \mathsf{RS}_\#(X^M), \mathsf{RS}_\#(Y^M)\big)\,.$$

Lemma 5.4. *Let $X' = \mathsf{num}(X^M)$ and $Y' = \mathsf{num}(Y^M)$ for a non-crossing matching M of strings $X, Y \in \Sigma^{\le n}$. The encoding $\mathsf{E}^M(X,Y)$ allows answering LCE queries on the suffixes of X', Y' and on the suffixes of $\overline{X'}, \overline{Y'}$ in $\mathcal{O}(\log n)$ time.*

Proof. Consider a query asking for $\mathsf{LCE}(X'[x\,..\,], Y'[y\,..\,])$. Observe that $\mathsf{LCE}(X'[x\,..\,], Y'[y\,..\,]) = \mathsf{LCE}(X^M[x\,..\,], Y^M[y\,..\,])$ if $\mathsf{rank}_\#(X^M, x) = \mathsf{rank}_\#(Y^M, y)$. Otherwise, $\mathsf{LCE}(X'[x\,..\,], Y'[y\,..\,]) = \min(\mathsf{LCE}(X^M[x\,..\,], Y^M[y\,..\,]), \mathsf{select}_\#(X^M, \mathsf{rank}_\#(X^M, x) + 1) - x, \mathsf{select}_\#(Y^M, \mathsf{rank}_\#(Y^M, y) + 1) - y)$. In either case, the query can be answered in $\mathcal{O}(\log n)$ time using Propositions 5.2 and 4.2(c). The values $\mathsf{LCE}(\overline{X'}[x\,..\,], \overline{Y'}[y\,..\,])$ are answered in a similar way. □

Corollary 5.5. *Given an integer $k > 0$ and the encoding $\mathsf{E}^M(X,Y)$ for a non-crossing matching M of strings $X, Y \in \Sigma^{\le n}$, one can in $\mathcal{O}(k^2 \log n)$ time compute a integer $d \in [0\,..\,k+1]$ such that*

$$d = \begin{cases} \mathsf{ed}(X,Y) & \text{if } \mathsf{ed}(X,Y) \le k \text{ and } M \subseteq \mathcal{M}_k(X,Y), \\ k+1 & \text{if } \mathsf{ed}(X,Y) > k. \end{cases}$$

Proof. We compute $d := \min(k+1, \mathsf{ed}(\mathsf{num}(X^M), \mathsf{num}(Y^M)))$ using the Landau–Vishkin algorithm [42], with $\mathcal{O}(k^2)$ LCE queries on suffixes of $\mathsf{num}(X^M), \mathsf{num}(Y^M)$ implemented using Lemma 5.4. Due to Fact 5.1, we have $d \ge \min(k+1, \mathsf{ed}(X,Y))$ and, in particular, $d = k+1$ if $\mathsf{ed}(X,Y) > k$. If $\mathsf{ed}(X,Y) \le k$ and $M \subseteq \mathcal{M}_k(X,Y)$, then Lemma 3.9 yields $d = \min(k+1, \mathsf{ed}(X,Y)) = \mathsf{ed}(X,Y)$. □

5.1 Greedy Encoding and Its Size

Definition 5.6. *For strings $X, Y \in \Sigma^*$ and an integer k, we define the* greedy encoding

$$\mathsf{GR}_k(X,Y) = \begin{cases} \mathsf{E}^{\mathcal{M}_k(X,Y)}(X,Y) & \text{if } k \ge \mathsf{ed}(X,Y), \\ \bot & \text{otherwise.} \end{cases}$$

Lemma 3.10. *Let $M = \mathcal{M}_k(X,Y)$ for strings $X, Y \in \Sigma^*$ and a positive integer $k \ge \mathsf{ed}(X,Y)$. Then, $|\mathsf{LZ}(X^M)|, |\mathsf{LZ}(Y^M)| = \mathcal{O}(k^2)$, and X^M, Y^M contain $\mathcal{O}(k)$ dummy segments.*

Proof. We show the claim of the lemma for X^M, the claim for Y^M follows by symmetry.

As $\mathsf{ed}(X,Y) \le k$, by Fact 3.7 $\mathsf{GA}_k(X,Y) \ne \emptyset$, and therefore there is an alignment $\mathcal{A} \in \mathsf{GA}_k(X,Y)$ of cost at most k between X and Y. We consider yet another graphical representation of an alignment. Namely, we represent $\mathcal{A}$ as a set of at most k horizontal segments, where a horizontal segment $I = [i\,..\,j; \Delta]$ from (i,Δ) to (j,Δ) means that $X[i\,..\,j] \simeq_{\mathcal{A}} Y[i+\Delta\,..\,j+\Delta]$ (see Fig. 1). This representation induces a partitioning of $X = f_1 \cdots f_z$, where $z = \mathcal{O}(k)$, and each factor f_ℓ is either a single character deleted under $\mathcal{A}$ or a fragment $X[i\,..\,j]$ corresponding to a horizontal segment $I = [i\,..\,j; \Delta]$.

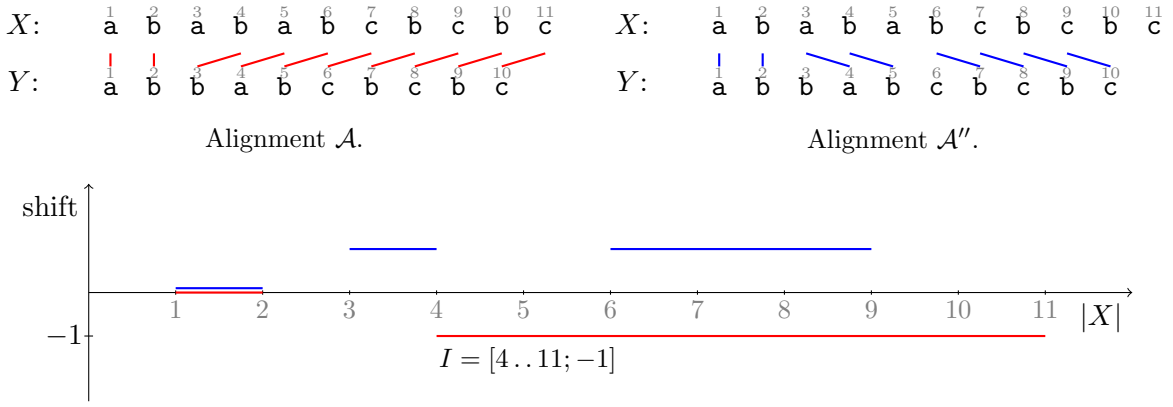


Figure 1: Graphical representations of two greedy alignments $\mathcal{A}, \mathcal{A}'' \in \mathsf{GA}_5(X, Y)$, where $X =$ `abababcbcbc` and $Y =$ `abbabcbcbc`. The alignment $\mathcal{A}$ (red) has cost 1 (we delete $X[3]$), and the alignment $\mathcal{A}''$ cost 5 (we delete $X[5]$, $X[10]$, $X[11]$, $Y[3]$, $Y[6]$). The alignments imply that $X[6\,..\,9] = Y[7\,..\,10] = X[8\,..\,11]$.

If $|\mathsf{GA}_k(X, Y)| = 1$, then every factor f_ℓ with $|f_\ell| \geq 2$ is replaced with an equal length string of dummy symbols. The upper bound on $|\mathsf{LZ}(X^M)|$ follows by sub-additivity of the Lempel–Ziv encoding, and the upper bound on the number of dummy segments is trivial.

Suppose now that $|\mathsf{GA}_k(X, Y)| > 1$. We show that in this case the Lempel–Ziv encoding of any factor f_ℓ has size $\mathcal{O}(k)$. Fix ℓ. If f_ℓ is a single character, the claim obviously holds. Suppose now that the factor f_ℓ of the partitioning defined above corresponds to a horizontal segment $I = [i\,..\,j; \Delta]$. Let us show that there is $i \leq q \leq j$ such that $X^M[i\,..\,q] = X[i\,..\,q]$ and $X^M[q+1\,..\,j] = \#\#\ldots\#$. This follows immediately from the following observation. Let $\mathcal{A}' \neq \mathcal{A}$ be another alignment $\in \mathsf{GA}_k(X, Y)$. The alignment $\mathcal{A}'$ is greedy, and therefore, if for some $i \leq p \leq j$ the alignment $\mathcal{A}'$ matches $X[p]$ and $Y[p+\Delta]$, then it also matches $X[p+1]$ and $Y[p+1+\Delta]$, $\ldots$, $X[j]$ and $Y[j+\Delta]$. We obtain an upper bound on the number of dummy segments as an immediate corollary, but an upper bound on $\mathsf{LZ}(X^M)$ requires more work.

Consider an alignment $\mathcal{A}'' \in \mathsf{GA}_k(X, Y)$ realising q. We have $X^M[q+1\,..\,j] = \#\#\,..\,\#$, and therefore $|\mathsf{LZ}(X^M[q+1\,..\,j])| = \mathcal{O}(1)$. Consider now $X^M[i\,..\,q] = X[i\,..\,q]$. Consider the set of the horizontal segments corresponding to $\mathcal{A}''$ that contain positions in $[i\,..\,q]$. We claim that either all segments in the set have height less than Δ, or all segments have height larger than Δ. Suppose that there are two consecutive horizontal segments $[i_1\,..\,j_1; \Delta_1]$ and $[i_2\,..\,j_2; \Delta_2]$ such that one of them is above I and the other is below. $\mathcal{A}''$ must delete the characters $X[j_1+1], X[j_1+2], \ldots, X[i_2-1]$ and $Y[j_1+\Delta_1], Y[j_1+\Delta_1+1], \ldots, Y[i_2+\Delta_2-1]$ in some order. As it deletes the characters one by one, at some moment it arrives to a pair $X[x], Y[x+\Delta]$, where $i < x < q$. However, since $\mathcal{A}''$ is greedy, from this pair it must follow the segment I, a contradiction.

We now use this property to show that $|\mathsf{LZ}(X[i\,..\,q])| = \mathcal{O}(k)$. First, consider the case when the segments corresponding to $\mathcal{A}''$ are below I. Let $I' = [i'\,..\,j'; \Delta']$ be one such segment, where $i \leq i' \leq j' \leq q$, and $-k \leq \Delta' < \Delta$. Let $i'' = \max\{i + 2k, i'\}$. If $i'' \leq j'$, we have $X[i''\,..\,j'] = Y[i''+\Delta'\,..\,j'+\Delta']$, and the alignment $\mathcal{A}$ implies that $Y[i''+\Delta'\,..\,j'+\Delta'] = X[i''+(\Delta'-\Delta)\,..\,j'+(\Delta'-\Delta)]$. Consequently, $X[i''\,..\,j'] = X[i''+(\Delta'-\Delta)\,..\,j'+(\Delta'-\Delta)]$ is a previous factor as $\Delta'-\Delta < 0$. It follows that $\mathcal{A}''$ defines a partitioning of $X[i\,..\,q] = t_1 \cdots t_z$, where $z = \mathcal{O}(k)$ and each factor t_i is either a single character, or a previous factor. By Fact 4.1, we obtain that $|\mathsf{LZ}(X[i\,..\,q])| = \mathcal{O}(k)$.

The proof for the complementary case is similar. There are at most k horizontal segments

corresponding to $\mathcal{A}'$ that contain positions in $[i\,..\,q]$, denote them by $[i'_\ell\,..\,j'_\ell;\Delta'_\ell]$. For all ℓ, we have $k \geq \Delta'_\ell > \Delta$. Let $j''_\ell = \min\{j'_\ell, q-2k\}$. If $i'_\ell \leq j''_\ell$, we have $X[i'_\ell\,..\,j''_\ell] = Y[i'_\ell+\Delta'_\ell\,..\,j''_\ell+\Delta'_\ell]$ (from the alignment $\mathcal{A}''$), and $Y[i''_\ell+\Delta'_\ell\,..\,j''_\ell+\Delta'_\ell] = X[i'_\ell+(\Delta'_\ell-\Delta)\,..\,j''_\ell+(\Delta'_\ell-\Delta)]$. Consequently, $X[i'_\ell+(\Delta'_\ell-\Delta)\,..\,j''_\ell+(\Delta'_\ell-\Delta)] = X[i'_\ell\,..\,j''_\ell]$ is a previous factor as $\Delta'_\ell-\Delta > 0$. As the cost of $\mathcal{A}''$ is bounded from above by k, we have $\left|[i\,..\,q] - \cup_\ell[i'_\ell\,..\,j''_\ell]\right| \leq k$ and $\sum_\ell |\Delta'_\ell - \Delta'_{\ell+1}| \leq k$. Therefore, $\left|[i\,..\,q] - \cup_\ell[i'_\ell+(\Delta'_\ell-\Delta)\,..\,j''_\ell+(\Delta'_\ell-\Delta))\right| = \mathcal{O}(k)$, and we can conclude as above.

The claim follows. □

Corollary 5.7. *For every $X, Y \in \Sigma^{\leq n}$ and $k \in \mathbb{Z}_+$, the encoding $\mathsf{GR}_k(X,Y)$ takes $\mathcal{O}(k^2\log^2 n)$ space.*

Proof. Follows immediately from Propositions 4.2 and 5.2 and Lemma 3.10. □

5.2 Construction of greedy encodings

In this section, we give a deterministic approach to construction of $\mathsf{GR}_k(X,Y)$. We start with a combinatorial lemma that is a key to this approach.

Lemma 3.11. *For all $X, Y \in \Sigma^*$ and $k \in \mathbb{Z}_+$, the set $\mathcal{B}_k(X,Y) = \bigcup_{\mathcal{A}\in\mathsf{GA}_k(X,Y)} \mathcal{B}(\mathcal{A})$ is of size $\mathcal{O}(k^5)$.*

Proof[6]. We prove $\mathcal{B}_k(X,Y) \leq 136k^5$ by induction on $|X|$. In the base case of $|X| \leq 42k^4$, we have $|\mathcal{B}_k(X,Y)| \leq (2k+1)|X| \leq 136k^5$ since each $(x,y) \in \mathcal{B}_k(X,Y)$ satisfies $|x-y| \leq k$. Thus, we assume that $|X| > 42k^4$, $\mathsf{GA}_k(X,Y) \neq \emptyset$ (otherwise, $\mathcal{B}_k(X,Y) = \emptyset$), and the claim holds for shorter strings.

Case 1. If $\mathcal{M}_k(X,Y)$ contains two adjacent pairs $(x-1,y-1),(x,y)$, then we construct strings X^* and Y^* by deleting $X[x]$ and $Y[y]$, respectively, aiming to prove that $|\mathcal{B}_k(X,Y)| \leq |\mathcal{B}_k(X^*,Y^*)| \leq 136k^5$. For this, consider an alignment $\mathcal{A} \in \mathsf{GA}_k(X,Y)$. Due to $(x-1,y-1),(x,y) \in \mathcal{M}(\mathcal{A})$, the pairs $(x-1,y-1)$, (x,y), and $(x+1,y+1)$ are three consecutive elements of $\mathcal{A}$. Let X^* be the string obtained by removing $X[x]$ from X, and Y^* be the string obtained by removing $Y[y]$ from Y. We define an alignment $\mathcal{A}^*$ of X^*, Y^* by removing (x,y) and shifting all pairs to the right of (x,y) by one position to the left. Each breakpoint $(x',y') \in \mathcal{B}(\mathcal{A})$ is preserved (along with $X[x'] \neq Y[y']$), if it is located to the left of (x,y), or shifted by 1 position to the left (along with $X[x'] \neq Y[y']$), if it is located to the right of (x,y). Consequently, $\mathcal{A}^* \in \mathsf{GA}_k(X^*,Y^*)$ and, since the shifts are independent of $\mathcal{A}$, we derive $|\mathcal{B}_k(X^*,Y^*)| \geq |\mathcal{B}_k(X,Y)|$.

Case 2: If $\mathcal{M}_k(X,Y)$ does not contain any two adjacent pairs, we first prove the following claim:

Claim 5.8. *There are alignments $\mathcal{A}, \mathcal{A}' \in \mathsf{GA}_k(X,Y)$ such that $X[a\,..\,a+9k^2) \simeq_{\mathcal{A}} Y[i\,..\,i+9k^2) \simeq_{\mathcal{A}'} X[b\,..\,b+9k^2)$ holds for some positions $a \neq b$.*

Proof. Consider $k+1$ disjoint fragments of X, each of length $21k^3 \leq |X|/(k+1)$. Each $\mathcal{A} \in \mathsf{GA}_k(X,Y)$ aligns at least one fragment without mismatches: $X[s\,..\,t] \simeq_{\mathcal{A}} Y[s'\,..\,t']$ for some positions s,t,s',t'. Due to $X[t] = Y[t']$, if $(t-1,t'-1) \in \mathcal{M}_k(X,Y)$, then $(t,t') \in \mathcal{M}_k(X,Y)$ holds by the greedy nature of the considered alignments. Consequently, there is $\mathcal{A}' \in \mathsf{GA}_k(X,Y)$ such that $X[t-1] \not\simeq_{\mathcal{A}'} Y[t'-1]$.

Next, consider $k+1$ disjoint parts of $Y[s+k\,..\,t-k)$, each of length $9k^2 \leq (21k^3-1-2k)/(k+1)$. By Fact 2.3, $\mathcal{A}$ aligns the entire $Y[s+k\,..\,t-k)$ without mismatches to a fragment of $X[s\,..\,t)$.

[6]The authors would like to thank Panagiotis Charalampopoulos for an early write-up of the proof of this lemma, which was originally meant to be included into [12].

Moreover, $\mathcal{A}'$ aligns at least one part without mismatches. Denote this part $Y[i\,..\,i+9k^2)$, and fix positions a, b so that $X[a\,..\,a+9k^2) \simeq_{\mathcal{A}} Y[i\,..\,i+9k^2) \simeq_{\mathcal{A}'} X[b\,..\,b+9k^2)$. Note that $a \in [s\,..\,t)$ and thus $X[a\,..\,t) \simeq_{\mathcal{A}} Y[i\,..\,t')$. Now, if $a = b$, then we would have $X[a] \simeq_{\mathcal{A}'} Y[i]$ and the greedy nature of $\mathcal{A}'$ would guarantee $X[a\,..\,t) \simeq_{\mathcal{A}'} Y[i\,..\,t')$, contradicting $X[t-1] \not\simeq_{\mathcal{A}'} Y[t'-1]$. Consequently, $a \neq b$. □

By Fact 2.3, we have $p := \mathsf{per}(Y[i\,..\,i+9k^2)) \leq |a-b| \leq |i-a| + |i-b| \leq 2k$. Moreover, due to $7k^2 \geq 2p$, the fragment $X[i+k^2\,..\,i+8k^2)$ has the same period (up to cyclic rotation).

Claim 5.9. *For every $\mathcal{A}'' \in \mathsf{GA}_k(X,Y)$, we have $X[i+3k^2\,..\,i+8k^2) \simeq_{\mathcal{A}''} Y[v\,..\,v+5k^2)$ for some $v \in [i+2k^2\,..\,i+4k^2]$.*

Proof. Consider $k+1$ disjoint parts of $X[i+k^2\,..\,i+3k^2+p)$, each of length p (note that $(k+1)p \leq 2k^2+p$). Each $\mathcal{A}'' \in \mathsf{GA}_k(X,Y)$ matches at least one part without mismatches: $X[j\,..\,j+p) \simeq_{\mathcal{A}''} Y[u\,..\,u+p)$ for some $j \in [i+k^2\,..\,i+3k^2]$ and $u \in [j-k\,..\,j+k]$. Note that $X[j\,..\,i+8k^2)$ and $Y[u\,..\,i+9k^2)$ have, up to cyclic rotation, the same string period of length p. This string period is primitive, so $X[j\,..\,j+p) = Y[u\,..\,u+p)$ implies that the periods synchronize, i.e., $X[j\,..\,i+8k^2) = Y[u\,..\,u+(i+8k^2-j)]$. By the greedy nature of $\mathcal{A}''$, we must have $X[j\,..\,i+8k^2) \simeq_{\mathcal{A}''} Y[u\,..\,u+(i+8k^2-j)]$, and therefore $X[i+3k^2\,..\,i+8k^2) \simeq_{\mathcal{A}''} Y[v\,..\,v+5k^2]$ for $v = u+i+3k^2-j \in [i+3k^2-k\,..\,i+3k^2+k]$. □

Consider fragments $Q := X[i+3k^2\,..\,i+8k^2)$ and $F := Y[i+4k^2\,..\,i+7k^2)$, where F is contained in all fragments $Y[v\,..\,v+5k^2)$ with $v \in [i+2k^2\,..\,i+4k^2]$. We observe that $|Q| \geq |F| = 3k^2 > p$ and construct X^* and Y^* by removing $X[i+8k^2-p\,..\,i+8k^2)$ and $Y[i+7k^2-p\,..\,i+7k^2)$. For every $\mathcal{A}'' \in \mathsf{GA}_k(X,Y)$, we derive $\mathcal{A}^*$ by removing the pairs (x,y) with $y \in [i+7k^2-p\,..\,i+7k^2)$, and shifting all the subsequent pairs by p positions to the left. We conclude from Claim 5.9 that the breakpoints $(x',y') \in \mathcal{B}(\mathcal{A})$ are preserved (along with $X[x'] \neq Y[y']$), if located to the left of $(i+3k^2, i+4k^2)$, or shifted by p positions to the left (along with $X[x'] \neq Y[y']$), if located to the right of $(i+8k^2, i+7k^2)$. Consequently, $\mathcal{A}^* \in \mathsf{GA}_k(X^*,Y^*)$ and, since the shifts are independent of $\mathcal{A}''$, we derive $|\mathcal{B}_k(X,Y)| \leq |\mathcal{B}_k(X^*,Y^*)| \leq 136k^5$ from the inductive assumption. □

Algorithm 1 is the key procedure for computing greedy encodings.

Lemma 5.10. *Given two strings $X, Y \in \Sigma^*$ and integer $k \geq \mathsf{ed}(X,Y)$, Algorithm 1 computes the dummy segments in $X^{\mathcal{M}_k(X,Y)}$. Moreover, it can be implemented in $\mathcal{O}(|\mathcal{B}_k(X,Y)|(t+\log k))$ time and $\tilde{\mathcal{O}}(k)$ space, provided with an oracle that, given $x \in [1\,..\,|X|+1]$ and $y \in [1\,..\,|Y|+1]$, in $\mathcal{O}(t)$ time computes $\mathsf{LCE}(X[x\,..],Y[y\,..])$ and $\min(k+1, \mathsf{ed}(X[x\,..],Y[y\,..]))$.*

Proof. To show correctness of Algorithm 1, consider an (imaginary set) of alignments $\mathcal{G}$ generated by the algorithm. First, associate with each point in $[1\,..\,|X|] \times [1\,..\,|Y|]$ an empty set of alignments. Let (x,y) be the latest point extracted from Q and $\mathcal{G}[(x,y)]$ the set of alignments associated with it. In line 7 (if the algorithm executes it), we create a new set of alignments containing all alignments in $\mathcal{G}[(x,y)]$ extended by $(x+1,y)$ (deletion of $X[x]$), $(x+1,y), (x+2,y+1), \ldots (x+\ell, y+\ell-1)$, where $\ell = \mathsf{LCE}(X[x+1\,..],Y[y\,..])$ (matching of $X[x+1\,..\,x+\ell]$ and $Y[y\,..\,y+\ell-1]$). We union the resulting set of alignments with $\mathcal{G}[x+\ell, y+\ell-1]$ and push $(x+\ell, y+\ell-1)$ to Q. In lines 9 and 11 we process $\mathcal{G}[(x,y)]$ analogously. Finally, we define $\mathcal{G} = G[|X|,|Y|]$. We claim that $\mathcal{G} = \mathsf{GA}_k(X,Y)$. First note that all alignments in $\mathcal{G}$ are greedy by construction: an alignment can only delete $X[x]$, $Y[y]$, or substitute $X[x]$ for $Y[y]$ if $X[x] \neq Y[y]$. In other words, for every breakpoint (x,y) we have $X[x] \neq Y[y]$. Additionally, the cost of every alignment in $\mathcal{G}$ is at most k as at each call of `GreedyMatch` `credit` decreases by one. Hence, $\mathcal{G} = \mathsf{GA}_k(X,Y)$.

```
1  Greedy(k)
2      Q = ∅;
3      GreedyMatch(Q, 1, 1, k);
4      while Q is not empty do
5          (x, y), credit = Q.extract-min();
6          if x ≤ |X| then
7              GreedyMatch(Q, x + 1, y, credit − 1);                    // delete X[x]
8          if y ≤ |Y| then
9              GreedyMatch(Q, x, y + 1, credit − 1);                    // delete Y[y]
10         if x ≤ |X| and and y ≤ |Y| then
11             GreedyMatch(Q, x + 1, y + 1, credit − 1);        // substitute X[x] for Y[y]
12     GreedyMatch(Q, x, y, credit):
13         if ed(X[x..], Y[y..]) ≤ credit then
14             ℓ = LCE(X[x..], Y[y..]); // match X[x] and Y[y],..., X[x + ℓ − 1] and Y[y + ℓ − 1]
15             x = x + ℓ; y = y + ℓ;
16             Push(Q, (x, y), credit);
17             if |Q| = 1 then output [x − ℓ, x − 1];
18     Push(Q, (x, y), credit)
19         if (x, y) ∉ Q then Q[(x, y)] = credit;
20         else Q[(x, y)] = max{Q[(x, y)], credit};     // points are ordered lexicographically.
```

Algorithm 1: The algorithm receives as an input strings X, Y, and integer k. Lexicographic order on points is defined in the following way: $(x_1, y_1) < (x_2, y_2)$ if either $x_1 < x_2$ or $x_1 = x_2$ and $y_1 < y_2$.

We define successors and predecessors in the lexicographic order in a standard way. For an alignment $\mathcal{A} \in \mathcal{G}$ and a point $(x, y) \in \mathcal{B}(\mathcal{A})$, define $\mathsf{credit}_{\mathcal{A}}(x, y)$ as the difference between k and the number of edits that $\mathcal{A}$ makes to reach (x, y). Algorithm 1 satisfies the following properties:

1. The set $\mathcal{P}$ of the points that are ever added to Q is equal to $\mathcal{B}(\mathcal{G})$, and when we process a point (x, y), the credit associated with it equals $\max_{\mathcal{A} \in \mathcal{G}:(x,y) \in \mathcal{B}(\mathcal{A})} \mathsf{credit}_{\mathcal{A}}(x, y)$;
2. When we process a point (x, y), $Q = \cup_{\mathcal{A} \in \mathcal{G}} \{(x', y')$ is the successor of (x, y) in $\mathcal{B}(\mathcal{A})\}$.
3. If Q contains points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ such that $x_1 - y_1 = x_2 - y_2$, then $p_1 = p_2$.

