Small-space and streaming pattern matching with k edits

Tomasz Kociumaka
University of California,
Berkeley, USA
kociumaka@berkeley.edu

Ely Porat
Bar-Ilan University,
Ramat Gan, Israel
porately@cs.biu.ac.il

Tatiana Starikovskaya
DI/ENS, PSL Research University,
Paris, France
tat.starikovskaya@gmail.com

2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS) | 978-1-6654-2055-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/FOCS52979.2021.00090

***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[1] and $\tilde{\mathcal{O}}(k^8)$ amortized time per character of the text, providing answers correct with high probability. 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 at most k with a certain property (we call such alignments *greedy*). On strings of length at most n, the encoding occupies $\mathcal{O}(k^2)$ 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 parameterized 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 Zhang [FOCS 2016] and very recent $\tilde{\mathcal{O}}(k^3)$-size sketches of Jin, Nelson, and Wu [STACS 2021].

***Index Terms*—pattern matching; edit distance; streaming**

I. 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 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 the *approximate pattern matching problem*, can be formalized 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 [38]; 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 threshold k; a variant known as the *k-mismatch problem*. The first algorithm for the k-mismatch problem was given by Landau and Vishkin [39], who achieved the running time of $\mathcal{O}(nk)$ via so-called "kangaroo jumps", a technique utilizing 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, with running times $\mathcal{O}(n\sqrt{k \log k})$ and $\tilde{\mathcal{O}}(n + k^3 n/m)$, respectively. Continuing this line of research, Clifford et al. [14] presented an $\tilde{\mathcal{O}}(n + k^2 n/m)$-time algorithm, whereas Gawrychowski and Uznański [30] 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 $\mathcal{O}(n + kn\sqrt{\log m}/\sqrt{m})$ at the cost of Monte-Carlo randomization.

For the edit distance, a detailed survey of previous solutions

A full version of this paper is available at arxiv.org/abs/2106.06037.

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

2575-8454/21/$31.00 ©2021 IEEE
DOI 10.1109/FOCS52979.2021.00090

can be found in [45], 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 [51]. The algorithm is based on dynamic programming and uses $\mathcal{O}(nm)$ time. Masek and Paterson [42] 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 [35] 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 $\operatorname{poly}\log n$ 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 word size $w = 1$, any randomized algorithm that computes the edit distances between the pattern and the text online must spend $\Omega(\sqrt{\log n/(\log\log n)^3})$ expected amortized 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 [40]; this by-now classical algorithm solves the problem in $\mathcal{O}(nk)$ time. This remains the state of the art for large k, whereas the currently best result for small k was achieved by a series of work [50], [17] with the running time of $\mathcal{O}(n + nk^4/m)$. Very recently, Charalampopoulos et al. [12] studied the problem for both distances in the grammar-compressed and dynamic settings. Their result, in particular, implies an $\mathcal{O}(k^4)$-space and $\mathcal{O}(nk^4)$-time algorithm for the read-only model, where read-only random access to characters in P and T is allowed and one accounts only for the working space, that is, 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 [47], 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], [52], [33], [31], [16], [32], [48], search of repetitions in streams [20], [18], [19], [28], [43], [44], [29], and recognizing formal languages in streams [41], [24], [25], [22], [23], [27], [4], [21], [26].

All known streaming algorithms for approximate pattern matching under the Hamming distance [47], [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 [37], which allows computing the Hamming distance between two strings or certify that it exceeds k. The current best algorithm, by Clifford et al. [16], uses $\mathcal{O}(k\log\frac{m}{k})$ 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 randomized and its answers are correct 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 computing the edit distance between strings. When k is small, the state-of-the-art solution [52] 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 randomized and outputs all substrings at edit distance at most k from the pattern with high probability. Another related 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 randomized 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 on par with that of the Hamming distance? In this work, we answer this question affirmatively.

A. 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 takes $\tilde{\mathcal{O}}(k^8)$ amortized time per character of the text (Theorem IV.6). The algorithm is randomized, and its answers are correct with high probability.

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 available for read-only random access, 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 takes $\tilde{\mathcal{O}}(k^6)$ amortized time per character of the text (Theorem IV.5).

[2]With high probability means with probability at least $1 - n^{-c}$ for any predefined constant $c > 0$.

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 III.18). The sketch is of size $\tilde{\mathcal{O}}(k^2)$, and it can be built using a streaming algorithm that takes $\tilde{\mathcal{O}}(nk)$ total time and uses $\tilde{\mathcal{O}}(k^2)$ space. Given the sketches of two strings X and Y, in $\tilde{\mathcal{O}}(k^3)$ time and $\tilde{\mathcal{O}}(k^2)$ space, we can compute the edit distance between X and 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 [36].

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

II. Preliminaries

A *string* Y is a sequence of characters from a given *alphabet* Σ indexed from 1 to $|Y|$. By $Y[i]$ we denote the i-th character of Y. For a string Y of length $n := |Y|$, we denote its *reverse* $Y[n]Y[n-1]\cdots Y[1]$ by $\overline{Y}$. We define $Y[i\,..\,j]$ to be equal to $Y[i]\cdots Y[j]$ 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 $Y[i]\cdots Y[j-1]$ and $Y[i+1]\cdots Y[j]$, 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 *string period* of a string T if $X = T[1\,..\,x]$ and $T[i] = T[i+x]$ for all $i \in \{1,\ldots,|T|-x\}$. 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 $|X|$.