To show the properties, we exploit the fact that the points are processed in the lexicographical order: When we process a point $q \in Q$, it is the lexicographically smallest point in Q, and all the points that we add to Q while processing q are larger than it.

Property 1. We first prove that $\mathcal{P} \supseteq \mathcal{B}(\mathcal{G})$ and that when we process a point (x, y), the credit associated with it is at least $\max_{\mathcal{A} \in \mathcal{G}:(x,y) \in \mathcal{B}(\mathcal{G})} \mathsf{credit}_{\mathcal{A}}(x, y)$. Let $(x, y) \in \mathcal{B}(\mathcal{G})$ be the lexicographically smallest point which either does not belong to $\mathcal{P}$ or such that the credit associated with it is smaller than $\max_{\mathcal{A} \in \mathcal{G}:(x,y) \in \mathcal{B}(\mathcal{A})} \mathsf{credit}_{\mathcal{A}}(x, y)$. Consider an alignment $\mathcal{A} \in \mathcal{G}$ such that $(x, y) \in \mathcal{B}(\mathcal{A})$ and let (x', y') be the predecessor of (x, y) in $\mathcal{B}(\mathcal{A})$. When we process (x', y'), which happens before we process (x, y) as we process the points in the lexicographic order, the credit associated with it is at least $\mathsf{credit}_{\mathcal{A}}(x', y') \geq \mathsf{ed}(X[x'..], Y[y'..])$, and therefore (x, y) is added to Q with credit at least $\mathsf{credit}_{\mathcal{A}}(x', y') - 1 = \mathsf{credit}_{\mathcal{A}}(x, y)$. As it holds for any $\mathcal{A} \in \mathcal{G}$, we obtain a contradiction.

We now show the opposite direction. Consider a point $(x, y) \in Q$ associated with the edit distance credit c. Suppose that c is achieved when we process a point (x', y'). By the induction assumption, there is a greedy alignment $\mathcal{A} \in \mathcal{G}$ such that the credit $c + 1$ associated with

(x', y') equals $\mathsf{credit}_{\mathcal{A}}(x', y')$. Consider a greedy alignment $\mathcal{A}'$ that behaves as $\mathcal{A}$ until (x', y') $(k - \mathsf{credit}_{\mathcal{A}}(x', y')$ edits), then proceeds to (x', y') (one edit), and ends as an arbitrary optimal greedy alignment between $X[x'..]$ and $Y[y'..]$. The cost of the latter is $\mathsf{ed}(X[x'..], Y[y'..]) \leq c$. Therefore, the cost of $\mathcal{A}'$ is at most k and it is in $\mathcal{G}$. It follows that $(x', y') \in \mathcal{B}(\mathcal{A}')$ and that $c = \mathsf{credit}_{\mathcal{A}'}(x', y')$, finishing the proof.

Property 2. Let (x, y) be the lexicographically smallest point in $\mathcal{B}(\mathcal{G})$ such that when we process it $Q \ni (u, v) \notin \cup_{\mathcal{A} \in \mathcal{G}}\{(x', y')$ is the successor of (x, y) in $\mathcal{B}(\mathcal{A})\}$. Suppose that (u, v) was added to Q when we processed a point (u', v'). Similarly to above, we can construct an alignment $\mathcal{A} \in \mathcal{G}$ such that $(u', v'), (u, v) \in \mathcal{B}(\mathcal{A})$. As we processed (u', v') before (x, y), we must have that (u', v') is lexicographically smaller than (x, y), while (u, v) is lexicographically larger than (x, y), and therefore (u, v) is the successor of (x, y) in $\mathcal{B}(\mathcal{A})$, a contradiction. On the other hand, if (u, v) is the successor of (x, y) in $\mathcal{B}(\mathcal{A})$ for some $\mathcal{A} \in \mathcal{G}$, then it must be in Q when we process (x, y): if (u', v') is the predecessor of (u, v) in $\mathcal{B}(\mathcal{A})$, then (u', v') is at most (x, y), and when we process it, we add (u, v) to Q.

Property 3. We show the property by induction. It is true at initialisation. Suppose that the property is violated when we process a point $(x, y) \in Q$. Recall that we extract (x, y) and call `GreedyMatch` for a subset of points $\{(x + 1, y + 1), (x + 1, y), (x, y + 1)\}$. The point $(x + 1, y + 1)$ is in the same diagonal as (x, y) that we extract from Q, and therefore the call for it does not violate the property. Consider now the point $(x + 1, y)$ (resp., $(x, y + 1)$) and assume that the diagonal containing it also contains a point $(x', y') \in Q$. If we added (x', y') to Q when we called `GreedyMatch` for a point (x'', y''), we have that (x'', y'') is in the same diagonal, $X[x''..x' - 1] = Y[y''..y' - 1]$, and $X[x'] \neq Y[y']$. As we process the points in the lexicographic order, at least one of the points $(x'' - 1, y'' - 1)$, $(x'' - 1, y'')$, $(x'', y'' - 1)$ is strictly smaller than (x, y). By considering each of the cases, from the definition we obtain that (x'', y'') is at most $(x + 1, y)$ (resp., $(x, y + 1)$). As all three points (x'', y''), $(x + 1, y)$ (resp., $(x, y + 1)$), (x', y') are on the same diagonal, it follows that `GreedyMatch` for $(x + 1, y)$ outputs (x', y').

Correctness and complexity. $[x, x']$ is a dummy segment in $X^{\mathcal{M}_k(X,Y)}$ iff there exists $-k \leq \delta \leq k$ such that the following three conditions are satisfied, where $y = x + \delta$ and $y' = x' + \delta$:

1. $\mathcal{B}(\mathcal{G})$ does not contain any points with the x-coordinate in $[x..x']$ or the y-coordinate in $[y..y']$;
2. $\mathcal{B}(\mathcal{G})$ contains $q = (x' + 1, y' + 1)$;
3. The predecessor p of q in $\mathcal{B}(\mathcal{G})$ is one of the points $(x - 1, y - 1), (x - 1, y), (x, y - 1)$.

Therefore, if $[x, x']$ is a maximal dummy segment, then by Property 2, after we have processed p, Q contains a single point q that is obtained by calling `GreedyMatch` for the point (x, y). Hence, the implementation creates the dummy segment $[x, x']$. On the other hand, if the implementation creates a segment $[x, x']$, then $\mathcal{B}(\mathcal{G})$ satisfies all three conditions and hence $[x, x']$ is a maximal dummy segment.

Algorithm 1 implements the set Q as a binary search tree. By Lemma 3.11 and Property 1, we process $|\mathcal{B}(\mathcal{G})| = |\mathcal{B}_k(X, Y)|$ points, spending $\mathcal{O}(\log k + t)$ time per point, which gives the desired time complexity. To show the space complexity, it suffices to show that at any moment $|Q| = \mathcal{O}(k)$. By Fact 2.3, for any $(x, y) \in Q$ we have $y = x + \delta$ for some $-k \leq \delta \leq k$, in other words, the point belongs to one of $2k + 1$ diagonals of the grid. By Property 3, each of the diagonals contains at most one point from Q, which concludes the proof. □

We proceed with an auxiliary claim that allows constructing an edit distance oracle.

Claim 5.11. *Given two strings U, V, and a data structure of size s that can answer the* LCE *queries on the suffixes of $\overline{U}, \overline{V}$ in t time. We can build in $\mathcal{O}(k^2 t)$ time a data structure that occupies $\mathcal{O}(k^2)$ space and can retrieve $\min(k + 1, \mathsf{ed}(U[u..], V[v..]))$ in $\mathcal{O}(\log n)$ time.*

Proof. The data structure is based on the Landau–Myers–Schmidt algorithm [41]. Consider a table D of size $|U| \times |V|$ such that $D[i,j] = \mathsf{ed}(U[|U|-i+1\,..], V[|V|-j+1\,..]) = \mathsf{ed}(\overline{U}[1\,..\,i], \overline{V}[1\,..\,j])$. By Fact 2.3, if $D[i,i+\delta] \le h$, then $-h \le \delta \le h$. Let $L^h(\delta) = \max\{i : D[i,i+\delta] \le h\}$. The data structure consists of $2k+1$ sorted arrays containing $L^0(\delta), L^1(\delta), \ldots, L^k(\delta)$, $-k \le \delta \le k$. The arrays occupy $\mathcal{O}(k^2)$ space and allow to retrieve $\mathsf{ed}(U[u\,..], V[v\,..])$ in $\tilde{\mathcal{O}}(1)$ time by simple binary search. The data structure can be computed via the following recurrence, where $\textsc{Slide}_\delta(u) = u + \mathsf{LCE}(\overline{U}[u\,..], \overline{V}[u+\delta\,..])$.

$$L^h(\delta) = \textsc{Slide}_\delta\left(\max\begin{cases} L^{h-1}(\delta-1), & \delta > -h; \\ L^{h-1}(\delta)+1, & \text{always}; \\ L^{h-1}(\delta+1)+1, & \delta < h. \end{cases}\right) \tag{1}$$

Therefore, given L^{h-1}, $L^h(\delta)$ can be computed via three LCE queries on the suffixes of $\overline{U}, \overline{V}$. As a corollary, the arrays $\{L^0(\delta), L^1(\delta), \ldots, L^k(\delta)\}$, $-k \le \delta \le k$, can be built in $\tilde{\mathcal{O}}(k^2)$ time. □

Proposition 5.12. *Consider a non-crossing matching M of strings $X, Y \in \Sigma^{\le n}$ and an integer $k \in \mathbb{Z}_+$ such that $M \subseteq \mathcal{M}_k(X,Y)$ holds if $\mathsf{ed}(X,Y) \le k$. Given k and $\mathsf{E}^M(X,Y)$, the greedy encoding $\mathsf{GR}_k(X,Y)$ can be computed in $\tilde{\mathcal{O}}(zk + k^2 + |\mathcal{B}_k(X,Y)|)$ time and $\tilde{\mathcal{O}}(z+k^2)$ space, where $z = |\mathsf{LZ}(X^M Y^M)|$.*

Proof. First, we pass k and $\mathsf{E}^M(X,Y)$ to the procedure of Corollary 5.5. Observe that the returned value d satisfies $d \le k$ if and only if $\mathsf{ed}(X,Y) \le k$. If $\mathsf{ed}(X,Y) > k$, then we simply return $\mathsf{GR}_k(X,Y) = \bot$. In this case, the running time and the space complexity are $\tilde{\mathcal{O}}(k^2)$.

Otherwise, our strategy is to compute $\mathcal{M}_k(X,Y)$ and then mask out the corresponding characters of X^M and Y^M to obtain $X^{\mathcal{M}_k(X,Y)}$ and $Y^{\mathcal{M}_k(X,Y)}$. By Lemma 3.9, we have $\mathcal{M}_k(X,Y) = \mathcal{M}_k(X',Y')$, where $X' = \mathsf{num}(X^M)$ and $Y' = \mathsf{num}(Y^M)$. Hence, we shall use Lemma 3.11 to compute the dummy segments corresponding to $\mathcal{M}_k(X',Y')$. For this, we need to support LCE queries and edit distance queries on the suffixes of X', Y'. As for LCE queries, we rely on Lemma 5.4, which provides $\mathcal{O}(\log n)$-time LCE queries on the suffixes of X', Y' and on the suffixes of $\overline{X'}, \overline{Y'}$. We use the latter queries to build a component of Claim 5.11 for the edit distance queries. After $\mathcal{O}(k^2 \log n)$-time preprocessing, these queries can be answered in $\mathcal{O}(\log k)$ time. Overall, constructing the dummy segments corresponding to $\mathcal{M}_k(X',Y')$ costs $\tilde{\mathcal{O}}(k^2 + |\mathcal{B}_k(X,Y)|)$ time and $\mathcal{O}(k^2)$ working space.

Our next goal is to convert $\mathsf{D}(X^M Y^M)$ to $\mathsf{D}(X^{\mathcal{M}_k(X,Y)} Y^{\mathcal{M}_k(X,Y)})$. For this, we need to place #s within each of the computed dummy segments. Consider updating a working string Z by setting $Z[\ell\,..\,r) := \#^{r-\ell}$. To implement this operation, we build $\mathsf{LZ}(\#^{r-\ell})$ (of size at most 2), derive $\mathsf{D}(\#^{r-\ell})$ (Proposition 4.2(h)), extract $\mathsf{D}(Z[1\,..\,\ell))$ and $\mathsf{D}(Z[r\,..\,|Z|])$ (Proposition 4.2(g)), and finally concatenate into $\mathsf{D}(Z[1\,..\,\ell)\#^{r-\ell}Z[r\,..\,|Z|])$ (Proposition 4.2(i)). Overall, each dummy segment is processed in $\mathcal{O}((z+k)\log^4 n)$ time and space, for a total of $\tilde{\mathcal{O}}(zk+k^2)$ time and $\tilde{\mathcal{O}}(z+k)$ space across all segments.

Finally, $\mathsf{RS}_\#(X^{\mathcal{M}_k(X,Y)})$ and $\mathsf{RS}_\#(Y^{\mathcal{M}_k(X,Y)})$ are constructed in $\mathcal{O}(k)$ time and space using Proposition 5.2. □

Corollary 5.13. *Given an integer $k \in \mathbb{Z}_+$ and two strings $X, Y \in \Sigma^{\le k^2}$, one can compute $\mathsf{GR}_k(X,Y)$ in $\tilde{\mathcal{O}}(k^3)$ time and $\tilde{\mathcal{O}}(k^2)$ space.*

Proof. We shall construct $\mathsf{E}^\emptyset(X,Y)$ and then derive $\mathsf{GR}_k(X,Y)$ from Proposition 5.12. Hence, we build a trivial LZ-like representation of XY (with length-1 phrases) and derive $\mathsf{D}(XY) = \mathsf{D}(X^\emptyset Y^\emptyset)$ using Proposition 4.2(h). This costs $\mathcal{O}(k^2 \log^2 k)$ time and space. The components $\mathsf{RS}_\#(X^\emptyset)$ and $\mathsf{RS}_\#(Y^\emptyset)$ are trivial (there are no dummy segments). Using Proposition 5.12 costs $\tilde{\mathcal{O}}(k^3 + |\mathcal{B}_k(X,Y)|)$ time and $\tilde{\mathcal{O}}(k^2)$ space. To finish the proof, note that $\mathcal{B}_k(X,Y)$ can

only contain pairs (x, y) such that $1 \le x \le |X|$, $1 \le y \le |Y|$, and $|x - y| \le k$, which implies $|\mathcal{B}_k(X, Y)| = \mathcal{O}(|X|k) = \mathcal{O}(k^3)$. □

5.3 Concatenations and Quasi-greedy Alignments

Definition 5.14 (Quasi-greedy alignment)**.** *We say that an alignment $\mathcal{A}$ of two strings $X, Y \in \Sigma^*$ is* quasi-greedy *if it satisfies at least one of the following symmetric conditions:*

- *there exists $\delta \in [1\,..\,|X|+1]$ such that $(\delta, 1) \in \mathcal{A}$ and $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}) \cap ([\delta\,..\,|X|] \times [1\,..\,|Y|])$;*
- *there exists $\delta \in [1\,..\,|Y|+1]$ such that $(1, \delta) \in \mathcal{A}$ and $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}) \cap ([1\,..\,|X|] \times [\delta\,..\,|Y|])$;*

Given $k \ge 0$, we denote by $\mathsf{qGA}_k(X, Y)$ the set of all quasi-greedy alignments $\mathcal{A}$ of X, Y with $\mathsf{cost}(\mathcal{A}) \le k$.

Lemma 5.15. *Consider an alignment $\mathcal{A} \in \mathsf{qGA}_k(X_pX_s, Y_pY_s)$, where $X_p, X_s, Y_p, Y_s \in \Sigma^*$ and $k \in \mathbb{Z}_{\ge 0}$. Then, $\mathcal{A}_{[1\,..\,|X_p|+1),[1\,..\,|Y_p|+1)} \in \mathsf{qGA}_{k+||X_s|-|Y_s||}(X_p, Y_p)$ and $\mathcal{A}_{[|X_p|+1\,..\,],[|Y_p|+1\,..\,]} \in \mathsf{qGA}_{k+||X_p|-|Y_p||}(X_s, Y_s)$.*

Proof. For brevity, let $X = X_pX_s$, $Y = Y_pY_s$; $n_p = |X_p|$, $n_s = |X_s|$, $n = |X|$, $m_p = |Y_p|$, $m_s = |Y_s|$, $m = |Y|$.

Let $\mathcal{A} = (x_t, y_t)_{t=1}^{q}$ and let (x_{i_p}, y_{i_p}) be the leftmost pair $(x, y) \in \mathcal{A}$ such that $x > n_p$ or $y > m_p$. By symmetry, we assume without loss of generality that $x_{i_p} = n_p + 1$; see Fig. 2. Observe that $\mathcal{A}_p := \mathcal{A}_{[1\,..\,n_p+1),[1\,..\,m_p+1)}$ is the union of $(x_t, y_t)_{t=1}^{i_p-1}$ and $(n_p + 1, y)_{y=y_{i_p}}^{m_p+1}$. Furthermore, $\mathcal{B}(\mathcal{A}_p) = \{(x_t, y_t) \in \mathcal{B}(\mathcal{A}) : t \in [1\,..\,p)\} \cup \{(n_p + 1, y) : y \in [y_p\,..\,m_p + 1]\}$. Let k_p be the size of the former component. Then, $k \ge \mathsf{cost}(\mathcal{A}) \ge k_p + \mathsf{ed}(X_s, Y_p[y_{i_p}\,..]Y_s) \ge k_p + |Y_p[y_{i_p}\,..]| + m_s - n_s$, so that $\mathsf{cost}(\mathcal{A}_p) = k_p + |Y_p[y_p\,..]| \le k + n_s - m_s \le k - \big||X_s| - |Y_s|\big|$. It remains to prove that $\mathcal{A}_p$ is a quasi-greedy alignment. By Definition 3.6 applied to $\mathcal{A}$, we have two possibilities:

- There exists $\delta \in [1\,..\,n+1]$ such that $(\delta, 1) \in \mathcal{A}$ and $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}) \cap ([\delta\,..\,n] \times [1\,..\,m])$. If $\delta \le n_p + 1$, then we also have that $(\delta, 1) \in \mathcal{A}_p$ and $X_p[x] \neq Y_p[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}_p) \cap ([\delta\,..\,n_p] \times [1\,..\,m_p])$. Otherwise, $(x_{i_p}, y_{i_p}) = (n_p + 1, 1) \in \mathcal{A}_p$, so $\mathcal{A}_p$ is trivially quasi-greedy.
- There exists $\delta \in [1\,..\,m+1]$ such that $(1, \delta) \in \mathcal{A}$ and $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}) \cap ([1\,..\,n] \times [\delta\,..\,m])$. If $\delta \le y_{i_p}$, then we also have that $(1, \delta) \in \mathcal{A}_p$ and $X_p[x] \neq Y_p[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}_p) \cap ([1\,..\,n_p] \times [\delta\,..\,m_p])$. Otherwise, $n_p = 0$ because $(1, \delta)$ cannot cross $(n_p + 1, y_p)$. Hence, $\mathcal{A}_p$ is trivially quasi-greedy.

Now, let (x_{i_s}, y_{i_s}) be the leftmost pair $(x, y) \in \mathcal{A}$ such that $x > n_p$ and $y > m_p$. By symmetry, we assume without loss of generality that $y_{i_s} = m_p + 1$; see Fig. 2. Observe that $\mathcal{A}_s := \mathcal{A}_{[n_p+1\,..\,],[m_p+1\,..\,]}$ is the union of $(x, 1)_{x=1}^{x_{i_s}-n_p-1}$ and $(x_t - n_p, y_t - m_p)_{t=i_s}^{q}$. Furthermore, $\mathcal{B}(\mathcal{A}_s) = \{(x_t - n_s, y_t - m_s) \in \mathcal{B}(\mathcal{A}) : t \in [i_s\,..\,q]\} \cup \{(x, 1) : x \in [1\,..\,x_{i_s} - n_p)\}$. Let k_s be the size of the former component. Then, $k \ge \mathsf{cost}(\mathcal{A}) \ge k_s - 1 + \mathsf{ed}(X_pX_s[1\,..\,x_{i_s} - n_p), Y_p) \ge k_s - 1 + x_{i_s} - m_p$, so that $\mathsf{cost}(\mathcal{A}_s) = k_s - 1 + x_{i_s} - n_p \le k + m_p - n_p \le k - \big||X_p| - |Y_p|\big|$. It remains to prove that $\mathcal{A}_s$ is a quasi-greedy alignment. By Definition 3.6 applied to $\mathcal{A}$, we have two possibilities:

- There exists $\delta \in [1\,..\,n+1]$ such that $(\delta, 1) \in \mathcal{A}$ and $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}) \cap ([\delta\,..\,n] \times [1\,..\,m])$. If $\delta \le x_{i_s}$, then $(x_{i_s} - n_p, 1) \in \mathcal{A}_s$ and $X_s[x] \neq Y_s[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}_s) \cap ([x_{i_s} - n_p\,..\,n_s] \times [1\,..\,m_s])$. Otherwise, $(\delta - n_p, 1) \in \mathcal{A}_s$ and $X_s[x] \neq Y_s[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}_s) \cap ([\delta - n_p\,..\,n_s] \times [1\,..\,m_s])$.
- There exists $\delta \in [1\,..\,m+1]$ such that $(1, \delta) \in \mathcal{A}$ and $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}) \cap ([1\,..\,n] \times [\delta\,..\,m])$. If $\delta \le m_p + 1$, then $(x_{i_s} - n_p, 1) \in \mathcal{A}_s$ and $X_s[x] \neq Y_s[y]$ holds for every $(x, y) \in \mathcal{B}(\mathcal{A}_s) \cap ([x_{i_s} - n_p\,..\,n_s] \times [1\,..\,m_s])$. Otherwise, $x_{i_s} = 1$ and $n_p = 0$

because $(1,\delta)$ cannot cross (x_{i_s}, m_p+1). Hence, $(1,\delta-m_p)\in\mathcal{A}$ and $X_s[x]\neq Y_s[y]$ holds for every $(x,y)\in\mathcal{B}(\mathcal{A}_s)\cap([1\,..\,n_s]\times[\delta-m_p\,..\,m_s])$. □

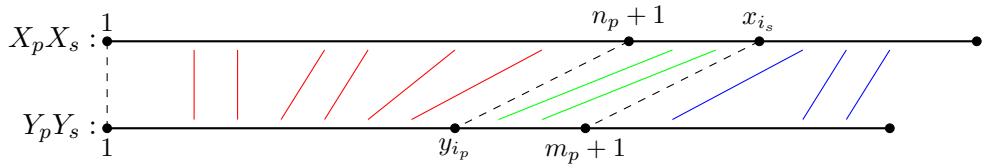


Figure 2: Alignment $\mathcal{A}\in\mathsf{qGA}_k(X_pX_s,Y_pY_s)$.

For strings $X,Y\in\Sigma^*$ and an integer $k\geq\mathsf{ed}(X,Y)$, we define a set

$$\mathsf{q}\mathcal{M}_k(X,Y)=\bigcap_{\mathcal{A}\in\mathsf{qGA}_k(X,Y)}\mathcal{M}_{X,Y}(\mathcal{A})$$

of *common matches* of all alignments $\mathcal{A}\in\mathsf{qGA}_k(X,Y)$.

Definition 5.16. *For strings $X,Y\in\Sigma^*$ and an integer k, we define the* quasi-greedy encoding

$$\mathsf{qGR}_k(X,Y)=\begin{cases}\mathsf{E}^{\mathsf{q}\mathcal{M}_k(X,Y)}(X,Y) & \text{if } \mathsf{ed}(X,Y)\leq k,\\ \bot & \text{otherwise.}\end{cases}$$

Corollary 5.17. *Consider strings $X,Y\in\Sigma^{\leq n}$. Define $X'=\$_1X$ and $Y'=\$_2Y$, where $\$_1\neq\$_2$ are special symbols not in Σ. Let $M'=\mathcal{M}_{k+1}(X',Y')$ and $M=\mathsf{q}\mathcal{M}_k(X,Y)$. The following properties hold:*

- *(a) Given the dummy segments for one of the strings $X^M,(X')^{M'}$, the dummy segments for the other string can be constructed in $\mathcal{O}(k)$ time and space;*
- *(b) Given one of the encodings $\mathsf{qGR}_k(X,Y),\mathsf{GR}_{k+1}(X',Y')$, the other encoding can be constructed in $\mathcal{O}(k^2\log^4 n)$ time and space;*
- *(c) The quasi-greedy encoding $\mathsf{qGR}_k(X,Y)$ occupies $\mathcal{O}(k^2\log^4 n)$ space.*

Proof. Let us first show that $\mathcal{A}'\mapsto\mathcal{A}'_{[2..],[2..]}$ bijectively maps $\mathsf{GA}_{k+1}(X',Y')$ to $\mathsf{qGA}_k(X,Y)$.

Consider a greedy alignment $\mathcal{A}'\in\mathsf{GA}_{k+1}(X',Y')$ and let $\mathcal{A}=\mathcal{A}'_{[2..],[2..]}$. Lemma 5.15 yields $\mathcal{A}\in\mathsf{qGA}_{k+1}(X,Y)$. However, we actually have $\mathsf{cost}(\mathcal{A})\leq k$ because $\mathcal{M}(\mathcal{A}')=\{(x,y):(x+1,y+1)\in\mathcal{M}(\mathcal{A})\}$ holds due to $X'[1]\neq Y'[1]$.

We now show the opposite direction. Let $\mathcal{A}\in\mathsf{qGA}_k(X,Y)$. Suppose that there is $\delta\in[1\,..\,|Y|]$ such that $(1,\delta)\in\mathcal{A}$ and $X[x]\neq Y[y]$ holds for every $(x,y)\in\mathcal{B}(\mathcal{A})\cap([1\,..\,|X|]\times[\delta\,..\,|Y|])$ (the other case is symmetrical). It follows in particular that $\mathcal{A}$ contains elements $\{(1,1),(1,2),\ldots,(1,\delta)\}$. We start by replacing each element (x,y) of $\mathcal{A}$ with $(x+1,y+1)$. We add an element $(1,1)$ to $\mathcal{A}$ and replace each element $(2,i)$, $2\leq i\leq\delta+1$, with $(1,i)$. Finally, we add an element $(2,\delta+1)$. We claim that the resulting alignment $\mathcal{A}'$, that we treat as an alignment of X' and Y', is greedy. For every element $(x,y)\in\mathcal{B}(\mathcal{A}')\cap([2\,..\,|X'|]\times[\delta+1\,..\,|Y'|])$ we have $(x-1,y-1)\in\mathcal{B}(\mathcal{A})\cap([1\,..\,|X|]\times[\delta\,..\,|Y|])$ and hence from quasi-greediness of $\mathcal{A}$ we obtain $X'[x]=X[x-1]\neq Y[y-1]=Y'[y]$. For every element $(1,i)\in\mathcal{A}'$ we have $X'[1]=\$_1\neq Y[i]$. It follows that $\mathcal{A}'$ is greedy. In addition, the cost of $\mathcal{A}'$ equals the cost of $\mathcal{A}$ plus one and hence is bounded by $k+1$. Finally, note that $\mathcal{M}(\mathcal{A}')=\{(x,y):(x-1,y-1)\in\mathcal{M}(\mathcal{A})\}$.

Let $M'=\mathcal{M}_{k+1}(X',Y')$ and $M=\mathsf{q}\mathcal{M}_k(X,Y)$. From above, we obtain $M'=\{(x+1,y+1):(x,y)\in M\}$. Therefore, $(X')^{M'}[2\,..\,]=X^M$ and $(Y')^{M'}[2\,..\,]=Y^M$.

It follows that we can obtain the dummy segments for X^M by subtracting one from the endpoints of each dummy segment of $(X')^{M'}$, and analogously for Y^M. The reverse claim is obtained analogously, by adding one to the endpoints. As the number of the dummy segments in X^M and $(X')^{M'}$ is $\mathcal{O}(k)$ by Lemma 3.10, (a) follows.

From (a) it follows that given the rank data structures for one pair of strings X^M, Y^M and $(X')^{M'}, (Y')^{M'}$, the rank data structures for the other pair can be built in $\mathcal{O}(k)$ time and space. We now must show that given the D data structure for one of the two strings $X^M Y^M, (X')^{M'}(Y')^{M'}$, we can construct the D data structure for the other string efficiently. Assume that we are given $\mathsf{D}((X')^{M'}(Y')^{M'})$, the other case is analogous. We extract all non-dummy segments using Proposition 4.2(g) and concatenate them using Proposition 4.2(i). In total, it takes $\mathcal{O}(k^2 \log^4 n)$ time and space, giving (b).

(c) follows immediately from (b). □

From Corollaries 5.13 and 5.17 it immediately follows that:

Claim 5.18. *Given an integer $k \in \mathbb{Z}_+$ and strings $X, Y \in \Sigma^{\leq k^2}$, one can compute $\mathsf{qGR}_k(X, Y)$ in $\tilde{\mathcal{O}}(k^3)$ time and $\tilde{\mathcal{O}}(k^2)$ space.*

Corollary 5.19. *Consider a non-crossing matching M of strings $X, Y \in \Sigma^{\leq n}$ and an integer $k \in \mathbb{Z}_+$ such that $M \subseteq \mathsf{q}\mathcal{M}_k(X, Y)$ if $\mathsf{ed}(X, Y) \leq k$. Given k and $\mathsf{E}^M(X, Y)$, the quasi-greedy encoding $\mathsf{qGR}_k(X, Y)$ can be computed in $\tilde{\mathcal{O}}(k^5 + zk)$ time and $\tilde{\mathcal{O}}(k^2 + z)$ space, where $z = |\mathsf{LZ}(X^M Y^M)|$.*

Proof. Let $\$_1 \neq \$_2$ be auxiliary symbols not in $\Sigma \cup \{\#\}$. We construct $M' = \{(x+1, y+1) : (x, y) \in M\}$ and $\mathsf{E}^{M'}(\$_1 X, \$_2 Y)$. Note that M' is a non-crossing matching of $\$_1 X, \$_2 Y$. Moreover, if $M \subseteq \mathsf{q}\mathcal{M}_k(X, Y)$, then $M' \subseteq \mathcal{M}_{k+1}(\$_1 X, \$_2 Y)$ holds by Corollary 5.17. Hence, we can construct $\mathsf{GR}_{k+1}(\$_1 X, \$_2 Y)$ using Proposition 5.12. Then, we derive $\mathsf{qGR}_k(X, Y)$ using Corollary 5.17 again. By Lemma 3.11, we have $|\mathcal{B}_{k+1}(\$_1 X, \$_2 Y)| = \mathcal{O}(k^5)$, which gives the desired time bound. □

Observation 5.20. *Let $k' \leq k$. Given $\mathsf{qGR}_k(X, Y)$, the encoding $\mathsf{qGR}_{k'}(X, Y)$ can be computed in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.*

Proof. For $k' \leq k$, we have $\mathsf{q}\mathcal{M}_k(X, Y) \subseteq \mathsf{q}\mathcal{M}_{k'}(X, Y)$ by definition. The claim follows from Corollary 5.19. □

We now show that the quasi-greedy encodings are *concatenatable*.

Lemma 5.21. *Consider strings $X_p, Y_p, X_s, Y_s \in \Sigma^{\leq n}$ and $k \in \mathbb{Z}_+$. Assume $\max\{\big||X_p| - |Y_p|\big|, \big||X_s| - |Y_s|\big|\} = \mathcal{O}(k)$. Given $\mathsf{qGR}_{k+||X_s|-|Y_s||}(X_p, Y_p)$ and $\mathsf{qGR}_{k+||X_p|-|Y_p||}(X_s, Y_s)$, one can compute $\mathsf{qGR}_k(X_p X_s, Y_p Y_s)$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.*

Proof. For brevity, let $M_p = \mathsf{q}\mathcal{M}_{k+||X_s|-|Y_s||}(X_p, Y_p)$ and $M_s = \mathsf{q}\mathcal{M}_{k+||X_p|-|Y_p||}(X_s, Y_s)$. Let also $X = X_p X_s$, $Y = Y_p Y_s$.

By Lemma 5.15, if $\mathsf{ed}(X, Y) \leq k$, then $M := M_p \cup \{(x + |X_p|, y + |Y_p|) : (x, y) \in M_s\} \subseteq \mathsf{q}\mathcal{M}_k(X, Y)$. Hence, we shall construct $\mathsf{E}^M(X, Y)$ and then apply Corollary 5.19.

For this, we extract $\mathsf{D}(X_p^{M_p}), \mathsf{D}(X_s^{M_s}), \mathsf{D}(Y_p^{M_p}), \mathsf{D}(Y_s^{M_s})$ using Proposition 4.2(g), and then concatenate them to $\mathsf{D}(X^M Y^M) = \mathsf{D}(X_p^{M_p} X_s^{M_s} Y_p^{M_p} Y_s^{M_s})$ using Proposition 4.2(i). Overall, this takes $\mathcal{O}(k^2 \log^4 n)$ time. Then, we use Proposition 5.2 to build $\mathsf{RS}_\#(X^M)$ and $\mathsf{RS}_\#(Y^M)$ in $\tilde{\mathcal{O}}(k)$ time.

Finally, we note that using Corollary 5.19 costs $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space (here we use the fact that $\big||X_p| - |Y_p|\big| = \mathcal{O}(k)$). □

Finally, as a corollary we derive an algorithm that can compute the quasi-greedy encoding of arbitrarily long strings.

Corollary 5.22. *Assuming constant-time random access to a string $X \in \Sigma^\ell$ and streaming access to a string $Y \in \Sigma^\ell$, where $\ell \le n$, there is an algorithm that constructs $\mathsf{qGR}_k(X,Y)$ with a delay of at most k^2 characters in $\tilde{\mathcal{O}}(k^3)$ amortised time per character and $\tilde{\mathcal{O}}(k^2)$ space. The delay means that at the moment when the y-th character of Y arrives, the algorithm knows $\mathsf{qGR}_k(X[..y'], Y[..y'])$, where $|y - y'| \le k^2$.*

Proof. If $\ell \le k^2$, construct $\mathsf{qGR}_k(X,Y)$ via Claim 5.18 in $\tilde{\mathcal{O}}(k^3)$ (total) time and $\tilde{\mathcal{O}}(k^2)$ space. Otherwise, partition the strings into non-overlapping blocks $X = X_1 \cdots X_p$ and $Y = Y_1 \cdots Y_p$ so that $|X_i| = |Y_i| = k^2$ for all $i \in [1..p)$ and $|X_p| = |Y_z| = \ell \bmod k^2$. Suppose that we have computed $\mathsf{qGR}_k(X_1 \cdots X_{i-1}, Y_1 \cdots Y_{i-1})$ for $i \in [1..p)$. Compute $\mathsf{qGR}_k(X_1 \cdots X_i, Y_1 \cdots Y_i)$ in the following manner: first, compute $\mathsf{qGR}_k(X_i, Y_i)$ in $\mathcal{O}(k^5)$ time and $\mathcal{O}(k^2)$ space via Claim 5.18, and then compute $\mathsf{qGR}_k(X_1 \cdots X_i, Y_1 \cdots Y_i)$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space via Lemma 5.21. □

5.4 Products of Greedy Alignments

Definition 3.12. *Consider strings $X, Y, Z \in \Sigma^*$, an alignment $\mathcal{A}^{X,Y}$ of X, Y, an alignment $\mathcal{A}^{Y,Z}$ of Y, Z, and an alignment $\mathcal{A}^{X,Z}$ of X, Z. We say that $\mathcal{A}^{X,Z}$ is a* product *of $\mathcal{A}^{X,Y}$ and $\mathcal{A}^{Y,Z}$ if, for every $(x,z) \in \mathcal{A}^{X,Z}$, there is $y \in [1..|Y|+1]$ such that $(x,y) \in \mathcal{A}^{X,Y}$ and $(y,z) \in \mathcal{A}^{Y,Z}$.*

Lemma 3.13. *Consider strings $X, Y, Z \in \Sigma^*$ and $k \in \mathbb{Z}_{\ge 0}$. Every alignment $\mathcal{A}^{X,Z} \in \mathsf{GA}_k(X,Z)$ is a product of alignments $\mathcal{A}^{X,Y} \in \mathsf{GA}_d(X,Y)$ and $\mathcal{A}^{Y,Z} \in \mathsf{GA}_d(Y,Z)$, where $d = 2k + \mathsf{ed}(X,Y)$.*

Proof. We proceed by induction on $|X| + |Y| + |Z|$. In the base case, when at least one of the strings X, Y, Z is empty, we set $\mathcal{A}^{X,Y}$ and $\mathcal{A}^{Y,Z}$ to be any greedy optimal alignments of X, Y and Y, Z, respectively, so that $\mathsf{cost}(\mathcal{A}^{X,Y}) = \mathsf{ed}(X,Y)$ and $\mathsf{cost}(\mathcal{A}^{Y,Z}) = \mathsf{ed}(Y,Z) \le \mathsf{ed}(X,Y) + \mathsf{ed}(X,Z) \le \mathsf{ed}(X,Y) + \mathsf{cost}(\mathcal{A}^{X,Z})$. Moreover, it easy to check that $\mathcal{A}^{X,Z}$ is a product of $\mathcal{A}^{X,Y}$ and $\mathcal{A}^{Y,Z}$ because two out of these three alignments simply delete all characters of the non-empty string.

In the inductive step, we assume that all strings X, Y, Z are non-empty, and we consider several cases.

1. $\mathbf{X[1] = Z[1] = Y[1]}$.
   We recurse on $X' = X[2..]$, $Y' = Y[2..]$, $Z' = Z[2..]$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[2..],[2..]}$, which is greedy due to $X[1] \simeq_{\mathcal{A}^{X,Z}} Z[1]$. This yields greedy alignments $\mathcal{A}^{X',Y'}, \mathcal{A}^{Y',Z'}$ of cost at most $d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) + \mathsf{ed}(X,Y) = d$. We extend them so that $X[1] \simeq_{\mathcal{A}^{X,Y}} Y[1]$ and $Y[1] \simeq_{\mathcal{A}^{Y,Z}} Z[1]$, obtaining alignments of cost up to d.
2. $\mathbf{X[1] = Z[1] \neq Y[1]}$.
   In this case, we have $\mathsf{ed}(X,Y) > \min(\mathsf{ed}(X[2..],Y[2..]), \mathsf{ed}(X[2..],Y), \mathsf{ed}(X,Y[2..]))$.
   (a) If $\mathsf{ed}(X,Y) > \mathsf{ed}(X[2..],Y[2..])$, we recurse on $X' = X[2..]$, $Y' = Y[2..]$, $Z' = Z[2..]$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[2..],[2..]}$, which is greedy due to $X[1] \simeq_{\mathcal{A}^{X,Z}} Z[1]$. This yields greedy alignments $\mathcal{A}^{X',Y'}, \mathcal{A}^{Y',Z'}$ of cost at most $d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) + \mathsf{ed}(X,Y) - 1 = d - 1$. We extend them so that $X[1] \sim_{\mathcal{A}^{X,Y}} Y[1]$ and $Y[1] \sim_{\mathcal{A}^{Y,Z}} Z[1]$, obtaining alignments of cost up to d.
   (b) If $\mathsf{ed}(X,Y) > \mathsf{ed}(X[2..],Y)$, we recurse on $X' = X[2..]$, $Y' = Y$, $Z' = Z[2..]$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[2..],[2..]}$, which is greedy due to $X[1] \simeq_{\mathcal{A}^{X,Z}} Z[1]$. This yields greedy alignments $\mathcal{A}^{X',Y'}, \mathcal{A}^{Y',Z'}$ of cost at most $d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) + \mathsf{ed}(X,Y) - 1 = d - 1$. We extend them so that $\mathcal{A}^{X,Y}$ deletes $X[1]$ and $\mathcal{A}^{Y,Z}$ deletes $Z[1]$, obtaining alignments of cost up to d.

(c) If $\mathsf{ed}(X,Y) > \mathsf{ed}(X,Y[2\,..])$, we recurse on $X' = X$, $Y' = Y[2\,..]$, $Z' = Z$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}$. This yields greedy alignments $\mathcal{A}^{X',Y'}, A^{Y',Z'}$ of cost at most $d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) + \mathsf{ed}(X,Y) - 1 = d - 1$. We extend them so that $\mathcal{A}^{X,Y}$ and $\mathcal{A}^{Y,Z}$ both delete $Y[1]$, obtaining alignments of cost up to d.

3. $\mathbf{X}[\mathbf{1}] \neq \mathbf{Z}[\mathbf{1}] = \mathbf{Y}[\mathbf{1}]$.
   (a) If $\mathcal{A}^{X,Z}$ deletes $X[1]$, we recurse on $X' = X[2\,..]$, $Y' = Y$, $Z' = Z$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[2\,..],[1\,..]}$. This yields greedy alignments $\mathcal{A}^{X',Y'}, A^{Y',Z'}$ of cost $\leq d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) - 2 + \mathsf{ed}(X',Y') \leq d - 1$. We derive $\mathcal{A}^{Y,Z} = \mathcal{A}^{Y',Z'}$ and extend $\mathcal{A}^{X',Y'}$ so that $\mathcal{A}^{X,Y}$ deletes $X[1]$, obtaining an alignment of cost up to d.
   (b) If $\mathcal{A}^{X,Z}$ deletes $Z[1]$, we recurse on $X' = X$, $Y' = Y[2\,..]$, $Z' = Z[2\,..]$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[1\,..],[2\,..]}$. This yields greedy alignments $\mathcal{A}^{X',Y'}, A^{Y',Z'}$ of cost $\leq d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) - 2 + \mathsf{ed}(X',Y') \leq d - 1$. We extend them so that $\mathcal{A}^{X,Y}$ deletes $Y[1]$ and $Y[1] \simeq_{\mathcal{A}^{Y,Z}} Z[1]$, obtaining alignments of cost up to d.
   (c) If $X[1] \sim_{\mathcal{A}^{X,Z}} Z[1]$, we recurse on $X' = X[2\,..]$, $Y' = Y[2\,..]$, $Z' = Z[2\,..]$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[2\,..],[2\,..]}$. This yields greedy alignments $\mathcal{A}^{X',Y'}, A^{Y',Z'}$ of cost $\leq d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) - 2 + \mathsf{ed}(X',Y') \leq d - 1$. We extend them so that $X[1] \sim_{\mathcal{A}^{X,Y}} Y[1]$ and $Y[1] \simeq_{\mathcal{A}^{Y,Z}} Z[1]$, obtaining alignments of cost up to d.
4. $\mathbf{X}[\mathbf{1}] \neq \mathbf{Z}[\mathbf{1}] \neq \mathbf{Y}[\mathbf{1}]$. (Note that this case allows both $X[1] = Y[1]$ and $X[1] \neq Y[1]$.)
   (a) If $\mathcal{A}^{X,Z}$ deletes $X[1]$, we recurse on $X' = X[2\,..]$, $Y' = Y[2\,..]$, $Z' = Z$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[2\,..],[1\,..]}$. This yields greedy alignments $\mathcal{A}^{X',Y'}, A^{Y',Z'}$ of cost at most $d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) - 2 + \mathsf{ed}(X',Y') \leq d - 2$. We extend them so that $X[1] \sim_{\mathcal{A}^{X,Y}} Y[1]$ and $\mathcal{A}^{Y,Z}$ deletes $Y[1]$, obtaining alignments of cost up to $d - 1$.
   (b) If $\mathcal{A}^{X,Z}$ deletes $Z[1]$, we recurse on $X' = X$, $Y' = Y$, $Z' = Z[2\,..]$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[1\,..],[2\,..]}$. This yields greedy alignments $\mathcal{A}^{X',Y'}, A^{Y',Z'}$ of cost $\leq d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) - 2 + \mathsf{ed}(X,Y) = d - 2$. We derive $\mathcal{A}^{X,Y} = \mathcal{A}^{X',Y'}$ and extend $\mathcal{A}^{Y',Z'}$ so that $\mathcal{A}^{Y,Z}$ deletes $Z[1]$, obtaining an alignment of cost up to $d - 1$.
   (c) If $X[1] \sim_{\mathcal{A}^{X,Z}} Z[1]$, we recurse on $X' = X[2\,..]$, $Y' = Y[2\,..]$, $Z' = Z[2\,..]$, and $\mathcal{A}^{X',Z'} = \mathcal{A}^{X,Z}_{[2\,..],[2\,..]}$. This yields greedy alignments $\mathcal{A}^{X',Y'}, A^{Y',Z'}$ of cost $\leq d' = 2\mathsf{cost}(\mathcal{A}^{X,Z}) - 2 + \mathsf{ed}(X',Y') \leq d - 2$. We extend them so that $X[1] \sim_{\mathcal{A}^{X,Y}} Y[1]$ and $Y[1] \sim_{\mathcal{A}^{Y,Z}} Z[1]$, obtaining alignments of cost up to $d - 1$.

In all the cases above, $\mathcal{A}^{X,Y}$ is greedy because $\mathcal{A}^{X',Y'}$ is greedy and $\mathcal{A}^{X,Y}$ matches $X[1]$ with $Y[1]$ whenever $X[1] = Y[1]$. Similarly, $\mathcal{A}^{Y,Z}$ is greedy because $\mathcal{A}^{Y',Z'}$ is greedy and $\mathcal{A}^{Y,Z}$ matches $Y[1]$ with $Z[1]$ whenever $Y[1] = Z[1]$. Moreover, $\mathcal{A}^{X,Z}$ is a product of $\mathcal{A}^{X,Y}$ and $\mathcal{A}^{Y,Z}$ because each alignment starts with $(1,1)$ and since $\mathcal{A}^{X',Z'}$ is a product of $\mathcal{A}^{X',Y'}$ and $\mathcal{A}^{Y',Z'}$. □

Assume that we are given three strings X, Y, Z. Let $d = \mathsf{ed}(X,Y) + 2k$, and define $\mathcal{G}_X = \mathsf{GR}_d(X,Y)$ and $\mathcal{G}_Z = \mathsf{GR}_d(Y,Z)$. We show that given $\mathcal{G}_X$ and $\mathcal{G}_Z$, we can compute an optimal alignment between X and Z efficiently if its cost is at most k. Let $M = \{(x,z) : \exists y \text{ such that } (x,y) \in \mathcal{M}_d(X,Y) \text{ and } (y,z) \in \mathcal{M}_d(Y,Z)\}$.

Lemma 5.23. *If* $\mathsf{ed}(X,Z) \leq k$, *then* $M \subseteq \mathcal{M}_k(X,Z)$.

Proof. Suppose that $(x,y) \in \mathcal{M}_d(X,Y)$ and $(y,z) \in \mathcal{M}_d(Y,Z)$. For a proof by contradiction, suppose that $X[x] \not\sim_{\mathcal{A}} Z[z]$ for some $\mathcal{A} \in \mathsf{GA}_k(X,Z)$. Then, there exists $(x',z') \in \mathcal{A}$ such that either $x' \leq x$ and $z' > z$, or $x' > x$ and $z' \leq z$. By symmetry, without loss of generality, we consider the first alternative. By Lemma 3.13, $\mathcal{A}$ is a product $\mathcal{A}^{X,Y} \in \mathsf{GA}_d(X,Y)$ and $\mathcal{A}^{Y,Z} \in \mathsf{GA}_d(Y,Z)$. According to Definition 3.12, this means that there exists y' such that $(x',y') \in \mathcal{A}^{X,Y}$ and $(y',z') \in \mathcal{A}^{Y,Z}$. If $y' \leq y$, then $(y,z),(y',z') \in \mathcal{A}^{Y,Z}$ implies that $\mathcal{A}^{Y,Z}$

deletes $Z[z]$, contradicting $(y,z) \in \mathcal{M}(\mathcal{A}^{Y,Z})$. Similarly, if $y' > y$, then $(x,y),(x',y') \in \mathcal{A}^{X,Y}$ implies that $\mathcal{A}^{X,Y}$ deletes $Y[y]$, contradicting $(x,y) \in \mathcal{M}(\mathcal{A}^{X,Y})$. This completes the proof that $X[x] \sim_{\mathcal{A}} Z[z]$ for every $\mathcal{A} \in \mathsf{GA}_k(X,Z)$. Due to $X[x] = Y[y] = Z[z]$, we also have $X[x] \simeq_{\mathcal{A}} Z[z]$ for every $\mathcal{A} \in \mathsf{GA}_k(X,Z)$, i.e., $(x,y) \in \mathcal{M}_k(X,Z)$ holds as claimed. □

Lemma 5.24. *If* $\mathsf{ed}(X,Y) = \mathcal{O}(k)$ *and* $\mathcal{G}_X, \mathcal{G}_Z \neq \bot$, $\mathsf{E}^M(X,Z)$ *can be computed in* $\tilde{\mathcal{O}}(k^3)$ *time and* $\tilde{\mathcal{O}}(k^2)$ *space.*

Proof. Let us first explain how X^M can be constructed. Consider a position $x \in [1\,..\,|X|]$ such that $(x,z) \notin M$ for every $z \in [1\,..\,|Z|]$. If $X^{\mathcal{M}_d(X,Y)}[x] \neq \#$, then $X^M[x] = X[x] = X^{\mathcal{M}_d(X,Y)}[x]$. Otherwise, $(x,y) \in \mathcal{M}_d(X,Y)$ for some $y \in [1\,..\,|Y|]$. By Lemma 5.23, we have $(y,z) \notin \mathcal{M}_d(Y,Z)$ for every $z \in [1\,..\,|Z|]$. Hence, $Y^{\mathcal{M}_d(Y,Z)}[y] \neq \#$, and we have $X[x] = Y[y] = Y^{\mathcal{M}_d(Y,Z)}[y]$. In words, non-dummy characters of X^M can be retrieved from non-dummy characters of $X^{\mathcal{M}_d(X,Y)}$ and non-dummy characters of $Y^{\mathcal{M}_d(Y,Z)}$.

Thus, for each dummy segment $[\ell\,..\,r)$ in $X^{\mathcal{M}_d(X,Y)}$ and its counterpart $[\ell'\,..\,r')$ in $Y^{\mathcal{M}_d(X,Y)}$, we need to set $X^M[\ell\,..\,r) := Y^{\mathcal{M}_d(Y,Z)}[\ell'\,..\,r')$. Such a copy-paste operation can be implemented using Proposition 4.2(g)(i), in $\mathcal{O}(k^2 \log^4 n)$ time per dummy segment. We can also keep track of the dummy segments in X^M within the same procedure. The algorithm for Z^M is symmetric, and thus we can construct $\mathsf{D}(X^M Z^M)$ along with the dummy segments in $\tilde{\mathcal{O}}(k^3)$ time and $\tilde{\mathcal{O}}(k^2)$ space. Finally, we build $\mathsf{RS}_{\#}(X^M)$ and $\mathsf{RS}_{\#}(Z^M)$ in $\mathcal{O}(k)$ time. □

Corollary 5.25. *Given* $\mathcal{G}_X$ *and* $\mathcal{G}_Z$. *If* $\mathsf{ed}(X,Y) = \mathcal{O}(k)$, *then we can compute* $\min(k+1, \mathsf{ed}(X,Z))$ *in* $\tilde{\mathcal{O}}(k^3)$ *time and* $\tilde{\mathcal{O}}(k^2)$ *space.*

Proof. If $\mathcal{G}_X$ or $\mathcal{G}_Z$ equals to $\bot$, then $\mathsf{ed}(X,Z) > k$ by Lemma 3.13. Otherwise, construct $\mathsf{E}^M(X,Z)$ using Lemma 5.24. Finally, we pass k and $\mathsf{E}^M(X,Z)$ to the algorithm of Corollary 5.5, and we return the resulting value d.

Note that M is a non-crossing matching of X, Z, so $d = k+1$ is correctly returned if $\mathsf{ed}(X,Z) > k$. If $\mathsf{ed}(X,Z) \leq k$, then Lemma 5.23 implies $M \subseteq \mathcal{M}_k(X,Z)$, and thus $d = \mathsf{ed}(X,Z)$ holds as claimed.

The overall runtime and space complexity are dominated by the procedure of Lemma 5.24. □

Remark 5.26. Using Proposition 5.12 instead of Corollary 5.5, we could construct $\mathsf{GR}_k(X,Z)$.

Corollary 5.27. *Consider three strings* X, Y, Z. *Let* $d = \mathsf{ed}(X,Y) + 2k$, *and assume that we are given* $\mathsf{qGR}_d(X,Y)$ *and* $\mathsf{qGR}_d(Y,Z)$. *If* $\mathsf{ed}(X,Y) = \mathcal{O}(k)$, *then we can compute* $\min(k+1, \mathsf{ed}(X,Z))$ *in* $\tilde{\mathcal{O}}(k^3)$ *time and* $\tilde{\mathcal{O}}(k^2)$ *space.*

Proof. We compute $\mathcal{G}_X = \mathsf{GR}_{d+1}(\$_1 X, \$_2 Y)$ and $\mathsf{qGR}_{d+1}(\$_2 Y, \$_3 Z)$ in $\tilde{\mathcal{O}}(k^2)$ time and space via Corollary 5.17, and apply Corollary 5.25 to compute $\min(k+2, \mathsf{ed}(\$_1 X, \$_2 Z)) - 1 = \min(k+1, \mathsf{ed}(X,Z))$ in $\tilde{\mathcal{O}}(k^3)$ time and $\tilde{\mathcal{O}}(k^2)$ space. □

6 Edit Distance Sketches

6.1 CGK embedding

In this section, we prove Proposition 3.16 based on the CGK embedding introduced in [10]. Recall that the Hamming distance between the embeddings of two strings $X, Y \in \Sigma^{\leq n}$ is bounded in terms of the edit distance $\mathsf{ed}(X,Y)$, which allows using Hamming distance sketches to approximate edit distance.

Definition 6.1 (CGK embedding [10]). *Consider an alphabet Σ, a sentinel character $\perp \notin \Sigma$, and a 2-independent family $\mathcal{H}$ of hash functions $h : \Sigma \to \{0,1\}$. For an integer $n \in \mathbb{Z}_+$, a (uniformly random) sequence $R \in \mathcal{H}^{3n}$, and a string $S \in \Sigma^{\leq n}$, the* CGK walk $W_{\mathsf{CGK}}(S) = (s_t)_{t=1}^{3n+1}$ *and the* CGK embedding $\mathsf{CGK}(S) \in (\Sigma \cup \{\perp\})^{3n}$ *are defined by the following algorithm:*

Input: An integer $n \in \mathbb{Z}_+$, a string $S \in \Sigma^{\leq n}$.
Randomness: A sequence $R \in \mathcal{H}^{3n}$ of 2-independent hash functions $\Sigma \to \{0,1\}$.
Output: The CGK walk $W_{\mathsf{CGK}}(S) = (s_t)_{t=1}^{3n+1}$ and the CGK embedding $\mathsf{CGK}(S)$.

$s_1 := 1$;
for $t := 1$ **to** $3n$ **do**
  if $s_t \leq |S|$ **then**
    $\mathsf{CGK}(S)[t] := S[s_t]$;
    $s_{t+1} := s_t + R_t(S[s_t])$;
  else
    $\mathsf{CGK}(S)[t] := \perp$;
    $s_{t+1} := s_t$;

Algorithm 2: The CGK algorithm

Recall from Definition 3.14 that an m-step walk over S is complete if $s_{m+1} = |S| + 1$.

Fact 6.2 ([10, Theorem 4.1]). *Each $S \in \Sigma^{\leq n}$ satisfies $\Pr_R[W_{\mathsf{CGK}}(S)$ is complete$] \geq 1 - e^{-\Omega(n)}$.*

The following result summarizes the central property of the CGK embedding.

Fact 6.3 ([10, Theorem 4.3]). *For every $X, Y \in \Sigma^{\leq n}$ and every constant $c > 0$, the embeddings $\mathsf{CGK}(X)$, $\mathsf{CGK}(Y)$ satisfy $\Pr_{R \in \mathcal{H}^{3n}}[\mathsf{hd}(\mathsf{CGK}(X), \mathsf{CGK}(Y)) > c \cdot \mathsf{ed}(X,Y)^2] < \frac{12}{\sqrt{c}}$.*

If $W_{\mathsf{CGK}}(X)$ and $W_{\mathsf{CGK}}(Y)$ are complete, Definition 3.15 yields an edit-distance alignment of X, Y, which we call the *CGK alignment* of X, Y. that they induce an edit-distance alignment

Fact 6.4 (see also [10, Theorem 4.2]). *If $W_{\mathsf{CGK}}(X)$ and $W_{\mathsf{CGK}}(Y)$ are complete for some $X, Y \in \Sigma^{\leq n}$ and $R \in \mathcal{H}^{3n}$, then the CGK alignment of X, Y belongs to $\mathsf{GA}_{\mathsf{hd}(\mathsf{CGK}(X),\mathsf{CGK}(Y))}(X, Y)$.*

Proof. Let $\mathcal{A}$ be the CGK alignment of X, Y, i.e., the zip alignment of $W_{\mathsf{CGK}}(X) = (x_t)_{t=1}^{3n+1}$ and $W_{\mathsf{CGK}}(Y) = (y_t)_{t=1}^{3n+1}$. Consider $(x_t, y_t) \in \mathcal{B}(\mathcal{A})$, with $t \in [1\,..\,3n+1]$ chosen so that $t = 3n+1$ or $(x_{t+1}, y_{t+1}) \neq (x_t, y_t)$. If $t = 3n+1$, then $(x_t, y_t) = (|X|+1, |Y|+1)$ by the assumption that $W_{\mathsf{CGK}}(X)$ and $W_{\mathsf{CGK}}(Y)$ are complete. If $(x_{t+1}, y_{t+1}) = (x_t+1, y_t+1)$, then we have $\mathsf{CGK}(X)[t] = X[x_t] \neq Y[y_t] = \mathsf{CGK}(Y)[t]$. The remaining possibility $x_{t+1} - x_t \neq y_{t+1} - y_t$ holds only in the following three cases: if $x_t = |X|+1$ and $y_t \leq |Y|$ (when $\mathsf{CGK}(X)[t] = \perp \neq \mathsf{CGK}(Y)[t] = Y[y_t]$), if $x_t \leq |X|$ and $y_t = |Y|+1$ (when $X[x_t] = \mathsf{CGK}(X)[t] \neq \perp = \mathsf{CGK}(Y)[t]$), or if $x_t \leq |X|$, $y_t \leq |Y|$, and $R_t(X[x_t]) \neq R_t(Y[y_t])$ (when $\mathsf{CGK}(X)[t] = X[x_t] \neq Y[y_t] = \mathsf{CGK}(Y)[t]$). Overall, we have $X[x_t] \neq Y[y_t]$ whenever $(x_t, y_t) \in [1\,..\,|X|] \times [1\,..\,|Y|]$, and $\mathsf{CGK}(X)[t] \neq \mathsf{CGK}(Y)[t]$ whenever $t \in [1\,..\,3n]$. Consequently, $\mathcal{A} \in \mathsf{GA}_{\mathsf{hd}(\mathsf{CGK}(X),\mathsf{CGK}(Y))}(X, Y)$. □

The final property of the CGK alignment required in this work is that its width is within $\mathcal{O}(\mathsf{ed}(X,Y))$ with good probability. For this, we first prove a fact about random walks:

Fact 6.5. *Let $(w_i)_{i \geq 0}$ be an unbiased lazy random walk (that is, $w_0 = 0$, and, for every $i \geq 1$, we have $\Pr[w_{i+1} = w_i \mid w_0, \ldots, w_i] = \frac{1}{2}$, $\Pr[w_{i+1} = w_i + 1 \mid w_0, \ldots, w_i] = \Pr[w_{i+1} = w_i - 1 \mid w_0, \ldots, w_i] = \frac{1}{4}$). Then, for every $m, \ell \in \mathbb{Z}_+$, we have $\Pr[\max_{i=0}^{m} |w_i| \geq \ell] \leq \frac{m}{\ell^2}$.*

Proof. By the reflection principle [44, Excercise 2.10], we have $\Pr[\max_{i=0}^{m} |w_i| \geq \ell] \leq 2\Pr[|w_m| \geq \ell] = 2\Pr[w_m^2 \geq \ell^2]$. By Markov's inequality,

$$2\Pr[w_m^2 \geq \ell^2] \leq \frac{2\mathbb{E}[w_m^2]}{\ell^2} = \frac{2\mathbb{E}[(w_1 - w_0)^2 + \cdots + (w_m - w_{m-1})^2]}{\ell^2} = \frac{m}{\ell^2}. \qquad \square$$

Corollary 6.6. *For every constant $\delta \in (0,1)$, there exists a constant c such that for sufficiently large n, all strings $X, Y \in \Sigma^{\leq n}$, and their CGK alignment $\mathcal{A}$, the probability over $R \in \mathcal{H}^{3n}$ that $W_{\mathsf{CGK}}(X), W_{\mathsf{CGK}}(Y)$ are complete, $\mathcal{A} \in \mathsf{GA}_{c\cdot\mathsf{ed}(X,Y)^2}(X,Y)$, and $\mathsf{width}(\mathcal{A}) \leq c \cdot \mathsf{ed}(X,Y)$ is at least $1-\delta$.*

Proof. By Fact 6.2, $\Pr[W_{\mathsf{CGK}}(S) \text{ is incomplete}] \leq \frac{\delta}{4}$ holds for every $S \in \Sigma^{\leq n}$ and sufficiently large $n = \Omega(\log \frac{1}{\delta})$. By Fact 6.3, there is a constant c such that $\Pr[\mathsf{hd}(\mathsf{CGK}(X), \mathsf{CGK}(Y)) > c\cdot\mathsf{ed}(X,Y)^2] \leq \frac{\delta}{4}$. We may take arbitrarily large c, so we shall also assume that $c \geq \frac{4}{\delta}$. Moreover, by Fact 6.4, if $W_{\mathsf{CGK}}(X)$ and $W_{\mathsf{CGK}}(Y)$ are complete and $\mathsf{hd}(\mathsf{CGK}(X), \mathsf{CGK}(Y)) \leq c \cdot \mathsf{ed}(X,Y)^2$, then $\mathcal{A} \in \mathsf{GA}_{c\cdot\mathsf{ed}(X,Y)^2}(X,Y)$.

To bound $\mathsf{width}(\mathcal{A})$, let us analyze the CGK walks $W_{\mathsf{CGK}}(X) = (x_t)_{t=1}^{3n+1}$ and $W_{\mathsf{CGK}}(Y) = (y_t)_{t=1}^{3n+1}$. Let T be the first step such that $x_T = |X|+1$, $y_T = |Y|+1$, or $T = 3n+1$. Note that it suffices to bound $\Pr[\max_{t=1}^{T} |x_t - y_t| \geq ck]$, where $k = \mathsf{ed}(X,Y)$.

Let $A = \{1\} \cup \{t+1 : t \in [1\,..\,T) \text{ and } X[x_t] \neq Y[y_t]\}$. Observe that $x_t - y_t = x_{t-1} - y_{t-1}$ holds for $t \in [1\,..\,T] \setminus A$, so we may focus on bounding $\Pr[\max_{t\in A} |x_t - y_t| \geq ck]$. Let $A = \{t_0, \ldots, t_a\}$ with $t_0 < \cdots < t_a$, and let $d_i = x_{t_i} - y_{t_i}$ for $i \in [0\,..\,a]$. Observe that $(d_i)_{i=0}^{a}$ is an a-step unbiased lazy random walk and that $a \leq \mathsf{hd}(\mathsf{CGK}(X), \mathsf{CGK}(Y))$. Moreover, if we extend $(d_i)_{i=0}^{a}$ to an infinite unbiased lazy random walk $(d_i)_{i=0}^{\infty}$, then Fact 6.5 yields $\Pr[\max_{i=0}^{\lfloor ck^2 \rfloor} |d_i| \geq ck] \leq \frac{ck^2}{c^2k^2} = \frac{1}{c} \leq \frac{\delta}{4}$. Therefore,

$$\Pr\left[\max_{i=0}^{a} |d_i| \geq ck\right] \leq \Pr\left[\mathsf{hd}(\mathsf{CGK}(X), \mathsf{CGK}(Y)) > ck^2\right] + \Pr\left[\max_{i=0}^{\lfloor ck^2 \rfloor} |d_i| \geq ck\right].$$

Consequently, if $\mathsf{CGK}(X)$ and $\mathsf{CGK}(Y)$ are complete, $\mathsf{hd}(\mathsf{CGK}(X), \mathsf{CGK}(Y)) \leq ck^2$, and the random walk satisfies $\Pr\left[\max_{i=0}^{\lfloor ck^2 \rfloor} |d_i| < ck\right]$, then the alignment $\mathcal{A}$ satisfies the lemma. The total probability of the complementary events is at most δ, so this completes the proof. $\square$

To complete the proof of Proposition 3.16 (repeated below), it remains to reduce the number of random bits using Nisan's pseudorandom generator [50].

Proposition 3.16. *For every constant $\delta \in (0,1)$, there exists a constant c and an algorithm W that, given an integer n, a seed r of $\mathcal{O}(\log^2 n)$ random bits, and a string $S \in \Sigma^{\leq n}$, outputs a $3n$-step complete walk $\mathsf{W}(n,r,S)$ over S satisfying the following property for all $X, Y \in \Sigma^{\leq n}$ and the zip alignment $\mathcal{A}_{\mathsf{W}}$ of $\mathsf{W}(n,r,X)$ and $\mathsf{W}(n,r,Y)$:*

$$\Pr_r\left[\mathcal{A}_{\mathsf{W}} \in \mathsf{GA}_{c\cdot\mathsf{ed}(X,Y)^2}(X,Y) \text{ and } \mathsf{width}(\mathcal{A}_{\mathsf{W}}) \leq c \cdot \mathsf{ed}(X,Y)\right] \geq 1-\delta.$$

Moreover, W is an $\mathcal{O}(\log^2 n)$-bit streaming algorithm that costs $\mathcal{O}(n \log n)$ time and reports any element $s_t \in [1\,..\,|S|]$ of $\mathsf{W}(n,r,S)$ while processing the corresponding character $S[s_t]$.

Proof. Let $c_{6.6}$ and $n_{6.6}$ be the constant and threshold (respectively) of Corollary 6.6 for $\delta_{6.6} = \frac{\delta}{2}$.

If $n < \max(\frac{10}{\delta}, n_{6.6})$, we set $\mathsf{W}(n,r,S) = (\min(t, |S|+1))_{t=1}^{3n+1}$ for all $S \in \Sigma^{\leq n}$ so that $\mathsf{W}(n,r,S)$ is trivially a $3n$-step complete walk over S. Now, consider strings $X, Y \in \Sigma^{\leq n}$ and the zip alignment $\mathcal{A}_{\mathsf{W}}$ of $\mathsf{W}(n,r,X)$ and $\mathsf{W}(n,r,Y)$. Observe that $\mathcal{A}_{\mathsf{W}} \in \mathsf{GA}_0(X,Y)$ if $X = Y$ and $\mathcal{A}_{\mathsf{W}} \in \mathsf{GA}_n(X,Y)$ otherwise. Moreover, $\mathsf{width}(\mathcal{A}_{\mathsf{W}}) = \big||X| - |Y|\big| \leq \mathsf{ed}(X,Y)$. Consequently,

the claimed conditions are (deterministically) satisfied for $c \geq n_{6.6}$. The construction algorithm uses $\mathcal{O}(\log n)$ bits and $\mathcal{O}(n)$ time.

If $n \geq \max(\frac{10}{\delta}, n_{6.6})$, we first develop an algorithm W' that uses $\Theta(n \log \sigma)$ random bits, interpreted as a sequence $R \in \mathcal{H}^{3n}$. Specifically, each hash function R_t is specified by an $\lceil \log \sigma \rceil$-bit integer h_t so that $R_t(a)$ counts (modulo 2) the set bits in $a \mathbin{\mathsf{xor}} h_t$.

We set $\mathsf{W}'(n, R, S) = (\max(t-3n+|S|, s_t))_{t=1}^{3n+1}$ based on the CGK walk $W_{\mathsf{CGK}}(S) = (s_t)_{t=1}^{3n+1}$ for all $S \in \Sigma^{\leq n}$. Observe that this modification guarantees that $\mathsf{W}'(n, R, S)$ is a complete walk over S and that $\mathsf{W}'(n, R, S) = W_{\mathsf{CGK}}(S)$ if $W_{\mathsf{CGK}}(S)$ is already complete. Consequently, by Corollary 6.6, the claimed conditions are satisfied with probability at least $1 - \delta_{6.6} = 1 - \frac{\delta}{2}$ for $c \geq c_{6.6}$. The construction algorithm uses $\mathcal{O}(\log n)$ bits, costs $\mathcal{O}(n)$ time, reads the random bits $(h_t)_{t=1}^{3n+1}$ from left to right, and outputs the elements of $\mathsf{W}'(n, R, S)$ when required.

In order to use Nisan's pseudorandom generator [50], we need to argue that we only care about properties testable using $\mathcal{O}(\log n)$-bit algorithms with one-way access to the randomness R. As in [10], our testers are given two (read-only) strings $X, Y \in \Sigma^{\leq n}$ and a stream of random bits representing R (the sequence $(h_t)_{t=1}^{3n}$). Observe that the following properties of the zip alignment $\mathcal{A}'$ of $\mathsf{W}'(n, R, X) = (x_t)_{t=1}^{3n+1}$ and $\mathsf{W}'(n, R, Y) = (y_t)_{t=1}^{3n+1}$ are testable in $\mathcal{O}(\log n)$ bits and $\mathcal{O}(n)$ time:

- whether $\mathcal{A}'$ is greedy;
- whether $\mathsf{cost}(\mathcal{A}') \leq k$ for a given integer $k \in [0\,..\,n]$;
- whether $\mathsf{width}(\mathcal{A}') \leq w$ for a given integer $w \in [0\,..\,n]$.

All these testers simply construct triples (t, x_t, y_t) for subsequent $t \in [1\,..\,3n+1]$.

In this setting, Nisan's pseudorandom generator [50], given a sequence r of $\mathcal{O}(\log^2 n)$ random bits, constructs a sequence $\mathsf{PRG}(r)$ of pseudorandom bits in $\mathcal{O}(1)$ amortized time per bit, using $\mathcal{O}(\log^2 n)$ bits of working space. Moreover, for every tester, the probabilities of accepting a given input $\mathcal{I}$ with randomness R and $\mathsf{PRG}(r)$ differ by at most $\frac{1}{n^2}$ [10, Theorem 5]. Consequently, setting $\mathsf{W}(n, r, S) = \mathsf{W}'(n, \mathsf{PRG}(r), S)$, we can guarantee that the claimed condition is satisfied with probability at least $1 - \frac{1}{2}\delta - \frac{(2n+3)}{n^2} \geq 1 - \delta$. The streaming construction algorithm takes $\mathcal{O}(\log^2 n)$ bits and $\mathcal{O}(n \log n)$ time, dominated by the generation of $\mathsf{PRG}(r)$. □

6.2 Context encoding

For a string $S \in \Sigma^*$, define $\overline{\mathsf{maxLZ}}(S) = \max_{[\ell\,..\,r) \subseteq [1\,..\,|S|]} |\mathsf{LZ}(\overline{S[\ell\,..\,r)})|$.

Observation 6.7. *Consider a string $S \in \Sigma^*$ and an integer $k \in \mathbb{Z}_+$. If $\overline{\mathsf{maxLZ}}(S[\ell\,..\,r)) \leq k$ holds for a non-empty fragment $S[\ell\,..\,r)$, then:*

(a) $\overline{\mathsf{maxLZ}}(S[\ell\,..\,r]) \leq k$ *if and only if* $\mathsf{LZ}(\overline{S[\ell\,..\,r]}) \leq k$;
(b) $\overline{\mathsf{maxLZ}}(S[\ell+1\,..\,r)) \leq k$;
(c) $\overline{\mathsf{maxLZ}}(S[\ell\,..\,r]) \leq k+1$.

Proof. The first two parts follow from the fact that the size of the LZ-factorisation of a prefix of a string is bounded by the size of the LZ-factorisation of the string itself. The third claim follows from the optimality of the LZ77 parsing among all LZ-like parsings (Observation 4.1). □

Definition 6.8 ((Double) Context)**.** *Consider a string $S \in \Sigma^*$ and an integer $k \in \mathbb{Z}_+$. For a position $p \in [1\,..\,|S|]$, we define the* context $C_k(S, p)$ *as the longest prefix $S[p\,..\,q)$ of $S[p\,..\,|S|]$ such that $\overline{\mathsf{maxLZ}}(S[p\,..\,q)) \leq k$ and the* double context $C_k^2(S, p)$ *as the longest prefix $S[p\,..\,q)$ of $S[p\,..\,|S|]$ such that $\overline{\mathsf{maxLZ}}(S[p\,..\,r)) \leq k$ and $\overline{\mathsf{maxLZ}}(S[r\,..\,q)) \leq k$ for some $r \in [p\,..\,q]$.*

For integers $t < v$, let $\mu(t, v)$ denote the integer $w \in [t\,..\,v)$ divisible by the largest power of 2. We say that a position $S[s]$ is covered by a fragment $S[\ell\,..\,r)$ if $\ell \leq s < r$.

Input: An integer $k \in \mathbb{Z}_{\geq 0}$, a string $S \in \Sigma^*$, and a complete walk $(s_t)_{t=1}^{m+1}$ over S.
Output: The string $\mathsf{CE}_k(W)[1\,..\,m]$.

$\mathsf{CE}_k(W) := \perp^m$, where $\perp = (\mathsf{LZ}(\varepsilon), 0)$;
$s_0 := 0$; $\mu_0 := 0$; $u := 0$;
for $t := 1$ **to** m **do**
  if $s_t \leq |S|$ **then**
    $\mu_t := \mu(t, \min\{v \in [1\,..\,m+1] : s_v = s_t + |C_k(S, s_t)|\})$;
    if $\mu_t > \mu_{t-1}$ **then**
      $\mathsf{CE}_k(W)[t] := \left(\mathsf{LZ}\left(\overline{C_k^2(S, s_t)}\right), s_t - s_u\right)$;
      $u := t$;
return $\mathsf{CE}_k(W)$;

Algorithm 3: Function CE

Lemma 6.9. *Consider $\mathsf{CE}_k(W)$ constructed for an integer $k \in \mathbb{Z}_{\geq 0}$ and an m-complete walk $W = (s_t)_{t=1}^{m+1}$ over $S \in \Sigma^*$. Each position $s \in [1\,..\,|S|]$ satisfies*

$$1 \leq |\{t \in [1\,..\,m] : \mathsf{CE}_k(W)[t] \neq \perp \text{ and } C_k(S, s_t) \text{ covers } S[s]\}| \leq$$
$$\leq |\{t \in [1\,..\,m] : \mathsf{CE}_k(W)[t] \neq \perp \text{ and } C_k^2(S, s_t) \text{ covers } S[s]\}| \leq \mathcal{O}(\log m)$$

Proof. Let us first bound the covering number from below. Consider an index $t \in [1\,..\,m]$ such that $s = s_t$, and let $t' \in [1\,..\,m]$ be the smallest index such that $\mu_{t'} = \mu_t$. Note that $s_{\mu_{t'}} < s_{t'} + |C_k(S, s_{t'})|$ and $s_{\mu_t} \geq s_t$ by definition of μ and monotonicity of the walk W. Due to $\mu_{t'} = \mu_t$, this implies $s_{t'} \leq s_t < s_{t'} + |C_k(S, s_{t'})|$, i.e., that $S[s_t]$ is covered by $C_k(S, s_{t'})$. Due to $\mu_{t'-1} \neq \mu_t$, this guarantees $\mathsf{CE}_k(S)[t'] \neq \perp$.

As for the upper bound, note that the indexes $t \in [1\,..\,m]$ with $\mathsf{CE}_k(S)[t] \neq \perp$ have distinct values μ_t. Let us further classify them into $\mathcal{O}(\log n)$ groups depending on the largest power of two dividing μ_t. Consider two indexes $t < t'$ in the same group. Since $\mu_t < \mu_{t'}$ are divisible by the same largest power of two, there is a number $\nu \in (\mu_t\,..\,\mu_{t'})$ divisible by a strictly larger power of two. By definition of μ_t, we have $s_\nu \geq s_t + |C_k(S, s_t)|$ and, by definition of $\mu_{t'}$, we have $s_\nu < s_{t'}$. Consequently, $s_{t'} > s_t + |C_k(S, s_t)|$, i.e., the contexts $C_k(S, s_t)$ and $C_k(S, s_{t'})$ are disjoint. By monotonicity of $\overline{\mathsf{maxLZ}}$, this also means that $s_t + |C_k^2(S, s_t)| \leq s_{t'} + |C_k(S, s_{t'})|$. In particular, each position $s \in [1\,..\,|S|]$ is covered by at most one context $C_k(S, s_t)$ and at most two double contexts $C_k^2(S, s_t)$ for $t \in [1\,..\,m]$ belonging to a single group. □

Let $\mathcal{A}$ be an alignment between $X, Y \in \Sigma^*$. We define $\mathcal{B}_X(\mathcal{A}) = \{x : (x, y) \in \mathcal{B}(\mathcal{A})\} \cap [1\,..\,|X|]$. Moreover, for two alignments $\mathcal{A}, \mathcal{A}'$ between $X, Y \in \Sigma^*$, we define $\Delta_X(\mathcal{A}, \mathcal{A}')$ to contain $x \in [1\,..\,|X|]$ unless $(x, y) \in \mathcal{M}(\mathcal{A}) \cap \mathcal{M}(\mathcal{A}')$ for some $y \in [1\,..\,|Y|]$.

Lemma 6.10. *Let $b \in \mathbb{Z}_+$ and $\mathcal{A}, \mathcal{A}'$ be greedy alignments of strings X and Y. The positions in $\Delta_X(\mathcal{A}, \mathcal{A}')$ can be covered by at most $\mathsf{cost}(\mathcal{A}) + \frac{1}{b}\mathsf{cost}(\mathcal{A}')$ contexts $C_{\mathsf{width}(\mathcal{A})+\mathsf{width}(\mathcal{A}')+2b}(X, \cdot)$.*

Moreover, if $b > \mathsf{cost}(\mathcal{A}')$, then the positions in $\Delta_X(\mathcal{A}, \mathcal{A}')$ can be covered by contexts $C_{\mathsf{width}(\mathcal{A})+\mathsf{width}(\mathcal{A}')+2b}(X, x)$ with $x \in \mathcal{B}_X(\mathcal{A})$.

Proof. Let $w = \mathsf{width}(\mathcal{A})$ and $w' = \mathsf{width}(\mathcal{A}')$. Without loss of generality, we may trim the longest common prefix of X and Y; this is feasible because both $\mathcal{A}$ and $\mathcal{A}'$ match the common prefix, so $\Delta_X(\mathcal{A}, \mathcal{A}')$ only contains positions following the prefix. Moreover, we assume that $X \neq \varepsilon$; otherwise $\Delta_X(\mathcal{A}, \mathcal{A}') = \emptyset$ and the lemma is trivial.

In the remaining case, we are guaranteed that $1 \in \mathcal{B}_X(\mathcal{A})$. Let $\mathcal{B}_X(\mathcal{A}) = \{x_1, \dots, x_m\}$, where $1 = x_1 < \cdots < x_m$, and let $x_{m+1} = |X| + 1$. This yields a decomposition $X = X_1 \cdots X_m$ into m non-empty substrings $X_i := X[x_i \,..\, x_{i+1})$. The alignment $\mathcal{A}'$ further yields a decomposition $Y = Y_1 \cdots Y_m$ into m (possibly empty) substrings $Y_i := Y[y_i \,..\, y_{i+1})$ so that $X_i \sim_{\mathcal{A}'} Y_i$ and $|x_i - y_i| \le w'$ for $i \in [1 \,..\, m]$. Moreover, the cost of $\mathcal{A}'$ can be expressed as $\mathsf{cost}(\mathcal{A}') = \sum_{i=1}^{m} c'_i$, where c'_i denotes the cost of $\mathcal{A}'$ restricted to X_i and Y_i.

Claim 6.11. *For every $i \in [1 \,..\, m]$, the set $\Delta_X(\mathcal{A}, \mathcal{A}') \cap [x_i \,..\, x_{i+1})$ can be covered by at most $\lceil \frac{1}{b}(c'_i + 1) \rceil$ contexts $C_{w+w'+2b}(X, \cdot)$ including $C_{w+w'+2b}(X, x_i)$.*

Proof. Let $q_i = \max(\Delta_X(\mathcal{A}, \mathcal{A}') \cap [x_i \,..\, x_{i+1}))$. We may assume that $q_i \ge x_i + w + w'$; otherwise, the claim holds trivially. We shall prove that $X[x_i + 1 \,..\, q_i - (w + w')]$ can be decomposed into at most $2c'_i + 1$ phrases, each of which is a single character or has another occurrence at most $w + w'$ positions to the right. Accounting for $X[x_i]$, this yields a decomposition of $X[x_i \,..\, q_i - (w+w')]$ into at most $2c'_i + 2$ such phrases. The $\overline{\mathsf{maxLZ}}$ measure of the concatenation of every $2b$ subsequent phrases and $w + w'$ following single characters does not exceed $w + w' + 2b$. Hence, $X[x_i \,..\, q_i]$ can be covered by $\lceil \frac{1}{b}(c'_i + 1) \rceil$ contexts $C_{w+w'+2b}(X, \cdot)$ including $C_{w+w'+2b}(X, x_i)$.

Thus, it remains to construct the decomposition of $X[x_i + 1 \,..\, q_i - (w + w')]$ into phrases. By definition of $\mathcal{B}_X(\mathcal{A})$, we have $X[x_i + 1 \,..\, x_{i+1}) \simeq_{\mathcal{A}} Y[x_i + 1 + d \,..\, x_{i+1} + d)$ for some shift $d \in [-w \,..\, w]$. We consider two cases.

In the first case, we assume that $(x_i + 1, \bar{y}) \in \mathcal{A}'$ holds for some $\bar{y} \ge x_i + 1 + d$. The greedy nature of $\mathcal{A}'$ then guarantees that if $(x, y) \in \mathcal{A}'$ with $x \in [x_i + 1 \,..\, q_i)$, then $y > x + d$. We decompose $X[x_i + 1 \,..\, q_i - (w + w')]$ (which is contained in X_i) into maximal phrases $X[x \,..\, x']$ satisfying $X[x \,..\, x'] \simeq_{\mathcal{A}'} Y[x + d' \,..\, x' + d']$ for some $d' \in (d \,..\, w']$ and remaining single characters (deleted or substituted by $\mathcal{A}'$). Observe that this yields at most $1 + c'_i$ phrases and at most c'_i single characters. Moreover, we have $X[x \,..\, x'] \simeq_{\mathcal{A}'} Y[x + d' \,..\, x' + d'] \simeq_{\mathcal{A}} X[x + (d' - d) \,..\, x' + (d' - d)]$ due to $x_i + 1 \le x \le x + (d' - d)$ and $x' + (d' - d) \le x' + (w' + w) \le q_i$. Hence, each phrase $X[x \,..\, x']$ has another occurrence located $d' - d \in [1 \,..\, w + w']$ positions to the right.

In the second case, we assume that $(x_i + 1, \bar{y}) \in \mathcal{A}'$ holds for some $\bar{y} \le x_i + 1 + d$. The greedy nature of $\mathcal{A}'$ then guarantees that if $(x, y) \in \mathcal{A}'$ with $x \in [x_i + 1 \,..\, q_i)$, then $y < x + d$. We decompose $Y[x_i + 1 + d \,..\, q_i - (w + w') + d]$ (which is contained in Y_i) into maximal phrases $Y[y \,..\, y']$ satisfying $Y[y \,..\, y'] \simeq_{\mathcal{A}'} X[y - d' \,..\, y' - d']$ for some $d' \in [-w \,..\, d)$ and remaining single characters (deleted or substituted by $\mathcal{A}'$). Observe that this yields at most $1 + c'_i$ phrases and at most c'_i single characters. Moreover, we have $Y[y \,..\, y'] \simeq_{\mathcal{A}'} X[y - d' \,..\, y' - d'] \simeq_{\mathcal{A}} Y[y + (d - d') \,..\, y + (d - d')]$ due to $x_i + 1 \le x_i + 1 + d - d' \le y - d'$ and $y' - d' \le q_i - (w + w') + d - d' \le q_i$. Hence, each phrase $Y[y \,..\, y']$ has another occurrence located $d - d' \in [1 \,..\, w + w']$ characters to the right. Since $Y[x_i + 1 + d \,..\, q_i + d] = X[x_i + 1 \,..\, q_i]$, this decomposition of $Y[x_i + 1 + d \,..\, q_i - (w + w') + d]$ gives an analogous decomposition of $X[x_i + 1 \,..\, q_i - (w + w')]$. □

The set $\Delta_X(\mathcal{A}, \mathcal{A}')$ can be covered by $\sum_{i=1}^{m} \lceil \frac{1}{b}(c'_i + 1) \rceil \le m + \frac{\mathsf{cost}(\mathcal{A}')}{b} \le \mathsf{cost}(\mathcal{A}) + \frac{\mathsf{cost}(\mathcal{A}')}{b}$ contexts $C_{w+w'+2b}(X, \cdot)$. If $b \ge \mathsf{cost}(\mathcal{A}') + 1$, then the contexts starting at positions in $\mathcal{B}_X(\mathcal{A})$ are sufficient. □

Lemma 6.12. *Let $\mathcal{A}_W$ be the zip alignment of m-complete walks $W_X = (x_t)_{t=1}^{m+1}$, $W_Y = (y_t)_{t=1}^{m+1}$ over strings $X, Y \in \Sigma^*$, and let $k' \in \mathbb{Z}_+$. If $\mathcal{A}_W$ is greedy, then $\mathcal{B}_X(\mathcal{A}_W) \subseteq P_X$, where P_X is the set of positions $x \in [1 \,..\, |X|]$ such that $X[x]$ is covered by $C^2_{k'}(X, x_t)$ for some $t \in [1 \,..\, m]$ with $\perp \neq \mathsf{CE}_{k'}(W_X)[t] \neq \mathsf{CE}_{k'}(W_Y)[t]$. Moreover, for every positive integer $k \ge \mathsf{ed}(X, Y)$ such that $k' \ge \mathsf{width}(\mathcal{A}_W) + 5k$:*

- *every $\mathcal{A} \in \mathsf{GA}_k(X, Y)$ satisfies $\Delta_X(\mathcal{A}, \mathcal{A}_W) \subseteq P_X$,*

- $\mathsf{hd}(\mathsf{CE}_{k'}(X), \mathsf{CE}_{k'}(Y)) \leq c(k + \frac{1}{k}\mathsf{cost}(\mathcal{A}_W)) \log m$ *holds for a sufficiently large constant* c.

Proof. Consider a position $x \in \mathcal{B}_X(\mathcal{A}_W)$. By Lemma 6.9, there is an index $t \in [1\,..\,m]$ such that $\mathsf{CE}_{k'}(W_X)[t] \neq \bot$ and $C^2_{k'}(X, x_t)$ covers $X[x]$. To conclude that $x \in P_X$, it remains to prove $\mathsf{CE}_{k'}(W_X)[t] \neq \mathsf{CE}_{k'}(W_Y)[t]$. For a proof by contradiction, suppose that $\mathsf{CE}_{k'}(W_X)[t] = \mathsf{CE}_{k'}(W_Y)[t]$. This implies $C^2_{k'}(X, x_t) = C^2_{k'}(Y, y_t)$. Since $\mathcal{A}_W$ is greedy, we derive $X[x_t\,..\,x_t + |C^2_{k'}(X, x_t)|) \simeq_{\mathcal{A}_W} Y[y_t\,..\,y_t + |C^2_{k'}(Y, y_t)|)$, which contradicts $x \in \mathcal{B}_X(\mathcal{A}_W)$.

In the remainder of the proof, we assume that $k \geq \mathsf{ed}(X, Y)$ and $k' \geq \mathsf{width}(\mathcal{A}_W) + 5k$. Next, consider a greedy alignment $\mathcal{A}$ with $\mathsf{cost}(\mathcal{A}) \leq k$ and a position $x \in \Delta_X(\mathcal{A}, \mathcal{A}_W)$. Due to $k' \geq \mathsf{width}(\mathcal{A}_W) + \mathsf{width}(\mathcal{A}) + 4k$, Lemma 6.10 shows that there exists a position $r \in \mathcal{B}_X(\mathcal{A}_W)$ such that $C_{k'}(X, r)$ covers $X[x]$ and, by Lemma 6.9, there exists a position $t \in [1\,..\,m]$ such that $\mathsf{CE}_{k'}(X)[t] \neq \bot$ and $C_{k'}(X, x_t)$ covers $X[r-1]$. Since $\overline{\mathsf{maxLZ}}(X[x_t\,..\,r)), \overline{\mathsf{maxLZ}}(X[r\,..\,x]) \leq k'$, we conclude that both r and x are covered by $C^2_{k'}(X, x_t)$. We shall prove that $\mathsf{CE}_{k'}(X)[t] \neq \mathsf{CE}_{k'}(Y)[t]$. For a proof by contradiction, suppose that $\mathsf{CE}_{k'}(W_X)[t] = \mathsf{CE}_{k'}(W_Y)[t]$, which implies $C^2_{k'}(X, x_t) = C^2_{k'}(Y, y_t)$. Let us fix the smallest $t' \in [t\,..\,m+1]$ such that $(x_{t'}, y_{t'}) \in \mathcal{B}(\mathcal{A}_W)$. Observe that $X[x_t\,..\,x_{t'}) \simeq_{\mathcal{A}_W} Y[y_t\,..\,y_{t'})$ and, by the greedy nature of $\mathcal{A}_W$, $X[x_t\,..\,x_{t'}) = Y[y_t\,..\,y_{t'})$ is the longest common prefix of $X[x_t\,..]$ and $Y[y_t\,..]$. At the same time, due to $x_t < r \in \mathcal{B}_X(\mathcal{A}_W)$, we have $x_{t'} \leq r$, so $X[x_t\,..\,x_{t'}]$ is a prefix of $C^2_{k'}(X, x_t)$. However, $X[x_t\,..\,x_{t'}]$ is not a prefix $Y[y_t\,..]$, so $C^2_{k'}(X, x_t)$ is not a prefix of $Y[y_t\,..]$ and $C^2_{k'}(X, x_t) \neq C^2_{k'}(Y, y_t)$. The contradiction completes the proof.

Finally, we bound $\mathsf{hd}(\mathsf{CE}_{k'}(X), \mathsf{CE}_{k'}(Y))$ using several claims. Consider a set M of indices $t \in [1\,..\,m]$ such that $\mathsf{CE}_{k'}(X)[t]$ and $\mathsf{CE}_{k'}(Y)[t]$ differ on the first coordinate.

Claim 6.13. *If* $t \in M$ *satisfies* $\mathsf{CE}_{k'}(X)[t] \neq \bot$, *then*

$$[x_{t-1}\,..\,x_t + |C^2_{k'}(X, x_t)|] \cap (\mathcal{B}_X(\mathcal{A}_W) \cup \{|X| + 1\}) \neq \emptyset.$$

Proof. For a proof by contradiction, suppose that $[x_{t-1}\,..\,x_t + |C^2_{k'}(X, x_t)|] \cap (\mathcal{B}_X(\mathcal{A}_W) \cup \{|X| + 1\}) = \emptyset$. Let L be the length of the longest common prefix of $X[x_t\,..]$ and $Y[y_t\,..]$. Note that $(x_t + L, y_t + L) \in \mathcal{B}(\mathcal{A}_W)$, so $x_t + L \in \mathcal{B}_X(\mathcal{A}_W) \cup \{|X| + 1\}$. Consequently, $L > |C^2_{k'}(X, x_t)| \geq |C_{k'}(X, x_t)|$, and therefore $X[x_t\,..\,x_t + |C^2_{k'}(X, x_t)|] = Y[y_t\,..\,y_t + |C^2_{k'}(Y, y_t)|]$ and $X[x_t\,..\,x_t + |C_{k'}(X, x_t)|] = Y[y_t\,..\,y_t + |C_{k'}(Y, y_t)|]$. In particular, $C^2_{k'}(X, x_t) = C^2_{k'}(Y, y_t)$.

If $t = 1$, then we note that $\mathsf{CE}_{k'}(X)[t] \neq \bot \neq \mathsf{CE}_{k'}(Y)[t]$, so $C^2_{k'}(X, x_t) = C^2_{k'}(Y, y_t)$ contradicts $t \in M$.

In the following, we assume that $t \geq 2$. Due to $(x_{t-1}, y_{t-1}) \notin \mathcal{B}(\mathcal{A}_W)$, either $(x_{t-1}, y_{t-1}) = (x_t, y_t)$, or $(x_{t-1}, y_{t-1}) = (x_t - 1, y_t - 1)$ and $X[x_t - 1] = Y[y_t - 1]$. In either case, we have $X[x_{t-1}\,..\,x_{t-1} + |C_{k'}(X, x_{t-1})|] = Y[y_{t-1}\,..\,y_{t-1} + |C_{k'}(Y, y_{t-1})|]$ due to $X[x_{t-1}\,..\,x_t + |C_{k'}(X, x_t)|] = Y[y_{t-1}\,..\,y_t + |C_{k'}(Y, y_t)|]$ and by Observation 6.7. Consequently, $\min\{u \in [1\,..\,m] : x_u = x_{t-1} + |C_{k'}(X, x_{t-1})|\} = \min\{u \in [1\,..\,m] : y_u = y_{t-1} + |C_{k'}(Y, y_{t-1})|\}$ and, by a similar reasoning, $\min\{u \in [1\,..\,m] : x_u = x_t + |C_{k'}(X, x_t)|\} = \min\{u \in [1\,..\,m] : y_u = y_t + |C_{k'}(Y, y_t)|\}$. Hence, the assumption $\mathsf{CE}_{k'}(X)[t] \neq \bot$ implies $\mathsf{CE}_{k'}(Y)[t] \neq \bot$. Therefore, $C^2_{k'}(X, x_t) = C^2_{k'}(Y, y_t)$ contradicts $t \in M$. □

Claim 6.14. *There are* $\mathcal{O}((k + \frac{1}{k}\mathsf{cost}(\mathcal{A}_W)) \log m)$ *positions* $t \in M$ *such that* $\mathsf{CE}_{k'}(X)[t] \neq \bot$.

Proof. Let $\mathcal{O}$ be an optimum greedy alignment between X and Y. Due to $k' \geq \mathsf{width}(\mathcal{O}) + \mathsf{width}(\mathcal{A}_W) + 4k$, from Lemma 6.10 it follows that $\Delta_X(\mathcal{O}, \mathcal{A}_W)$ can be covered by $\mathcal{O}(\mathsf{cost}(\mathcal{O}) + \frac{1}{k}\mathsf{cost}(\mathcal{A}_W)) = \mathcal{O}(k + \frac{1}{k}\mathsf{cost}(\mathcal{A}_W))$ contexts $C_{k'}(X, \cdot)$. Let us fix such the smallest family $\mathcal{C}$ of contexts covering $\mathcal{B}_X(\mathcal{A}_W) \subseteq \Delta_X(\mathcal{O}, \mathcal{A}_W)$.

Define a set $R_X = \{|X|\} \cup \bigcup_{X[q\,..\,r] \in \mathcal{C}} \{q - 1, r + 1\}$ and note that $|R_X| = \mathcal{O}(|\mathcal{C}|)$. We shall prove that, if $t \in M$ and $\mathsf{CE}_{k'}(X)[t] \neq \bot$, then $C^2_{k'}(X, x_t)$ covers at least one position in R_X.

This is sufficient to derive the claim because, by Lemma 6.9, every position in X is covered by at most $\mathcal{O}(\log m)$ double contexts $C^2_{k'}(X, x_t)$ with $\mathsf{CE}_{k'}(X)[t] \neq \bot$.

By Claim 6.13, we have $[x_{t-1}\,..\,x_t + |C^2_{k'}(X, x_t)|] \cap (\mathcal{B}_X(\mathcal{A}_W) \cup \{|X|+1\}) \neq \emptyset$. If $\{|X|+1\} \in [x_{t-1}\,..\,x_t + |C^2_{k'}(X, x_t)|]$, then $|X|+1 = x_t + |C^2_{k'}(X, x_t)|$. Hence, $C^2_{k'}(X, x_t)$ covers the position $|X| \in R_X$. Thus, we may assume that $[x_{t-1}\,..\,x_t + |C^2_{k'}(X, x_t)|]$ contains a position in $\mathcal{B}_X(\mathcal{A}_W)$. Note that the fragment $X[q\,..\,r] \in \mathcal{C}$ covering that position in $\mathcal{B}_X(\mathcal{A}_W)$ satisfies $r \geq x_{t-1}$ and $q \leq x_t + |C^2_{k'}(X, x_t)|$. In particular, $r+1 \geq x_t$ and $q - 1 < x_t + |C^2_{k'}(X, x_t)|$. Now, if $C^2_{k'}(X, x_t)$ covers $q-1$ or $r+1$, then we are done. Otherwise, $q - 1 < x_t$ and $r + 1 \geq x_t + |C^2_{k'}(X, x_t)|$, so $q \leq x_t$ and $r \geq x_t + |C^2_{k'}(X, x_t)|$, i.e., $C^2_{k'}(X, x_t)$ is contained in $X[q\,..\,r]$ and, since $\overline{\mathsf{maxLZ}}$ is monotone, $\overline{\mathsf{maxLZ}}(C^2_{k'}(X, x_t)) \leq k'$. Consequently, $C^2_{k'}(X, x_t) = C_{k'}(X, x_t)$ is a suffix of X, so $C^2_{k'}(X, x_t)$ covers the position $|X| \in R_X$. □

A symmetric argument shows that there are $\mathcal{O}((k + \frac{1}{k}\mathsf{cost}(\mathcal{A}_W)) \log m)$ positions $t \in M$ such that $\mathsf{CE}_{k'}(Y)[t] \neq \bot$. Consequently, $|M| = \mathcal{O}((k + \frac{1}{k}\mathsf{cost}(\mathcal{A}_W)) \log m)$. We bound $\mathsf{hd}(\mathsf{CE}_{k'}(X), \mathsf{CE}_{k'}(Y))$ using the following claim.

Claim 6.15. $\mathsf{hd}(\mathsf{CE}_{k'}(X), \mathsf{CE}_{k'}(Y)) \leq 2|M|$.

Proof. Let $M' = \{t \in [1\,..\,m] : \mathsf{CE}_{k'}(X)[t] \neq \mathsf{CE}_{k'}(Y)[t]\}$. Note that $M \subseteq M'$, so the claim is equivalent to $|M' \setminus M| \leq |M|$. For every $t \in M' \setminus M$, we shall prove that t is not the leftmost position in M' and that the preceding position in M' belongs to M.

The assumption $t \in M' \setminus M$ implies $\mathsf{CE}_{k'}(X)[t] \neq \bot \neq \mathsf{CE}_{k'}(Y)[t]$. We define t' as the largest position in $[1\,..\,t)$ such that $\mathsf{CE}_{k'}(X)[t'] \neq \bot$ or $\mathsf{CE}_{k'}(Y)[t'] \neq \bot$. To see that t' is well defined, note that $t > 1$ and $\mathsf{CE}_{k'}(X)[1] \neq \bot$.

Now, suppose that $t' \notin M$. Consequently, we have $\mathsf{CE}_{k'}(X)[t'] \neq \bot \neq \mathsf{CE}_{k'}(Y)[t']$. Due to $\mathsf{CE}_{k'}(X)[t'+1\,..\,t) = \bot^{t-t'-1}$, Lemma 6.9 implies that $C_{k'}(X, x_{t'})$ covers $X[x_t - 1]$, that is, $\overline{\mathsf{maxLZ}}(X[x_{t'}\,..\,x_t)) \leq k'$, and hence $X[x_{t'}\,..\,x_t]$ is a prefix of $C^2_{k'}(X, x_{t'})$. Moreover, $C^2_{k'}(X, x_{t'}) = C^2_{k'}(Y, y_{t'})$ implies $X[x_{t'}\,..\,x_t] \simeq_{\mathcal{A}_W} Y[y_{t'}\,..\,y_{t'} + x_t - x_{t'}]$ by the greedy nature of $\mathcal{A}_W$. As $(x_t, y_t) \in \mathcal{A}_W$, we conclude that $y_t = y_{t'} + x_t - x_{t'}$. At the same time, since $\mathsf{CE}_{k'}(X)[t'+1\,..\,t) = \bot^{t-t'-1} = \mathsf{CE}_{k'}(Y)[t'+1\,..\,t)$, we have $\mathsf{CE}_{k'}(X)[t] = (\mathsf{LZ}(\overline{C^2_{k'}(X, x_t)}), x_t - x_{t'})$ and $\mathsf{CE}_{k'}(Y)[t] = (\mathsf{LZ}(\overline{C^2_{k'}(Y, y_t)}), y_t - y_{t'})$. Thus, $\mathsf{CE}_{k'}(X)[t] = \mathsf{CE}_{k'}(Y)[t]$, which contradicts $t \in M'$.

Consequently, $t' \in M$. In this case, $M' \cap [t'\,..\,t] = \{t', t\}$, so t' is the position of M' preceding t. Our goal was to show that such a position exists and belongs to M, so this completes the proof of the claim. □

Overall, we conclude that $\mathsf{hd}(\mathsf{CE}_{k'}(X), \mathsf{CE}_{k'}(Y)) \leq 2|M| = \mathcal{O}((k + \frac{1}{k}\mathsf{cost}(\mathcal{A}_W)) \log m)$. □

Lemma 6.16. *Given integers $n \geq k \geq 1$, a seed r of $\mathcal{O}(\log^2 n)$ random bits, and streaming access to a string $S \in \Sigma^{\leq n}$, the string $\mathsf{CE}_k(\mathsf{W}(S, n, r))$ can be computed in $\tilde{\mathcal{O}}(k)$ space and $\tilde{\mathcal{O}}(nk)$ time.*

Proof. Let us start with an auxiliary subroutine:

Claim 6.17. *There is a streaming algorithm that computes $\mathsf{D}(C_k(S, p))$ for subsequent positions $p \in [1\,..\,|S|]$. The algorithm uses $\tilde{\mathcal{O}}(k)$ space and $\tilde{\mathcal{O}}(nk)$ time. Moreover, if $C_k(S, p) = S[p\,..\,q)$, then $\mathsf{D}(C_k(S, p))$ is reported while the algorithm processes $S[q]$ (or the end-of-string token if $q = |S| + 1$).*

Proof. The algorithm maintains a fragment $S[p\,..\,q)$ satisfying $\overline{\mathsf{maxLZ}}(S[p\,..\,q)) \leq k$ and the encoding $\mathsf{D}(S[p\,..\,q))$. Initially, $S[p\,..\,q) = S[1\,..\,1) = \varepsilon$.

In each iteration, we read $S[q]$, compute $\mathsf{D}(S[p\,..\,q])$ (using Proposition 4.2(i)), and check whether $|\mathsf{LZ}(\overline{S[p\,..\,q]})| \leq k$ (using Proposition 4.2(f)). By Observation 6.7(a), this condition

is equivalent to $\overline{\mathsf{maxLZ}}(S[p\,..\,q]) \le k$. If $\overline{\mathsf{maxLZ}}(S[p\,..\,q]) \le k$, we discard $\mathsf{D}(S[p\,..\,q))$ and increment q. Otherwise, we are guaranteed that $S[p\,..\,q) = C_k(S,p)$, so we output $\mathsf{D}(S[p\,..\,q))$, compute $\mathsf{D}(S[p+1\,..\,q))$ (using Proposition 4.2(g)), discard $\mathsf{D}(S[p\,..\,q))$ and $\mathsf{D}(S[p\,..\,q])$, and increment p. By Observation 6.7(b), we are guaranteed that the invariant $\overline{\mathsf{maxLZ}}(S[p\,..\,q)) \le k$ remains satisfied. In the special case of $q = |S|+1$, we proceed as if $\overline{\mathsf{maxLZ}}(S[p\,..\,q]) > k$.

By Observation 6.7(c), we store $\mathsf{D}(X)$ only for strings X satisfying $|\overline{\mathsf{maxLZ}}(X)| \le k+1$, so the space usage and the per-iteration running time is $\tilde{\mathcal{O}}(k)$. Each iteration increments either p or q, so the algorithm reads S in a streaming fashion and the amortized running time is $\tilde{\mathcal{O}}(k)$ per character. □

We maintain two instances of the algorithm of Claim 6.17 and two instances of the algorithm of Proposition 3.16. We feed the first instance of Claim 6.17 with the input stream S, obtaining $\mathsf{D}(C_k(S,q))$ for subsequent positions $q \in [1\,..\,|S|]$. Upon retrieving $\mathsf{D}(C_k(S,q))$, we extract $S[q]$ using Proposition 4.2(b) and forward $S[q]$ to the first instance of Proposition 3.16, which lists indices $v \in [1\,..\,3n]$ such that $s_v = q$. We also forward $S[q]$ to the second instance of Claim 6.17, obtaining $\mathsf{D}(C_k(S,p))$ for all subsequent positions p such that $\mathsf{D}(C_k(S,p)) = S[p\,..\,q)$. Upon retrieving $\mathsf{D}(C_k(S,p))$, we extract $S[p]$ using Proposition 4.2(b) and forward $S[p]$ to the second instance of Proposition 3.16, which lists indices $t \in [1\,..\,3n]$ such that $s_t = q$. For each such position t, we have $C^2_k(S,s_t) = C_k(S,p)C_k(S,q)$. We compute $\mu_t = \mu(t, \min\{v \in [1\,..\,3n] : s_v = q\})$ based on the output of the first instance of Proposition 3.16. If $\mu_t > \mu_{t-1}$, we construct $\mathsf{LZ}\left(\overline{C^2_k(S,s_t)}\right)$ using Proposition 4.2(i) and Proposition 4.2(f).

By Propositions 4.2 and 3.16 and Claim 6.17, the algorithm uses $\tilde{\mathcal{O}}(k)$ space and $\tilde{\mathcal{O}}(nk)$ time. □

6.3 Applications of Hamming distance sketches

Let us start by reminding the fingerprints (sketches) for testing string equality.

Fact 6.18 (see e.g. [38])**.** *There exists a fingerprint ψ (parameterized by an integer $n \in \mathbb{Z}_+$, a threshold δ with $1 \ge \delta \ge n^{-\mathcal{O}(1)}$, an alphabet $\Sigma = [0\,..\,n^{\mathcal{O}(1)})$, and a seed of $\mathcal{O}(\log n)$ random bits) such that:*

1. *The fingerprint $\psi(S)$ of a string $S \in \Sigma^{\le n}$ takes $\mathcal{O}(\log \delta^{-1})$ bits. Given streaming access to S, it can be constructed in $\mathcal{O}(|S|)$ time using $\mathcal{O}(\log n)$ bits of space.*
2. *For all strings $X, Y \in \Sigma^{\le n}$, we have $\Pr[\psi(X) = \psi(Y)] \le \delta$ if $X \ne Y$ (and $\psi(X) = \psi(Y)$ otherwise).*

For two equal-length strings X, Y, the set of *mismatch positions* is defined as $\mathsf{MP}(X,Y) = \{i \in [1\,..\,|X|] : X[i] \ne Y[i]\}$ and the *mismatch information* $\mathsf{MI}(X,Y) = \{(i, X[i], Y[i]) : i \in \mathsf{MP}(X,Y)\}$. Below, we adapt the Hamming sketches of [16] to large alphabets.

Theorem 6.19. *For every constant $\delta \in (0,1)$, there exists a sketch sk^H_k (parameterized by integers $n \ge k \ge 1$, an alphabet $\Sigma = [0\,..\,\sigma)$, and a seed of $\mathcal{O}(\log(n \log \sigma))$ random bits) such that:*

1. *The sketch $\mathsf{sk}^H_k(S)$ of a string $S \in \Sigma^n$ takes $\mathcal{O}(k \log(n\sigma))$ bits. Given streaming access to S, it can be constructed in $\mathcal{O}(n \log(n\sigma) \log(n \log \sigma))$ time using $\mathcal{O}(k \log(n\sigma))$ bits of space.*
2. *There exists a decoding algorithm that, given $\mathsf{sk}^H_k(X)$ and $\mathsf{sk}^H_k(Y)$ for strings $X, Y \in \Sigma^n$, with probability at least $1-\delta$ either returns $\mathsf{MI}(X,Y)$ or certifies that $\mathsf{hd}(X,Y) > k$. The algorithm uses $\mathcal{O}(k \log(n\sigma) \log^2(n \log \sigma))$ time and $\mathcal{O}(k \log(n\sigma))$ bits of space.*

Proof. The construction of [16] satisfies the required conditions provided that $\sigma = n^{\mathcal{O}(1)}$. Henceforth, we assume without loss of generality that σ is a power of two satisfying $\sigma \ge n \log \sigma$.

We interpret each character in Σ as a block of $b = \lceil \log_{n \log \sigma} \sigma \rceil$ characters in $[0 \mathinner{..} n \log \sigma)$. Moreover, we consider the fingerprints ψ of Fact 6.18 with $\delta_{6.18} = \frac{\delta}{2n}$ and $n_{6.18} = \sigma_{6.18} = n \log \sigma$. This construction uses $\mathcal{O}(\log n_{6.18}) = \mathcal{O}(\log(n \log \sigma))$ random bits and produces fingerprints of $\mathcal{O}(\log \delta_{6.18}^{-1}) = \mathcal{O}(\log n)$ bits.

Given a string $S \in \Sigma^n$, we define a string $\bar{S}[1 \mathinner{..} nb]$ so that $\bar{S}[ib - j + 1] = (S[i][j], \psi(S[i]))$ for $i \in [1 \mathinner{..} |S|]$ and $j \in [1 \mathinner{..} b]$. Consequently, we set $\mathsf{sk}_k^H(S) := \mathsf{sk}_{\bar{k}}^H(\bar{S})$ using the the sketch of [16] with $\bar{\delta} = \frac{\delta}{2}$, $\bar{n} = nb$, $\bar{k} = kb$, and $\bar{\sigma} \leq n^{\mathcal{O}(1)} \log \sigma$. Note that this is feasible since $\bar{\sigma} \leq n^{\mathcal{O}(1)} \leq \bar{n}^{\mathcal{O}(1)}$ if $n \leq \log \sigma$ and $\bar{\sigma} \leq \log^{\mathcal{O}(1)} \sigma \leq (\log_{\log^2 \sigma} \sigma)^{\mathcal{O}(1)} \leq b^{\mathcal{O}(1)} \leq \bar{n}^{\mathcal{O}(1)}$ otherwise. This construction uses $\mathcal{O}(\log \bar{n}) = \mathcal{O}(\log(n \log \sigma))$ further random bits and produces a sketch of $\mathcal{O}(\bar{k} \log \bar{n}) = \mathcal{O}(kb \log(nb)) = \mathcal{O}(k \log_{n \log \sigma} \sigma \log(n \log \sigma) = \mathcal{O}(k \log \sigma)$ bits.

The auxiliary string $\bar{S}$ is constructed in $\mathcal{O}(\bar{n}) = \mathcal{O}(n \log \sigma)$ time using $\mathcal{O}(\log(n\sigma))$ bits of space. The stream representing $\bar{S}$ is passed to the encoding algorithm of [16], which takes $\mathcal{O}(\bar{n} \log^2 \bar{n}) = \mathcal{O}(n \log_{n \log \sigma} \sigma \log^2(n \log \sigma)) = \mathcal{O}(n \log(n\sigma) \log(n \log \sigma))$ time and $\mathcal{O}(\bar{k} \log \bar{n}) = \mathcal{O}(k \log \sigma)$ bits of space.

The decoding algorithm, given $\mathsf{sk}_{\bar{k}}^H(\bar{X})$ and $\mathsf{sk}_{\bar{k}}^H(\bar{Y})$, runs the decoding algorithm of [16]. If the latter certifies $\mathsf{hd}(\bar{X}, \bar{Y}) > \bar{Y}$, we certify that $\mathsf{hd}(X, Y) > k$. Otherwise, we interpret the output as $\mathsf{MI}(\bar{X}, \bar{Y})$. For each position $i \in [1 \mathinner{..} |X|]$ such that $(ib - b \mathinner{..} ib] \subseteq \mathsf{MP}(\bar{X}, \bar{Y})$ we retrieve $X[i][j]$ and $Y[i][j]$ for each $j \in [1 \mathinner{..} b]$ from $(ib - j + 1, \bar{X}[ib - j + 1], \bar{Y}[ib - j + 1]) \in \mathsf{MI}(\bar{X}, \bar{Y})$, and then we output $(i, X[i], Y[i]) \in \mathsf{MI}(X, Y)$.

As for correctness, with at most $\delta_{6.18} + n\bar{\delta} = \delta$ probability loss, we may assume that the decoder of [16] is successful and that, for all $i \in [1 \mathinner{..} |X|]$, we have $\psi(X[i]) = \psi(Y[i])$ if and only if $X[i] = Y[i]$. The latter assumption yields $\mathsf{MP}(\bar{X}, \bar{Y}) = \bigcup_{i \in \mathsf{MP}(X,Y)} (ib - b \mathinner{..} ib]$ and thus $\mathsf{hd}(\bar{X}, \bar{Y}) = b \cdot \mathsf{hd}(X, Y)$. Hence, we correctly certify $\mathsf{hd}(X, Y) > k$ if $\mathsf{hd}(\bar{X}, \bar{Y}) > \bar{k}$, and we correctly reconstruct $\mathsf{MI}(X, Y)$ otherwise.

The decoder uses $\mathcal{O}(\bar{k} \log^3 \bar{n}) = \mathcal{O}(k \log_{n \log \sigma} \sigma \log^3(n \log \sigma)) = \mathcal{O}(k \log \sigma \log^2(n \log \sigma))$ time and $\mathcal{O}(\bar{k} \log \bar{n}) = \mathcal{O}(k \log \sigma)$ bits of space. □

Next, consider an alphabet $\hat{\Sigma} := \Sigma \times \mathbb{Z}_{\geq 0}$ and a function $\pi : \hat{\Sigma} \to \mathbb{Z}_{\geq 0}$ defined so that $\pi((a, v)) = v$ for $(a, v) \in \hat{\Sigma}$ and $\pi(S) = \sum_{i=1}^{|S|} \pi(S[i])$ for $S \in \hat{\Sigma}^*$. For two strings $X, Y \in \hat{\Sigma}^*$ of the same length, we define the *prefix mismatch information*

$$\mathsf{PMI}(X, Y) = \{(i, \pi(X[1 \mathinner{..} i)), \pi(Y[1 \mathinner{..} i))) : i \in \mathsf{MP}(X, Y)\}.$$

Proposition 6.20. *For every constant $\delta \in (0, 1)$, there exists a sketch sk_k^P (parameterized by integers $n \geq k \geq 1$, an alphabet $\hat{\Sigma} = [1 \mathinner{..} \sigma] \times [0 \mathinner{..} n^{\mathcal{O}(1)}]$, and a seed of $\mathcal{O}(\log(n \log \sigma))$ random bits) such that:*

1. *The sketch $\mathsf{sk}_k^P(S)$ of a string $S \in \hat{\Sigma}^n$ takes $\mathcal{O}(k \log^2 n)$ bits. Given streaming access to S, it can be constructed in $\mathcal{O}(n \log n \log(n \log \sigma) + n \log^2 n)$ time using $\mathcal{O}(k \log^2 n + \log n \log(n \log \sigma))$ bits of space.*
2. *There exists a decoding algorithm that, given the sketches $\mathsf{sk}_k^P(X)$ and $\mathsf{sk}_k^P(Y)$ of strings $X, Y \in \hat{\Sigma}^n$ satisfying $\pi(X), \pi(Y) < n$, with probability at least $1 - \delta$ either returns $\mathsf{PMI}(X, Y)$ or certifies that $\mathsf{hd}(X, Y) > k$. The algorithm uses $\mathcal{O}(k \log^4 n)$ time and $\mathcal{O}(k \log^2 n)$ bits of space.*

Proof. Let us fix a complete binary tree $\mathcal{T}$ with n leaves, numbered with $[1 \mathinner{..} n]$ in the left-to-right order, and let $v_1, \ldots, v_{2n-1}$ denote the nodes of $\mathcal{T}$ in the pre-order. For each node $v_i \in \mathcal{T}$, let $[p_i \mathinner{..} q_i) \subseteq [1 \mathinner{..} n]$ be the indices of the leaves in the subtree of v_i. Consider the fingerprints ψ of Fact 6.18 parameterized by $\delta_{6.18} = \frac{\delta}{4n-2}$, $n_{6.18} = \mathcal{O}(n \log(n\sigma))$, and $\sigma_{6.18} = 2$: Given a string $S \in \hat{\Sigma}^n$, we define a string $\mathcal{T}(S)[1 \mathinner{..} 2n)$ so that $\mathcal{T}(S)[i] = (\pi(S[p_i \mathinner{..} q_i)), \psi(S[p_i \mathinner{..} q_i)))$ for every $i \in [1 \mathinner{..} 2n)$, where ψ expands each character of $\hat{\Sigma}$ into a sequence of $\mathcal{O}(\log(n\sigma))$ bits.

We set $\mathsf{sk}^P_k(S) := \mathsf{sk}^H_{k_{6.19}}(\mathcal{T}(S))$, where $\mathsf{sk}^H_{k_{6.19}}$ is the sketch of Theorem 6.19 with $\delta_{6.19} = \frac{\delta}{2}$, $n_{6.19} = 2n-1$, $k_{6.19} = \min(k\lceil\log(2n)\rceil, n_{6.19})$, and $\sigma_{6.19} \le 2^{\mathcal{O}(\log(n/\delta_{6.18}))} \le n^{\mathcal{O}(1)}$.

This construction uses $\mathcal{O}(\log n_{6.18} + \log(n_{6.19}\log\sigma_{6.19})) = \mathcal{O}(\log(n\log\sigma))$ random bits and produces sketches of $\mathcal{O}(k_{6.19}\log(n_{6.19}\sigma_{6.19})) = \mathcal{O}(k\log^2 n)$ bits.

The encoding algorithm transforms S into $\mathcal{T}(S)$ and feeds $\mathcal{T}(S)$ into the encoding procedure of Theorem 6.19. Construction of $\mathcal{T}(S)$ is organized in $\mathcal{O}(\log n)$ layers, each responsible for nodes $v_i \in \mathcal{T}$ at a fixed level. The intervals $[p_i\,..\,q_i)$ corresponding to these nodes are disjoint so, at any time, a layer produces a single character $\mathcal{T}(S)[i]$ and, by Fact 6.18, spends $\mathcal{O}(n\log\sigma)$ amortized time and uses $\mathcal{O}(\log n)$ bits of space to process a single character $S[j]$. Since the tree $\mathcal{T}$ is linearized in the post-order fashion, all levels can read the string S with a common left-to-right pass and outputting $\mathcal{T}(S)$ does not require any buffers. Overall, construction of $\mathcal{T}(S)$ takes $\mathcal{O}(n\log n\log(n\log\sigma))$ time and uses $\mathcal{O}(\log n\log(n\log\sigma))$ bits of space. The encoder of Theorem 6.19 takes $\mathcal{O}(n_{6.19}\log(n_{6.19}\sigma_{6.19})\log(n_{6.19}\log\sigma_{6.19})) = \mathcal{O}(n\log^2 n)$ time and uses $\mathcal{O}(k_{6.19}\log(n_{6.19}\sigma_{6.19})) = \mathcal{O}(k\log^2 n)$ bits of space.

The decoding algorithm first retrieves $\mathsf{MI}(\mathcal{T}(X),\mathcal{T}(Y))$ from $\mathsf{sk}^H_{k_{6.19}}(\mathcal{T}(X))$ and $\mathsf{sk}^H_{6.19}(\mathcal{T}(Y))$ using the decoder of Theorem 6.19. If the output of that subroutine can be interpreted as $\mathsf{MI}(\mathcal{T}(X),\mathcal{T}(Y))$, then we use Claim 6.21 below to retrieve $\mathsf{MI}(X,Y)$. If the size of the obtained set does not exceed k, we return the set. In the remaining cases, we certify that $\mathsf{hd}(X,Y) > k$.

Claim 6.21. *For every $X,Y \in \hat{\Sigma}^n$, the set $\mathsf{PMI}(X,Y)$ can be extracted from $\mathsf{MI}(\mathcal{T}(X),\mathcal{T}(Y))$ in time and space $\mathcal{O}(\mathsf{hd}(X,Y)\log n)$ with success probability at least $1-\frac{\delta}{2}$.*

Proof. We assume that, for every $i \in [1\,..\,2n)$, the equality $\psi(X[p_i\,..\,q_i)) = \psi(Y[p_i\,..\,q_i))$ implies $X[p_i\,..\,q_i) = Y[p_i\,..\,q_i)$. By Fact 6.18, each of the implications fails with probability at most $\delta_{6.18}$, so the overall failure probability can be bounded by $\delta_{6.18}\cdot(2n-1) = \frac{\delta}{2}$.

Our assumption yields

$$\mathsf{MP}(\mathcal{T}(X),\mathcal{T}(Y)) = \{i \in [1\,..\,2n) : [p_i\,..\,q_i) \cap \mathsf{MP}(X,Y) \neq \emptyset\}.$$

In particular, $\mathsf{MP}(X,Y) = \{p_i : v_i \text{ is a leaf and } i \in \mathsf{MP}(\mathcal{T}(X),\mathcal{T}(Y))\}$. Thus, it remains to describe how to extract $\pi(X[1\,..\,p))$ and $\pi(Y[1\,..\,p))$ for each $p \in \mathsf{MP}(X,Y)$; by symmetry, we focus on $\pi(X[1\,..\,p))$. For this, we process subsequent nodes v_i on the path from the root of $\mathcal{T}$ to the leaf v_j such that $\{p\} = [p_j\,..\,q_j)$ maintaining $\pi(X[1\,..\,p_i))$. Note that the values $\pi(X[p_i\,..\,q_i))$ can be extracted from $\mathsf{MI}(\mathcal{T}(X),\mathcal{T}(Y))$ due to $p \in [p_i\,..\,q_i)$.

If v_i is the root, then $\pi(X[1\,..\,p_i)) = \pi(\varepsilon) = 0$. If v_i is the left child of $v_{i'}$, then $p_i = p_{i'}$, so $\pi(X[1\,..\,p_i)) = \pi(X[1\,..\,p_{i'}))$ has already been computed. If v_i is the right child of $v_{i'}$, then $q_i = q_{i'}$, so $\pi(X[1\,..\,p_i)) + \pi(X[p_i\,..\,q_i)) = \pi(X[1\,..\,q_i)) = \pi(X[1\,..\,q_{i'})) = \pi(X[1\,..\,p_{i'})) + \pi(X[p_{i'}\,..\,q_{i'}))$. Consequently, $\pi(X[1\,..\,p_i))$ can be retrieved from $\pi(X[1\,..\,p_{i'}))$, which has already been computed, as well as $\pi(X[p_i\,..\,q_i))$ and $\pi(X[p_{i'}\,..\,q_{i'}))$, which are available in $\mathsf{MI}(\mathcal{T}(X),\mathcal{T}(Y))$. When this process reaches v_j, it results in the sought value $\pi(X[1\,..\,p)) = \pi(X[1\,..\,p_j))$. □

It remains to analyze correctness of the decoding algorithm. The decoder of Theorem 6.19 fails with probability at most $\delta_{6.19} = \frac{\delta}{2}$ and the procedure of Claim 6.21 fails with probability at most $\frac{\delta}{2}$. Thus, with at most δ probability loss, we may assume that both calls are successful.

If $\mathsf{hd}(\mathcal{T}(X),\mathcal{T}(Y)) \le k_{6.19}$, then the decoder of Theorem 6.19 retrieves $\mathsf{MI}(\mathcal{T}(X),\mathcal{T}(Y))$ and the procedure of Claim 6.21 results in $\mathsf{MI}(X,Y)$. Depending on whether $\mathsf{hd}(X,Y) = |\mathsf{MI}(X,Y)| \le k$ or not, our decoding algorithm thus correctly returns $\mathsf{MI}(X,Y)$ or certifies that $\mathsf{hd}(X,Y) > k$.

Otherwise, the algorithm of Theorem 6.19 certifies $\mathsf{hd}(\mathcal{T}(X),\mathcal{T}(Y)) > k_{6.19}$, and the whole decoding procedure certifies $\mathsf{hd}(X,Y) > k$. This is correct because of $\mathsf{hd}(\mathcal{T}(X),\mathcal{T}(Y)) \le$

$\mathsf{hd}(X,Y)\lceil\log(2n)\rceil$. To see this, observe that $[p_i\,..\,q_i)\cap\mathsf{MP}(X,Y)=\emptyset$ implies $X[p_i\,..\,q_i)=Y[p_i\,..\,q_i)$ and $\mathcal{T}(X)[i]=\mathcal{T}(Y)[i]$. However, every leaf of $\mathcal{T}$ has at most $\lceil\log(2n)\rceil$ ancestors (including itself). Consequently, for every $j\in\mathsf{MP}(X,Y)$, there are at most $\lceil\log(2n)\rceil$ nodes v_i such that $j\in[p_i\,..\,q_i)$. □

6.4 Edit Distance Sketches

We are now ready to show the main result of this section.

Theorem 3.17. *For every constant $\delta\in(0,1)$, there is a sketch sk_k^E (parametrized by integers $n\geq k\geq 1$, an alphabet $\Sigma=[0\,..\,n^{\mathcal{O}(1)})$, and a seed of $\mathcal{O}(\log^2 n)$ random bits) such that:*

(a) The sketch $\mathsf{sk}_k^E(S)$ of a string $\Sigma^{\leq n}$ takes $\mathcal{O}(k^2\log^3 n)$ bits. Given streaming access to S, it can be constructed in $\tilde{\mathcal{O}}(nk)$ time using $\tilde{\mathcal{O}}(k^2)$ space.

(b) There exists an $\tilde{\mathcal{O}}(k^2)$-space decoding algorithm that, given $\mathsf{sk}_k^E(X),\mathsf{sk}_k^E(Y)$ for $X,Y\in\Sigma^{\leq n}$, with probability at least $1-\delta$ outputs $\mathsf{GR}_k(X,Y)$ and $\min(\mathsf{ed}(X,Y),k+1)$. Retrieving $\mathsf{GR}_k(X,Y)$ costs $\tilde{\mathcal{O}}(k^5)$ time, whereas computing $\min(\mathsf{ed}(X,Y),k+1)$ costs $\tilde{\mathcal{O}}(k^3)$ time.

Proof. Let $c_{3.16}$ be the constant of Proposition 3.16 for $\delta_{3.16}=\frac{\delta}{3}$, and let $c_{6.12}$ be the constant of Lemma 6.12. We shall use $\mathsf{CE}_{k'}(W)$ with $k'=(c_{3.16}+5)k$ and $W=\mathsf{W}(n,r,S)$, where r is a random seed of $\mathcal{O}(\log^2 n)$ bits. Observe that the alphabet of $\mathsf{CE}_{k'}(W)$ can be interpreted as $[0\,..\,n^{\mathcal{O}(k)})\times[0\,..\,n]$ because the $|\mathsf{LZ}(\overline{C^2_{k'}(S,s)})|=\mathcal{O}(k)$ for each $s\in[1\,..\,|S|]$. We shall use Theorem 6.19 and Proposition 6.20 with $n_{6.19}=n_{6.20}=3n$, $\delta_{6.19}=\delta_{6.20}=\frac{\delta}{3}$, $k_{6.19}=k_{6.20}=\min(n_{6.19},\lfloor c_{6.12}(1+c_{3.16})k\log(3n)\rfloor)$, and $\sigma_{6.19}=\sigma_{6.20}=n^{\mathcal{O}(k)}$.

The sketch $\mathsf{sk}_k^E(S)$ of a string $S\in\Sigma^{\leq n}$ consists of $|S|$ as well as the sketches $\mathsf{sk}^H_{k_{6.19}}(\mathsf{CE}_{k'}(W))$ and $\mathsf{sk}^P_{k_{6.20}}(\mathsf{CE}_{k'}(W))$. This construction uses

$$\mathcal{O}(\log^2 n+\log(n_{6.19}\log\sigma_{6.19})+\log(n_{6.20}\log\sigma_{6.20}))=\mathcal{O}(\log^2 n+\log(nk\log n))=\mathcal{O}(\log^2 n)$$

random bits and produces sketches of bit-size

$$\mathcal{O}(k_{6.19}\log(n_{6.19}\sigma_{6.19})+k_{6.20}\log^2 n_{6.20})=\mathcal{O}(k\log n\log(n^{\mathcal{O}(k)})+k\log^3 n)=\mathcal{O}(k^2\log^3 n).$$

The encoding algorithm uses Lemma 6.16 to transform the input stream representing S to an auxiliary stream representing $\mathsf{CE}_{k'}(W)$, which we forward to the encoders constructing $\mathsf{sk}^H_{k_{6.19}}(\mathsf{CE}_{k'}(S))$ and $\mathsf{sk}^P_{k_{6.20}}(\mathsf{CE}_{k'}(W))$. Thus, it takes $\tilde{\mathcal{O}}(nk'+n_{6.19}\log\sigma_{6.19}+n_{6.20})=\tilde{\mathcal{O}}(nk)$ time and uses $\tilde{\mathcal{O}}(k'+k_{6.19}\log\sigma_{6.19}+k_{6.20})=\tilde{\mathcal{O}}(k^2)$ space.

Decoding Algorithm Let $\mathcal{A}_{\mathsf{W}}$ be the zip alignment of walks $W_X=(x_t)_{t=1}^{3n+1}=\mathsf{W}(n,r,X)$ and $W_Y=(y_t)_{t=1}^{3n+1}=\mathsf{W}(n,r,Y)$ over strings $X,Y\in\Sigma^{\leq n}$.

Given sketches $\mathsf{sk}_k^E(X),\mathsf{sk}_k^E(Y)$, we run the decoders for $\mathsf{sk}^H_{k_{6.19}}(\mathsf{CE}_{k'}(W_X)),\mathsf{sk}^H_{k_{6.19}}(\mathsf{CE}_{k'}(W_Y))$ and $\mathsf{sk}^P_{k_{6.20}}(\mathsf{CE}_{k'}(W_X)),\mathsf{sk}^P_{k_{6.20}}(\mathsf{CE}_{k'}(W_Y))$. We certify $\mathsf{ed}(X,Y)>k$ if either procedure certifies $\mathsf{hd}(\mathsf{CE}_{k'}(W_X),\mathsf{CE}_{k'}(W_Y))>k_{6.19}=k_{6.20}$. Otherwise, the decoder interprets the outputs as $\mathsf{MI}(\mathsf{CE}_{k'}(W_X),\mathsf{CE}_{k'}(W_Y))$ and $\mathsf{PMI}(\mathsf{CE}_{k'}(W_X),\mathsf{CE}_{k'}(W_Y))$, respectively. In this case, we construct $\mathsf{E}^M(X,Y)$ for $M=\{(x,y)\in\mathcal{M}_{X,Y}(\mathcal{A}_{\mathsf{W}}):x\in P_X\text{ or }y\in P_Y\}$, where P_X and P_Y are defined in the statement of Lemma 6.12. Finally, we compute $\mathsf{GR}_k(X,Y)$ using Proposition 5.12 or $\min(\mathsf{ed}(X,Y),k+1)$ using Corollary 5.5.

Claim 6.22. *Given $\mathsf{MI}(\mathsf{CE}_{k'}(W_X),\mathsf{CE}_{k'}(W_Y))$ and $\mathsf{PMI}(\mathsf{CE}_{k'}(W_X),\mathsf{CE}_{k'}(W_Y))$, the encoding $\mathsf{E}^M(X,Y)$ can be constructed in $\tilde{\mathcal{O}}(k^3)$ time using $\tilde{\mathcal{O}}(k^2)$ space. Moreover, $|\mathsf{LZ}(X^MY^M)|=\tilde{\mathcal{O}}(k^2)$.*

Proof. We proceed in two phases. In the first phase, we construct the compressed representation $\mathsf{D}(X')$ of a string $X' \in (\Sigma \cup \{\#\})^{|X|}$ such that $X'[x] = X[x]$ if $x \in P_X$ and $X'[x] = \#$ otherwise. We initialize $X' := \#^{|X|}$ (via $\mathsf{LZ}(\#^{|X|})$ using Proposition 4.2(h)). Next, we iterate over positions $t \in [1\,..\,3n]$ such that $\bot \neq \mathsf{CE}_{k'}(W_X)[t] \neq \mathsf{CE}_{k'}(W_Y)[t]$. We retrieve x_t and $\mathsf{LZ}\big(\overline{C^2_{k'}(X, x_t)}\big)$ from the mismatch information, convert $\mathsf{LZ}\big(\overline{C^2_{k'}(X, x_t)}\big)$ to $\mathsf{D}(C^2_{k'}(X, x_t))$ (Proposition 4.2(h)), and update $\mathsf{D}(X')$, setting $X'[x_t\,..\,x_t + |C^2_{k'}(X, x_t)|) := C^2_{k'}(X, x_t)$ (Proposition 4.2(g)(i)). We also symmetrically construct the compressed representation $\mathsf{D}(Y')$ of a string $Y' \in (\Sigma \cup \{\#\})^{|Y|}$ such that $Y'[y] = Y[y]$ if $y \in P_Y$ and $Y'[y] = \#$ otherwise.

In the second phase, we convert $\mathsf{D}(X')$ to $\mathsf{D}(X^M)$. Here, the goal is make sure that $X^M[x] = \#$ not only for $x \in P_X$, but also when $X[x] \simeq_{\mathcal{A}_\mathsf{W}} Y[y]$ for some $y \in P_Y$. For this, we iterate over dummy segments $Y'[y\,..\,y')$. By Lemma 6.12, $\mathcal{A}_\mathsf{W}$ matches $Y[y\,..\,y')$ to a fragment of $X[x\,..\,x')$. Hence, we shall identify the shift $x - y$ and set $X^M[x\,..\,x') := \#^{x'-x}$ (Proposition 4.2(h)(g)(i)). Let $\mathsf{MP}_\mathsf{CE} := \mathsf{MP}(\mathsf{CE}_{k'}(W_X), \mathsf{CE}_{k'}(W_Y))$. Define $u, v \in [1\,..\,3n+1]$ so that $u = v = 3n+1$ if $\pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,t]) \neq y'$ for all $t \in \mathsf{MP}_\mathsf{CE}$; otherwise, u is the smallest index in MP_CE with $\pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,u]) = y'$, whereas v is the smallest index MP_CE with $\pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,v)) = \pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,u))$. We claim that $x - y = \pi(\mathsf{CE}_{k'}(W_X)[1\,..\,v)) - \pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,v))$.

If $y' \leq |Y|$, then $y' - 1 \notin P_Y$ and $y' \in P_Y$. Hence, $Y[y']$ is covered by $C^2_{k'}(Y, y_t)$ for some $t \in \mathsf{MP}_\mathsf{CE}$ with $\mathsf{CE}_{k'}(W_X)[t] \neq \bot$, whereas $Y[y'-1]$ is not covered by $C^2_{k'}(Y, y_t)$. Hence, there is $t \in \mathsf{MP}_\mathsf{CE}$ with $y_t = y'$, and therefore $y_u = y'$. Similarly, $y_u = y'$ holds if $y' = |Y| + 1$. Let $w \in [1\,..\,3n]$ be the smallest index such that $\mathsf{CE}_{k'}(W_Y)[w] \neq \bot$ and $C^2_{k'}(Y, y_t)$ covers $Y[y'-1]$ (such an index exists by Lemma 6.9). Due to $y'-1 \in P_Y$, we have $\mathsf{CE}_{k'}(W_X)[w] = \mathsf{CE}_{k'}(W_Y)[w]$, and hence $x - y = x_w - y_w$. Definition of w further yields $\mathsf{CE}_{k'}(W_Y)[w+1\,..\,v) = \bot^{u-w-1}$, and thus $\pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,v)) = y_w$. Moreover, since $\pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,t)) = y_w$ for $t \in [w+1\,..\,v]$, we have $\pi(\mathsf{CE}_{k'}(W_X)[1\,..\,v)) = x_w$ by definition of v. This completes the proof that $x - y = \pi(\mathsf{CE}_{k'}(W_X)[1\,..\,v)) - \pi(\mathsf{CE}_{k'}(W_Y)[1\,..\,v))$.

To derive $\mathsf{E}^M(X, Y)$, it suffices to convert $\mathsf{D}(Y')$ to $\mathsf{D}(Y^M)$ (symmetrically), and to construct $\mathsf{D}(X^M Y^M)$ using Proposition 4.2(i). Since $\overline{\mathsf{maxLZ}}(C^2_{k'}(X, x_t)), \overline{\mathsf{maxLZ}}(C^2_{k'}(Y, y_t)) = \mathcal{O}(k)$ holds for all $t \in [1\,..\,3n+1]$ and since $|\mathsf{MP}_\mathsf{CE}| = \mathcal{O}(k \log n)$, the $\overline{\mathsf{maxLZ}}(\cdot)$ measure of all intermediate strings is $\tilde{\mathcal{O}}(k^2)$. Consequently, the $\tilde{\mathcal{O}}(k)$ applications of Proposition 4.2 cost $\tilde{\mathcal{O}}(k^3)$ time and use $\tilde{\mathcal{O}}(k^2)$ space. □

To complete the complexity analysis, observe that the decoding procedure of Theorem 6.19 uses $\mathcal{O}(k_{6.19} \log(n_{6.19}\sigma_{6.19}) \log^2(n_{6.19} \log \sigma_{6.19})) = \tilde{\mathcal{O}}(k^2)$ time and $\mathcal{O}(k_{6.19} \log(n_{6.19}\sigma_{6.19})) = \tilde{\mathcal{O}}(k^2)$ bits of space. The procedure of Proposition 6.20 uses $\mathcal{O}(k_{6.20} \log^2 n_{6.20}) = \tilde{\mathcal{O}}(k)$ bits of space and costs $\mathcal{O}(k_{6.20} \log^4 n_{6.20}) = \tilde{\mathcal{O}}(k)$ time. Finally, due to $|\mathsf{LZ}(X^M Y^M)| = \tilde{\mathcal{O}}(k^2)$ and $|\mathcal{B}_k(X, Y)| = \mathcal{O}(k^5)$ (Lemma 3.11), the algorithm of Proposition 5.12 uses $\tilde{\mathcal{O}}(k^2)$ space and costs $\tilde{\mathcal{O}}(k^5)$ time (dominating the overall decoding complexity). If we only aim to retrieve $\min(\mathsf{ed}(X, Y), k+1)$, the algorithm of Corollary 5.5 takes $\tilde{\mathcal{O}}(k^2)$ time and space (in which case the overall decoding uses $\tilde{\mathcal{O}}(k^2)$ space and costs $\tilde{\mathcal{O}}(k^3)$ time.

It remains to argue that the decoding algorithm is correct. With $\delta_{3.16} = \frac{\delta}{3}$ probability loss, we may assume that $\mathcal{A}_\mathsf{W} \in \mathsf{GA}_{c_{3.16}(\mathsf{ed}(X,Y))^2}(X, Y)$ and $\mathsf{width}(\mathcal{A}_\mathsf{W}) \leq c_{3.16}\mathsf{ed}(X, Y)$. With $\delta_{6.19} + \delta_{6.20} = \frac{2\delta}{3}$ further probability loss, we may assume that the decoders of Theorem 6.19 and Proposition 6.20 are successful. If $\mathsf{ed}(X, Y) > k$, then the correctness follows from Proposition 5.12 and Corollary 5.5 because $M \subseteq \mathcal{M}_{X,Y}(\mathcal{A}_\mathsf{W})$ is a non-crossing matching. Otherwise, Proposition 3.16 guarantees $\mathsf{cost}(\mathcal{A}_\mathsf{W}) \leq c_{3.16}k^2$ and $\mathsf{width}(\mathcal{A}_\mathsf{W}) \leq c_{3.16}k$ so, in particular, $k' \geq \mathsf{width}(\mathcal{A}_\mathsf{W}) + 5k$. By Lemma 6.12, we thus have $\mathsf{hd}(\mathsf{CE}_{k'}(W_X), \mathsf{CE}_{k'}(W_X)) \leq c_{6.12}(1 + c_{3.16})k \log(3n)$. Hence, the decoders of Theorem 6.19 and Proposition 6.20 report the mismatch information. Lemma 6.10 further implies that $\Delta_X(\mathcal{A}, \mathcal{A}_\mathsf{W}) \subseteq P_X$ and $\Delta_Y(\mathcal{A}, \mathcal{A}_\mathsf{W}) \subseteq P_Y$ for all $\mathcal{A} \in \mathsf{GA}_k(X, Y)$. In particular, $\mathcal{M}_{X,Y}(\mathcal{A}) \subseteq M$, and therefore $\mathcal{M}_k(X, Y) \subseteq M$. Conse-

quently, the algorithms of Proposition 5.12 and Corollary 5.5 correctly compute $\mathsf{GR}_k(X,Y)$ and $\min(\mathsf{ed}(X,Y),k+1)$, respectively. □

Next, we boost the success probability and strengthen the sketches so that we can retrieve $\mathsf{qGR}_k(X,Y)$ instead of $\mathsf{GR}_k(X,Y)$.

Corollary 6.23. *There is a sketch $\mathsf{sk}^{\mathsf{q}}_k$ (parametrized by $\delta \in (0,\frac{1}{2})$, integers $n \geq k \geq 1$, an alphabet $\Sigma = [0\,..\,n^{\mathcal{O}(1)})$, and a seed of $\mathcal{O}(\log^2 n \log(1/\delta))$ random bits) such that:*

(a) The sketch $\mathsf{sk}^{\mathsf{q}}_k(S)$ of a string $\Sigma^{\leq n}$ takes $\mathcal{O}(k^2\log^3 n\log(1/\delta))$ bits. Given streaming access to S, it can be constructed in $\tilde{\mathcal{O}}(nk\log(1/\delta))$ time using $\tilde{\mathcal{O}}(k^2\log(1/\delta))$ space.

(b) There exists a decoding algorithm that, given $\mathsf{sk}^{\mathsf{q}}_k(X),\mathsf{sk}^{\mathsf{q}}_k(Y)$ for $X,Y\in\Sigma^{\leq n}$, with probability at least $1-\delta$ computes $\mathsf{qGR}_k(X,Y)$. The algorithm takes $\tilde{\mathcal{O}}(k^5\log(1/\delta))$ time and uses $\tilde{\mathcal{O}}(k^2\log(1/\delta))$ space.

Proof. We shall use $\mu=\mathcal{O}(\log(1/\delta))$ sketches $\mathsf{sk}^{E}_{k_{3.17}}$ of Theorem 3.17 with $\delta_{3.17}=\frac{1}{3}$, $n_{3.17}=n+1$, $k_{3.17}=k+1$, and independent seeds. For each of the μ sketches $\mathsf{sk}^{E}_{k_{3.17}}$, the sketch $\mathsf{sk}^{\mathsf{q}}_k(S)$ contains $\mathsf{sk}^{E}_{k_{3.17}}(\$_1S)$, $\mathsf{sk}^{E}_{k_{3.17}}(\$_2S)$, where $\$_1,\$_2\notin\Sigma$ are two distinct symbols. This construction uses $\mathcal{O}(\mu\log^2 n_{3.17})=\mathcal{O}(\log^2 n\log(1/\delta))$ random bits and produces sketches of $\mathcal{O}(\mu k^2_{3.17}\log^3 n_{3.17})=\mathcal{O}(k^2\log^3 n\log(1/\delta))$ bits.

The encoding algorithm calls 2μ instances of the encoding algorithm of Theorem 3.17. Hence, it uses $\tilde{\mathcal{O}}(\mu k^2_{3.17})=\mathcal{O}(k^2\log(1/\delta))$ space and costs $\tilde{\mathcal{O}}(\mu n_{3.17}k_{3.17})=\tilde{\mathcal{O}}(nk\log(1/\delta))$ time.

The decoding algorithm, given $\mathsf{sk}^{\mathsf{q}}_k(X),\mathsf{sk}^{\mathsf{q}}_k(Y)$ for $X,Y\in\Sigma^{\leq n}$, runs the decoder of Theorem 3.17 for $\mathsf{sk}^{E}_{k_{3.17}}(\$_1X),\mathsf{sk}^{E}_{k_{3.17}}(\$_1Y)$ for each of the μ sketches $\mathsf{sk}^{E}_{k_{3.17}}$. This yields μ candidates for $\mathsf{GR}_{k_{3.17}}(\$_1X,\$_2Y)$, which we convert to candidates for $\mathsf{qGR}_k(X,Y)$ using Corollary 5.17. Finally, we determine the majority answer among the μ candidates. The equality test uses Proposition 4.2(i)(c) to compare two candidates for $\mathsf{D}(X^MY^M)$.

Recall that the decoding procedures of Theorem 3.17 use $\tilde{\mathcal{O}}(\mu k^5_{3.17})=\tilde{\mathcal{O}}(k^5\log(1/\delta))$ time and $\tilde{\mathcal{O}}(\mu k^2_{3.17})=\tilde{\mathcal{O}}(k^2\log(1/\delta))$ space. The applications of Corollary 5.17 and the equality tests take $\tilde{\mathcal{O}}(\mu k^2_{3.17})=\tilde{\mathcal{O}}(k^2\log(1/\delta))$ time and space. The entire decoding algorithm uses $\tilde{\mathcal{O}}(k^2\log(1/\delta))$ space and $\tilde{\mathcal{O}}(\mu k^5_{3.17})=\tilde{\mathcal{O}}(k^5\log(1/\delta))$ time.

As for correctness, since the μ sketches $\mathsf{sk}^{E}_{k_{3.17}}$ are independent, by the Chernoff bound, the majority answer is wrong with probability at most $\exp(-\mathcal{O}(\mu))$. Setting $\mu=\mathcal{O}(\log(1/\delta))$ (with a sufficiently large constant factor) guarantees a success probability of $1-\delta$. □

7 Pattern Matching with k Edits

In this section, we present solutions for pattern matching with k edits in the semi-streaming and streaming settings.

7.1 Periodicity under Edit Distance

We start by recalling combinatorial properties of strings periodic under the edit distance.

Definition 3.2 (k-periodic string)**.** *A string X is* k-periodic *if there exists a primitive string Q with $|Q|\leq|X|/128k$ such that the edit distance between X and a prefix of Q^∞ is at most $2k$. We call Q a* k-period *of X.*

Claim 7.1. *Suppose that a string X is a prefix of a string Y, where $|X|<|Y|\leq 2|X|$. If X is k-periodic with k-period Q, $|Q|\leq|X|/128k$, then either Y is not k-periodic, or Y is k-periodic with k-period Q.*

Proof. Suppose by contradiction that Y is k-periodic with k-period $Q' \neq Q$. Let $q = |Q|$ and $q' = |Q'|$. Assume first $q \leq q'$. Fix an alignment of the smallest cost between Y and a prefix of $(Q')^\infty$. It induces an alignment $\mathcal{A}'$ of cost at most $2k$ between X and a prefix of $(Q')^\infty$ and hence generates a partition $X = X_1 X_2 \dots X_z$, where each X_i, $1 \leq i \leq z-1$, is aligned with Q' and X_z is aligned with a prefix of Q'. From $|Q| \leq |X|/128k$ we obtain $|X| \geq 128k$ and therefore

$$z \geq (|X| - 2k)/q' \geq \frac{128-2}{128}|X|/(|Y|/128k) \gg 20k$$

Consider fragments $X_1X_2X_3X_4$, $X_5X_6X_7X_8$, and so on. The total number of such fragments is at least $(20k-3)/4 > 4k$, and at least $2k+1$ of them are not edited under $\mathcal{A}'$. Fix an optimal alignment $\mathcal{A}$ between X and a prefix Q^∞. The cost of $\mathcal{A}$ is bounded from above by $2k$, and therefore there is at least one fragment $X_{4i+1}X_{4i+2}X_{4i+3}X_{4i+4}$ that is not edited. Consider one such fragment F. On the one hand, $F = Q'Q'Q'Q'$. On the other hand, $F = \mathsf{suff}(Q)Q^j\mathsf{pref}(Q)$, where $\mathsf{suff}(Q)$ and $\mathsf{pref}(Q)$ are some suffix and some prefix of Q, respectively.

Suppose first that q' is a multiple of q. In this case, $Q' = (Q[\ell+1\,..\,q]Q[1\,..\,\ell])^r$, where $\ell = q - |\mathsf{suff}(Q)|$ and an integer r, which contradicts the fact that Q' is primitive. Otherwise, consider the copy of Q that contains $F[q']$. Consider also a substring QQQ of F formed by the copy of Q that contains $F[2q']$, and the preceding and succeeding copies of Q. We then obtain that there is an occurrence of Q in QQQ that is not aligned with any copy of Q (otherwise, q' is a multiple of q), and therefore Q is not primitive, a contradiction.

The case $q > q'$ can be treated analogously. □

Note that Claim 7.1 implies in particular that a string can have at most one k-period.

7.2 Semi-streaming Algorithm

We first present a deterministic algorithm for pattern matching with k edits in the semi-streaming setting.

7.2.1 Preprocessing Stage

Consider a set Π of $\mathcal{O}(\log m)$ prefixes P_i of P initialised to contain P itself as well as the prefixes of length 2^ℓ for all $0 \leq \ell \leq \lfloor \log |P| \rfloor$. Order the prefixes by lengths, and consider two consecutive prefixes P', P''. If P' is k-periodic with k-period Q' while P'' is not k-periodic, we add two more prefixes to Π. Namely, if ℓ be the maximum integer such that $P[1\,..\,\ell]$ is k-periodic with k-period Q', add to Π the prefixes $P[1\,..\,\ell]$ and $P[1\,..\,\ell+1]$. Note that $P[1\,..\,\ell+1]$ is not k-periodic by Claim 7.1.

Let $\Pi = \{P_1, P_2, \dots, P_z\}$ be the resulting set of prefixes. We assume that the patterns are ordered in the ascending order of their lengths. During the preprocessing step, for each i such that P_i is k-periodic, we compute its k-period Q_i. We use notation $\ell_i = P_i$ and $q_i = |Q_i|$ (if defined). Importantly, we do not have to store Q_i explicitly, we can simply memorize its endpoints in P_i which takes $\mathcal{O}(\log m)$ extra space in total. We also store, for each of the $\mathcal{O}(k)$ rotations D of Q_i that can be a difference of a chain of k-edit occurrences of P_i (Corollary 3.5), the encodings $\mathsf{qGR}_{32k}(D, D)$ and $\mathsf{qGR}_{30k}(P[\ell_{i-1}+1\,..\,\ell_i], D^\infty[1\,..\,\Delta_i])$, where $\Delta_i = \ell_i - \ell_{i-1} + k$.

7.2.2 Main Stage

The main stage of the algorithm starts after we have preprocessed the pattern. During the main stage, we receive the text as a stream, one character of a time. We exploit the following result:

Fact 7.2 (cf. [59]). *Given a read-only string X of length m and a streaming string Y. There is a dynamic programming algorithm that correctly identifies all prefixes Y' of Y within edit distance at most $k \leq m$ from X, as well as $\mathsf{ed}(Y', X)$ itself. The algorithm takes $\mathcal{O}(km)$ time and $\mathcal{O}(k)$ space besides the space required to store X.*

Chains of k-edit occurrences. During the main stage of the algorithm, we store the following information. Let r be the newly arrived position of the text T. For each $2 \leq i \leq z$, consider all k-edit occurrences of P_{i-1} in $T[r - \ell_i - k + 1 \,..\, r]$. We call such occurrences *active*. We denote the set of right endpoints of the active occurrences by $\mathsf{aOCC}_k^E(P_{i-1}, T)$. By Corollaries 3.3 and 3.5, $\mathsf{aOCC}_k^E(P_{i-1}, T)$ forms $\mathcal{O}(k^3)$ chains. For each chain $\mathcal{C}$, we store the following information:

1. The leftmost position lp and the size $|\mathcal{C}|$ of $\mathcal{C}$;
2. An integer $\mathsf{ed}(\mathcal{C})$ equal to the smallest edit distance from P_{i-1} to a suffix of $T[1 \,..\, r]$ for every endpoint $r \in \mathcal{C}$;
3. If $|\mathcal{C}| \geq 2$, we store the shift Δ of the difference $D = Q_{i-1}[1 + \Delta \,..\, q_{i-1}]Q_{i-1}[1 \,..\, q_{i-1}]$ of $\mathcal{C}$.

If p^* is the first position added to $\mathcal{C}$ (it can be different from lp as we will update chains to contain only active occurrences), then at the position $(p^* + 1)$ we start running the dynamic programming algorithm for $T[p^* + 1 \,..]$ and $P[\ell_{i-1} + 1 \,..\, \ell_i]$ (Fact 7.2).

Furthermore, consider the moment when we detect the second position in $\mathcal{C}$ (if it exists) and hence the difference D of the chain. Starting from this moment, for every newly added position $p \in \mathcal{C}$, at the position $(p + 1)$ we start computing the greedy encoding $\mathsf{qGR}_{32k}(T[p + 1 \,..\, p + \Delta_i], D^\infty[1 \,..\, \Delta_i])$. We continue running the algorithm until either the computation is over or a new position in the chain is detected. In the end, we compute the encoding for the rightmost position in the chain.

Detecting new k-edit occurrences of P_i. We now explain how to detect new k-edit occurrences of the prefixes P_i. Let r be the latest arrived position of T. If $i = 1$, then since $k \geq 1$, $r \in \mathsf{aOCC}_k^E(P_1, T)$. Below we consider three possible cases for $i \geq 2$: P_{i-1} is k-periodic, P_i is not k-periodic; P_{i-1} is not k-periodic; P_{i-1} and P_i are k-periodic.

Case 1: P_{i-1} is k-periodic, P_i is not k-periodic. By construction, in this case $\ell_{i-1}+1 = \ell_i$. The position $r \in \mathsf{aOCC}_k^E(P_i, T)$ iff one of the following conditions is satisfied:

1. The smallest edit distance from P_{i-1} to a suffix of $T[1 \,..\, r]$ is at most $k-1$ (this corresponds to the case when the last character of P_i is deleted in an optimal alignment of a suffix of $T[1 \,..\, r]$ and P_i);
2. The smallest edit distance from P_{i-1} to a suffix of $T[1 \,..\, r-1]$ is at most $k-1$ and $P_i[\ell_i] \neq T[r]$ (this corresponds to the case when the last character of P_i is substituted for $T[r]$ in an optimal alignment of a suffix of $T[1 \,..\, r]$ and P_i);
3. The smallest edit distance from P_{i-1} to a suffix of $T[1 \,..\, r-1]$ is at most k and $P_i[\ell_i] = T[r]$ (this corresponds to the case when the last character of P_i is matched with $T[r]$ in an optimal alignment of a suffix of $T[1 \,..\, r]$ and P_i);
4. The smallest edit distance from P_i to a suffix of $T[1 \,..\, r-1]$ is at most $k-1$ (this corresponds to the case when $T[r]$ is deleted in an optimal alignment of a suffix of $T[1 \,..\, r]$ and P_i).

We can decide which of the conditions is satisfied, and therefore whether $r \in \mathsf{aOCC}_k^E(P_i, T)$ in $\mathcal{O}(k^3)$ time using $\mathsf{aOCC}_k^E(P_{i-1}, T)$ and $\mathsf{aOCC}_k^E(P_i, T)$. Moreover, we can compute the smallest edit distance from P_i to a suffix of $T[1 \,..\, r]$ if it is bounded by k.

For the next two cases, we will use the following simple observation that follows from Fact 2.3:

Observation 7.3. *Let* $\mathsf{ed}_{i-1}(r')$ *be the smallest edit distance from* P_{i-1} *to a suffix of* $T[1\,..\,r']$*, and define*

$$d = \min_{\substack{r' \in \mathsf{aOCC}_k^E(P_{i-1},T) \\ r' \in [r+1-\Delta_i, r+1-\Delta_i+2k]}} \{\mathsf{ed}_{i-1}(r') + \mathsf{ed}(P[\ell_{i-1}+1\,..\,\ell_i], T[r'+1\,..\,r])\} \tag{2}$$

The smallest edit distance from P_i *to* $T[1\,..\,r]$ *is equal to* d *if* $d \le k$ *and is larger than* k *otherwise.*

It follows that to decide whether $r \in \mathsf{aOCC}_k^E(P_i, T)$, it suffices to compute the value $\min\{d, k+1\}$, where d is as defined above.

Case 2: P_{i-1} **is not** k**-periodic.** In this case, $\mathsf{aOCC}_k^E(P_{i-1}, T)$ is stored as $\mathcal{O}(k^3)$ chains of size one. Therefore, we can find the positions r' from Eq. (2) in $\mathcal{O}(k^3)$ time. Moreover, for each position r', we run the dynamic programming algorithm for $T[r'+1\,..]$ and $P[\ell_{i-1}+1\,..\,\ell_i]$, which outputs the edit distance between $T[r'+1\,..\,r]$ and $P[\ell_{i-1}+1\,..\,\ell_i]$ if it is at most k. As we also know the smallest edit distance between P_{i-1} and a suffix of $T[1\,..\,r']$, we can compute d in $\mathcal{O}(k^3)$ time.

Case 3: P_{i-1} **and** P_i **are** k**-periodic.** We can identify all positions r' from Eq. (2) in $\mathcal{O}(k^3)$ time. (It suffices to check each of the $\mathcal{O}(k^3)$ chains that we store for P_{i-1}). We must now test each of these positions. Consider a position r' and let $\mathcal{C}$ be the chain containing it. It suffices to compute the edit distance between $P[\ell_{i-1}+1\,..\,\ell_i]$ and $T[r'+1\,..\,r]$ as we already know the smallest edit distance from P_{i-1} to a suffix of $T[1\,..\,r']$. If $|\mathcal{C}| = 1$, the distance has been computed by the dynamic programming algorithm. Otherwise, we use quasi-greedy encodings. On a high level, our goal is to compute the edit distance between $\pi = P[\ell_{i-1}\,..\,\ell_i]$ and $\tau = T[r'+1\,..\,r]$ via a string $\mu = D^\infty[1\,..\,\Delta_i]$, where D is the difference of $\mathcal{C}$ and $\Delta_i = \ell_i - \ell_{i-1} + k$.

Lemma 7.4. *There is* $\mathsf{ed}(\pi, \mu) \le 26k$.

Proof. As P_{i-1} and P_i are k-periodic, by Claim 7.1 we obtain that $P_i = P[1\,..\,\ell_i]$ is k-periodic with k-period $Q_i = Q_{i-1}$, that is, there is a prefix of Q_i^∞ such that the edit distance between it and P_i is at most $2k$. By Fact 2.3, there is a substring $Q_i^\infty[r\,..\,t]$ such that $|r - \ell_{i-1}| \le 2k$ and $|t - \ell_i| \le 2k$ and $\mathsf{ed}(Q_i^\infty[r\,..\,t], \pi) \le 2k$. By the triangle inequality, we obtain that $\mathsf{ed}(Q_i^\infty[\ell_{i-1}+1\,..\,\ell_i], \pi) \le 6k$. Let $a = \ell_{i-1} - 7k \pmod{q_i}$ and $b = \ell_{i-1} + 7k \pmod{q_i}$. By Corollary 3.5, D is a rotation of $Q_{i-1} = Q_i$ with shift Δ, where $\Delta \in [a-3k, b+3k]$ if $a \le b$ and $\Delta \in [0, b+3k] \cup [a-3k, q_i)$ if $a > b$. It follows that $D^\infty = Q_i^\infty[s\,..]$, where $|s - \ell_{i-1}| \le 10k$. As $\mu = D^\infty[1\,..\,\Delta_i] = Q_i^\infty[s\,..\,s+\Delta_i-1]$, by the triangle inequality we obtain $\mathsf{ed}(\mu, Q_i^\infty[\ell_{i-1}+1\,..\,\ell_i]) \le 20k$. Applying the triangle inequality one more time, we obtain the claim. □

Let $\mathcal{G}_P = \mathsf{qGR}_{30k}(\pi, \mu)$ and $\mathcal{G}_T = \mathsf{qGR}_{30k}(\mu, \tau)$. By Corollary 5.27, knowing $\mathcal{G}_P$ and $\mathcal{G}_T$ is sufficient to compute the edit distance between π and τ. Note that we do not know $\mathcal{G}_T$ yet, we must compute it using the available information. Let p be the rightmost position in $\mathcal{C}$.

1. Recall that at the position $(p+1)$ we launched an algorithm that is computing $\mathsf{qGR}_{32k}(T[p+1\,..\,p+\Delta_i], D^\infty[1\,..\,\Delta_i])$ with a delay of k characters (Corollary 5.22). We have $p + \Delta_i \ge r' + \Delta_i \ge r$. Therefore, upon reaching r, we can use the memory of the algorithm to compute $\mathcal{G}_{T,1} = \mathsf{qGR}_{32k}(T[p+1\,..\,r], D^\infty[1\,..\,r-p])$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space (Lemma 5.21).
2. By the definition of chains, $T[r'+1\,..\,p] = D^j$ for some integer j. By Lemma 5.21, we can use $\mathsf{qGR}_{32k}(D, D)$ computed during the preprocessing step to compute $\mathcal{G}_{T,2} = \mathsf{qGR}_{32k}(T[r'+1\,..\,p], D^j)$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space. By applying Lemma 5.21 again, we can compute $\mathsf{qGR}_{32k}(T[r'+1\,..\,p], D^j D^\infty[1\,..\,r-p])$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.

3. Finally, we compute via Corollary 5.22 the encoding $\mathsf{qGR}_{30k}(\varepsilon, D^\infty[r-p+1\,..\,\Delta_i])$, where ε is the empty string and $\Delta_i - (r-p) \leq 2k$. We then apply Observation 5.20 to compute $\mathsf{qGR}_{30k+(\Delta_i-(r-p))}(T[r'+1\,..\,p], D^j D^\infty[1\,..\,r-p])$ and further Lemma 5.21 to compute $\mathsf{qGR}_{30k}(T[r'+1\,..\,p], D^\infty[1\,..\,\Delta_i]) = \mathcal{G}_T$.

Updating the chains. When we detect a new k-edit occurrence of P_i, we must decide if it should be added to some existing chain or if we must create a new chain for this occurrence.

To this end, for each $1 \leq i \leq z$, we consider $\mathcal{O}(k)$ of its rotations of Q_i that can be the difference of a chain of k-edit occurrences of P_i in T (Corollary 3.5). For each rotation R, we run a constant-space and linear-time deterministic pattern matching algorithm [54]. The algorithm processes the text T as a stream and if there is an occurrence $T[\ell\,..\,r]$ of the rotation reports it while reading $T[r]$. The algorithm uses $\mathcal{O}(1)$ space and $\mathcal{O}(1)$ amortised time per character of T.

Suppose that we detect a new right endpoint r of a k-edit occurrence $T[\ell\,..\,r]$ of P_i. We must decide whether r belongs to an existing chain of k-edit occurrences of P_i or starts a new one. In order to do this, we first find the chain $\mathcal{C}$ that contains $r - q_i + 1$ if it exists by checking each chain in turn. We then check that the smallest edit distance from a suffix of $T[1\,..\,r]$ to P_i equals to $\mathsf{ed}(\mathcal{C})$ and that $T[r-q_i+1\,..\,r]$ is equal to the difference of the chain. (Recall that we run the exact pattern matching algorithm for each rotation of Q_i that can be the difference of a chain so that both checks can be performed in $\mathcal{O}(1)$ time).

If these conditions are not satisfied, we create a new chain that contains r only. Otherwise, we add r to $\mathcal{C}$ (i.e., increment the size of $\mathcal{C}$). To finalize the update of the chains, we must delete all k-edit occurrences that become inactive: for each i and for each chain of k-edit occurrences of P_i, we "delete" the first k-edit occurrence if it starts before $r - \ell_i - k + 1$. To "delete" a k-edit occurrence, we simply update the endpoints of the first occurrence in the chain and the total number of occurrences in the chain.

7.2.3 Analysis

We summarize the results of this section:

Theorem 7.5. *Assume a read-only pattern P of length m and a streaming text T of length n. There is a deterministic algorithm that finds the set $\mathsf{OCC}_k(P,T)$ using $\tilde{\mathcal{O}}(k^5)$ space and $\tilde{\mathcal{O}}(k^6)$ amortised time per character of the text T.*

Proof. Let us first upper bound the space complexity of the algorithm. For each $i = 1, \ldots, z = \mathcal{O}(\log m)$, we store the set $\mathsf{aOCC}_k(P_i, T)$ as $\mathcal{O}(k^3)$ chains. For each chain, we launch the dynamic programming algorithm (Fact 7.2), which takes $\tilde{\mathcal{O}}(k^2)$ space and Corollary 5.22 that takes $\tilde{\mathcal{O}}(k^2)$ space. The pattern matching algorithms for the rotations of Q_i take $\tilde{\mathcal{O}}(k)$ space in total. Finally, testing if a position of the text is the rightmost position of a k-edit occurrence of P_i requires $\tilde{\mathcal{O}}(k^2)$ space.

We now show the time bound. Updating the chains takes $\tilde{\mathcal{O}}(1)$ time. At any time, we run $\tilde{\mathcal{O}}(k^3)$ instances of Corollary 5.22 that takes $\mathcal{O}(k^3)$ amortised time per character. To test each position r, we spend $\tilde{\mathcal{O}}(k \cdot k^3)$ time. □

7.3 Streaming Algorithm

We now modify the algorithm for the semi-streaming model to develop a fully streaming algorithm. W.l.o.g., assume $k \leq m$ and take $\delta = 1/n^c$ for c large enough.

7.3.1 Preprocessing

We define the prefixes P_i and their periods Q_i exactly in the same way as in Section 7.2. Recall that $\ell_i = |P_i|$ and $q_i = |Q_i|$. For every $i > 1$, we store the following information, where all sketches sk^{q} (Corollary 6.23) are parametrized by probability δ, maximal length Δ_i, the alphabet of P and T, and a seed of $\mathcal{O}(\log^2 n \log(1/\delta))$ random bits:

1. $P[\ell_i]$ and the sketch $\mathsf{sk}^{\mathsf{q}}_k(P[\ell_{i-1}\,..\,\ell_i])$;
2. For each of rotation D of Q_i that can be a difference of a chain of k-edit occurrences of P_i, the encoding $\mathsf{qGR}_{30k}(P[\ell_{i-1}+1\,..\,\ell_i], D^\infty[1\,..\,\Delta_i])$, where $\Delta_i = \ell_i - \ell_{i-1} + k$;
3. For each of rotation D of Q_i that can be a difference of a chain of k-edit occurrences of P_i, the sketches $\mathsf{sk}^{\mathsf{q}}_{32k}(D)$ and $\mathsf{sk}^{\mathsf{q}}_{32k}(D[1\,..\,\Delta_i \pmod{q_i}])$.

7.3.2 Main Stage

As in Section 7.2, for each i, we store $\mathsf{aOCC}^E_k(P_{i-1}, T)$ in $\mathcal{O}(k^3)$ chains. For each chain $\mathcal{C}$, we store the leftmost position lp in it, its size $|\mathcal{C}|$, and the smallest edit distance, $\mathsf{ed}(\mathcal{C})$, from a suffix of $T[1\,..\,lp]$ to P_{i-1}. If $|\mathcal{C}| \geq 2$, we also store its difference (defined by the shift of the rotation of Q_{i-1}).

If p^* is the first position added to $\mathcal{C}$, then at the position (p^*+1), we start running the streaming algorithm of Corollary 6.23(a) for computing the sketch $\mathsf{sk}^{\mathsf{q}}_k(T[p^a st+1\,..\,p^a st+\Delta_i])$.

Furthermore, consider the moment when we detect the second position in $\mathcal{C}$ (if it exists) and hence the difference D of the chain. Starting from this moment, for every newly added position $p \in \mathcal{C}$, at the position $(p+1)$ we start computing the quasi-greedy encoding $\mathsf{qGR}_{32k}(T[p+1\,..\,p+\Delta_i], D^\infty[1\,..\,\Delta_i])$ as follows: Assume that we have computed $\mathsf{qGR}_{32k}(T[p+1\,..\,p+\ell\cdot q_i], D^\infty[1\,..\,\ell\cdot q_i])$. Suppose first that $(\ell+1)\cdot q_i \leq \Delta_i$. While reading $T[p+\ell\cdot q_i+1\,..\,p+(\ell+1)\cdot q_i]$, we compute $\mathsf{sk}^{\mathsf{q}}_{32k}(T[p+\ell\cdot q_i+1\,..\,p+(\ell+1)\cdot q_i])$ again via the algorithm of Corollary 6.23(a). We then use this sketch and $\mathsf{sk}^{\mathsf{q}}_{32k+1}(D)$ to compute $\mathsf{qGR}_{32k}(T[p+\ell\cdot q_i+1\,..\,p+(\ell+1)\cdot q_i], D)$ (Corollary 6.23(b)) and then $\mathsf{qGR}_{32k}(T[p+1\,..\,p+(\ell+1)\cdot q_i], D^\infty[1\,..\,(\ell+1)\cdot q_i])$ (Lemma 5.21). If $(\ell+1)\cdot q_i > \Delta_i$, we use the sketches $\mathsf{sk}^{\mathsf{q}}_{32k}(T[p+\ell\cdot q_i\,..\,p+\Delta_i])$ and $\mathsf{sk}^{\mathsf{q}}_{32k}(D[1\,..\,\Delta_i \pmod{q_i}])$, the rest is analogous. We continue running the algorithm until either the computation is completed or a new k-edit occurrence in the chain has been detected. In other words, in the end we compute the encoding for the rightmost position in the chain.

Detecting new k-edit occurrences of P_i. We now explain how we modify the algorithm for detecting new k-edit occurrences of the prefixes P_i. The algorithm for Case 1 does not change. Instead of the dynamic programming algorithm in Case 2, we use $\mathsf{sk}^{\mathsf{q}}_k$ and use Corollary 6.23(b) and then Corollary 5.27 to compute the edit distance. It remains to explain how we modify the algorithm for Case 3.

We exploit Eq. (2) again. We can find all positions r' in $\mathcal{O}(k^3)$ time. To test a position r', it suffices to compute the edit distance between $P[\ell_{i-1}+1\,..\,\ell_i]$ and $T[r'+1\,..\,r]$. Let $\mathcal{C}$ be the chain with difference D that contains r'. If $|\mathcal{C}| = 1$, we use the edit distance sketch, and otherwise the quasi-greedy encodings. By Lemma 7.4, $\mathsf{ed}(P[\ell_{i-1}\,..\,\ell_i], D^\infty[1\,..\,\Delta_i]) \leq 26k$ and hence by Corollary 5.27, the edit distance between $P[\ell_{i-1}+1\,..\,\ell_i]$ and $T[r'+1\,..\,r]$ can be computed from the encodings $\mathcal{G}_P = \mathsf{qGR}_{30k}(\pi, \mu)$ and $\mathcal{G}_T = \mathsf{qGR}_{30k}(\mu, \tau)$ for $\pi = P[\ell_{i-1}\,..\,\ell_i]$, $\mu = D^\infty[1\,..\,\Delta_i]$, and $\tau = T[r'+1\,..\,r]$. $\mathcal{G}_P$ was computed during the preprocessing step and we store it explicitly. Hence, we only need to explain how to compute $\mathcal{G}_T$. Let p be the rightmost position in the chain $\mathcal{C}$.

1. Recall that $T[r'+1\,..\,p] = D^j$ for $j = (p-r')/q_i$. We first compute $\mathsf{qGR}_{32k}(D, D)$ from $\mathsf{sk}^{\mathsf{q}}_{32k}(D, D)$ via Corollary 6.23(b), and then $\mathcal{G}_{T,1} = \mathsf{qGR}_{32k}(T[r'+1\,..\,p], D^j) =$

$\mathsf{qGR}_{32k}(D^j, D^j)$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space as in Section 7.2.

2. At the position $(p+1)$ we launched the streaming algorithm computing $\mathsf{qGR}_{32k}(T[p+1\,..\,p+\Delta_i], D^\infty[1\,..\,\Delta_i])$ with a delay of q_i characters. We have $p+\Delta_i \geq r'+\Delta_i \geq r$. Therefore, at a position $p+\ell\cdot q_i$, where $\ell = \lfloor (r-p+1)/q_i \rfloor$, the algorithm computes $\mathcal{G}_{T,2} = \mathsf{qGR}_{32k}(T[p+1\,..\,p+\ell\cdot q_i], D^\infty[1\,..\,\ell\cdot q_i])$. The algorithm then continues to compute the sketch $\mathsf{sk}^{\mathsf{q}}_{32k}(T[p+\ell\cdot q_i+1\,..\,r])$. We use this sketch and the sketch $\mathsf{sk}^{\mathsf{q}}_{32k}(D[1\,..\,\Delta_i \pmod{q_i}])$ to compute $\mathcal{G}_{T,3} = \mathsf{qGR}_{30k}(T[p+\ell\cdot q_i+1\,..\,r], D[1\,..\,\Delta_i \pmod{q_i}])$ via Corollary 6.23(b) and Observation 5.20.
3. We finally concatenate $\mathcal{G}_{T,1}$ and $\mathcal{G}_{T,2}$ to obtain $\mathsf{qGR}_{32k}(T[r'+1\,..\,r'+(\ell+j)\cdot q_i], D^\infty[1\,..\,(\ell+j)\cdot q_i])$, and then $\mathsf{qGR}_{32k}(T[r'+1\,..\,r'+(\ell+j)\cdot q_i], D^\infty[1\,..\,(\ell+j)\cdot q_i])$ and $\mathcal{G}_{T,3}$ to obtain $\mathcal{G}_T$ via Lemma 5.21 (note that the difference of lengths of strings in $\mathcal{G}_{T,3}$ is bounded by $2k$).

Updating the chains. When we detect a new k-edit occurrence of P_i, we must decide if it should be added to some existing chain or if we must create a new chain for this occurrence. We use the algorithm of Section 7.2, but replace the constant-space pattern matching algorithm with the streaming pattern matching algorithm [52] that for a rotation of Q_i takes $\tilde{\mathcal{O}}(1)$ space and $\tilde{\mathcal{O}}(1)$ time per character and retrieves all its occurrences correctly with probability at least $1-\delta$.

7.3.3 Analysis

We summarize the results of this section:

Theorem 7.6. *Given a pattern P of length m and a text T of length n. There is a streaming algorithm that finds the set $\mathsf{OCC}^E_k(P,T)$ using $\tilde{\mathcal{O}}(k^5)$ space and $\tilde{\mathcal{O}}(k^8)$ amortised time per character of the text T. The algorithm computes $\mathsf{OCC}^E_k(P,T)$ correctly with high probability.*

Proof. By Corollary 3.5, for a fixed i only $\mathcal{O}(k)$ rotations of Q_i can be a difference of a chain of occurrences of P_i. The sketch $\mathsf{sk}^{\mathsf{q}}_{30k+1}(\cdot)$ takes $\tilde{\mathcal{O}}(k^2)$ space (Corollary 6.23(a)) and $\mathsf{qGR}_{30k}(\cdot,\cdot)$ $\tilde{\mathcal{O}}(k^2)$ space as well (Corollary 5.17). Therefore, the information computed during the preprocessing stage occupies $\tilde{\mathcal{O}}(k^3)$ space. During the main stage, we store $\tilde{\mathcal{O}}(k^3)$ chains. For each chain, we run the algorithm of Corollary 6.23(a) that takes $\tilde{\mathcal{O}}(k^2)$ space. The algorithm than computes the quasi-greedy encoding (Corollary 6.23(b)) takes $\tilde{\mathcal{O}}(k^2)$ space as well. In total, the information we store for the chains occupies $\tilde{\mathcal{O}}(k^5)$ space. When checking for new occurrences, we apply Lemma 5.21 and Corollary 5.27, which require an overhead of $\tilde{\mathcal{O}}(k^2)$ space. Finally, the streaming pattern matching algorithms for the rotations of Q_i that can be differences of occurrences of P_i take $\tilde{\mathcal{O}}(k)$ space in total. The space bound follows.

At any time, we run $\tilde{\mathcal{O}}(k^3)$ instances of the algorithm of Corollary 6.23(a) that takes $\tilde{\mathcal{O}}(k)$ amortised time per character. In addition, for every character we run $\tilde{\mathcal{O}}(k^3)$ instances of the algorithms of Lemma 5.21 and Corollary 5.27 taking $\tilde{\mathcal{O}}(k^8)$ time in total. The pattern matching algorithms for the rotations of Q_i take $\tilde{\mathcal{O}}(k)$ time per character.

Note that the only probabilistic procedures in the algorithm are streaming pattern matching [52] and that of Corollary 6.23(b) that computes quasi-greedy encodings. These procedures are called $\mathrm{poly}(n,k) = \mathrm{poly}(n)$ times. By choosing the constant c in $\delta = 1/n^c$ large enough, we can guarantee that the algorithm is correct with high probability by the union bound. □

References

[1] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: Or: A polylog shaved is a lower bound made. In *STOC 2016*, pages 375–388, 2016.

[2] Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi:10.1137/0216067.

[3] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004. URL: https://doi.org/10.1016/S0196-6774(03)00097-X, doi:10.1016/S0196-6774(03)00097-X.

[4] Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theor. Comput. Sci.*, 494:13–23, 2013.

[5] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC 2015*, pages 51–58, 2015. doi:10.1145/2746539.2746612.

[6] Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *FOCS 2016*, pages 51–60, 2016. doi:10.1109/FOCS.2016.15.

[7] Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.

[8] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS 2015*, pages 79–97, 2015. doi:10.1109/FOCS.2015.15.

[9] Diptarka Chakraborty, Debarati Das, and Michal Koucký. Approximate online pattern matching in sublinear time. In *FSTTCS 2019*, volume 150 of *LIPIcs*, pages 10:1–10:15, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.10.

[10] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *STOC 2016*, pages 712–725, 2016. doi:10.1145/2897518.2897577.

[11] Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *STOC 2020*, pages 643–656, 2020. URL: https://doi.org/10.1145/3357713.3384266, doi:10.1145/3357713.3384266.

[12] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *FOCS 2020*, pages 978–989, 2020. doi:10.1109/FOCS46700.2020.00095.

[13] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. Dictionary matching in a stream. In *ESA 2015*, volume 9294 of *LNCS*, pages 361–372, 2015. doi:10.1007/978-3-662-48350-3_31.

[14] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *SODA 2016*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142.

[15] Raphaël Clifford, Markus Jalsenius, and Benjamin Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In *SODA 2015*, pages 552–561, 2015. doi:10.1137/1.9781611973730.37.

[16] Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem, 2019. doi:10.1137/1.9781611975482.68.

[17] Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM J. Comput.*, 31(6):1761–1782, 2002. doi:10.1137/S0097539700370527.

[18] Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming periodicity with mismatches. In *APPROX 2017*, pages 42:1–42:21, 2017.

[19] Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Periodicity in data streams with wildcards. In *CSR 2018*, pages 90–105, 2018. doi:10.1007/978-3-319-90530-3_9.

[20] Funda Ergün, Hossein Jowhari, and Mert Sağlam. Periodicity in streams. In *APPROX 2010*, pages 545–559, 2010. doi:10.1007/978-3-642-15369-3_41.

[21] N. François, F. Magniez, M. de Rougemont, and O. Serre. Streaming property testing of visibly pushdown languages. In *ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17, 2016.

[22] Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *STACS 2018*, volume 96 of *LIPIcs*, pages 31:1–31:14, 2018. doi:10.4230/LIPIcs.STACS.2018.31.

[23] Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.18.

[24] Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 127:1–127:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.127.

[25] Moses Ganardi, Danny Hucke, and Markus Lohrey. Sliding window algorithms for regular languages. In *LATA 2018*, volume 10792, pages 26–35, 2018. doi:10.1007/978-3-319-77313-1_2.

[26] Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. In *ISAAC 2019*, volume 149 of *LIPIcs*, pages 6:1–6:13, 2019. doi:10.4230/LIPIcs.ISAAC.2019.6.

[27] Moses Ganardi, Artur Jeż, and Markus Lohrey. Sliding windows over context-free languages. In *MFCS 2018*, volume 117 of *LIPIcs*, pages 15:1–15:15, 2018. doi:10.4230/LIPIcs.MFCS.2018.15.

[28] Paweł Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemyslaw Uznański. Tight tradeoffs for real-time approximation of longest palindromes in streams. *Algorithmica*, 81(9):3630–3654, 2019. doi:10.1007/s00453-019-00591-8.

[29] Paweł Gawrychowski, Jakub Radoszewski, and Tatiana Starikovskaya. Quasi-periodicity in streams. In *CPM*, volume 128 of *LIPIcs*, pages 22:1–22:14, 2019. doi:10.4230/LIPIcs.CPM.2019.22.

[30] Pawel Gawrychowski and Tatiana Starikovskaya. Streaming dictionary matching with mismatches. In *CPM 2019*, volume 128 of *LIPIcs*, pages 21:1–21:15, 2019. doi:10.4230/LIPIcs.CPM.2019.21.

[31] Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62.

[32] Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. The streaming k-mismatch problem: Tradeoffs between space and total time. In *CPM 2020*, volume 161 of *LIPIcs*, pages 15:1–15:15, 2020. doi:10.4230/LIPIcs.CPM.2020.15.

[33] Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 65:1–65:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

[34] Shay Golan and Ely Porat. Real-time streaming multi-pattern search for constant alphabet. In *ESA 2017*, volume 107 of *LIPIcs*, pages 41:1–41:15, 2017. doi:10.4230/LIPIcs.ESA.2017.41.

[35] Tomohiro I. Longest common extensions with recompression. In *CPM 2017*, volume 78 of *LIPIcs*, pages 18:1–18:15, 2017.

[36] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

[37] Ce Jin, Jelani Nelson, and Kewen Wu. An improved sketching bound for edit distance, 2021. doi:10.4230/LIPIcs.STACS.2021.45.

[38] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. of R&D*, 31(2):249–260, 1987.

[39] Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture, 2020. doi:10.1109/FOCS46700.2020.00097.

[40] S. Kosaraju. Efficient string matching. 1987.

[41] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557582, April 1998.

[42] Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7.

[43] Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2.

[44] David Levin and Yuval Peres. *Markov Chains and Mixing Times*. American Mathematical Society, 2017. doi:10.1090/mbk/107.

[45] Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM J. Comput.*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

[46] 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. doi:https://doi.org/10.1016/0022-0000(80)90002-1.

[47] O. Merkurev and A. M. Shur. Searching runs in streams. In *SPIRE 2019*, volume 11811 of *LNCS*, pages 203–220, 2019.

[48] Oleg Merkurev and Arseny M. Shur. Searching long repeats in streams. In *CPM 2019*, volume 128 of *LIPIcs*, pages 31:1–31:14, 2019.

[49] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001. doi:10.1145/375360.375365.

[50] Noam Nisan. Pseudorandom generators for space-bounded computation. *Comb.*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

[51] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Discrete Applied Mathematics*, 274:116–129, 2020. Stringology Algorithms. doi:https://doi.org/10.1016/j.dam.2019.01.014.

[52] Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *FOCS 2009*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.

[53] Jakub Radoszewski and Tatiana Starikovskaya. Streaming k-mismatch with error correcting and applications. *Journal of Information and Computation*, 271:104513, 2020. doi:10.1016/j.ic.2019.104513.

[54] Wojciech Rytter. On maximal suffixes and constant-space linear-time versions of KMP algorithm. *Theoretical Computer Science*, 299(1):763 – 774, 2003. doi:https://doi.org/10.1016/S0304-3975(02)00590-X.

[55] Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *FOCS 1996*, pages 320–328, 1996. doi:10.1109/SFCS.1996.548491.

[56] Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359 – 373, 1980. doi:https://doi.org/10.1016/0196-6774(80)90016-4.

[57] Tatiana Starikovskaya. Communication and streaming complexity of approximate pattern matching. In *CPM 2017*, volume 78 of *LIPIcs*, pages 13:1–13:11, 2017. doi:10.4230/LIPIcs.CPM.2017.13.

[58] Alexandre Tiskin. Semi-local string comparison: algorithmic techniques and applications, 2013. arXiv:0707.3619v21.

[59] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985. doi:10.1016/S0019-9958(85)80046-2.