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 that 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).

A. 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 $|X| \ne |Y|$).

Definition II.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 and Y are clear from context. The cost of an alignment $\mathcal{A} = (x_t,y_t)_{t=1}^m$ is at least its *width*, defined as $\mathsf{width}(\mathcal{A}) = \max_{t=1}^m |x_t - y_t|$. 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 subscript if X and Y are clear from context. Observe that $|\mathcal{B}_{X,Y}(\mathcal{A})| = 1 + \mathsf{cost}_{X,Y}(\mathcal{A})$.

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 $\mathcal{M}_{X,Y}(\mathcal{A})$ is a non-crossing matching of X, Y.

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 $(\max(x,\min(x',x_t)) - x + 1, \max(y,\min(y',y_t)) - y + 1)_{t=1}^m$.

[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.

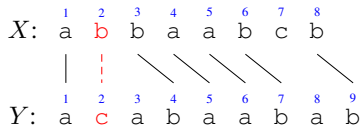

Fig. 1. Consider strings $X = \texttt{abbaabcb}$ and $Y = \texttt{acabaabab}$ and a cost-4 alignment $\mathcal{A} : (1,1),(2,2),(3,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,8),(8,9),(9,10)$. The breakpoints are $\mathcal{B}(\mathcal{A}) = \{(2,2),(3,3),(7,8),(8,8),(9,10)\}$; the first 4 breakpoints correspond 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.

Fact II.2. *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 $|(x'-x)-(y'-y)| \le \mathsf{ed}(X[x\,..\,x'), Y[y\,..\,y')) \le \mathsf{cost}(\mathcal{A}_{[x\,..\,x'),[y\,..\,y')}) \le \mathsf{cost}(\mathcal{A})$.*

III. 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 $T[\ell\,..\,r]$ is a *k-edit occurrence* of P if $\mathsf{ed}(P, T[\ell\,..\,r]) \le k$, 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 III.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 [47], 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 $\Sigma = [0\,..\,n^{\mathcal{O}(1)})$.

A. 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 [47], [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 to 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 $|r-p-(\ell_i-\ell_{i-1})| \le k$. 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 III.2 (k-periodic string)**.** *A string X is k-periodic if there exists a primitive string Q with $|Q| \le |X|/128k$ 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 III.3 (of [12, Theorem 5.1])**.** *Let $X \in \Sigma^m$, $k \in [1\,..\,m]$, and $Y \in \Sigma^n$ with $n \le 2m$. 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], [36], 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 III.4 (Chain of occurrences)**.** *Consider strings $X, Y \in \Sigma^*$ and an integer $k \in \mathbb{Z}_{\ge 0}$. 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 $p_j \in \mathsf{OCC}_{k'}^E(X,Y) \setminus \mathsf{OCC}_{k'-1}^E(X,Y)$ for $j \in [1\,..\,c]$.

Corollary III.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 $n \le 2m$. If X is k-periodic with period Q, then $\mathsf{OCC}_k^E(X,Y)$ can be decomposed into $\mathcal{O}(k^3)$ chains whose difference strings are of the form $\mathsf{rot}^s(Q)$, where $|m-s| \le 10k$.*

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 to maintain a sketch $\mathsf{sk}_k^E(T(p\,..\,r])$ for all active $p \in \mathsf{OCC}_k^E(P,T)$. If sk_k^E were a rolling sketch (like the k-mismatch sketches of [16]), we would compute $\mathsf{sk}_k^E(D)$ 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 $\mathsf{sk}_k^E(D) = \mathsf{sk}_k^E(T(p_j\,..\,p_{j+1}])$ to transform $\mathsf{sk}_k^E(T(p_j\,..\,r])$ into $\mathsf{sk}_k^E(T(p_{j+1}\,..\,r])$. 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 $p_j \in \mathsf{OCC}_k^E(P_{i-1},T)$ 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., when $|r-p_j-(\ell_i-\ell_{i-1})| \le k$. 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 to store 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 of D that 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 IV. Below, we outline the ideas behind our two main conceptual and technical contributions: the encoding $\mathsf{qGR}(\cdot,\cdot)$ and the sketch $\mathsf{sk}^\mathsf{q}(\cdot)$. The underlying details and proofs can be found in the full version of this paper.

B. 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 II.2, 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 III.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.

Fact III.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.

[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 [53].

Definition III.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 III.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 $\mathcal{M}_k(X, Y) = \mathcal{M}_k(\mathsf{num}(X^M), \mathsf{num}(Y^M))$.*

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 III.10. *Let $M = \mathcal{M}_k(X, Y)$ for strings $X, Y \in \Sigma^*$ and a positive integer $k \geq \mathsf{ed}(X, Y)$. Then, $|\mathsf{LZ}(X^M)|$, $|\mathsf{LZ}(Y^M)| = \mathcal{O}(k^2)$, and X^M, Y^M contain $\mathcal{O}(k)$ dummy segments.*

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 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. 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 III.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 III.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 III.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 $\mathcal{M}_k(X, Y)$.

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 III.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 $\mathsf{GA}_{k+1}(\$_1 X, \$_2 Y)$, 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. We summarize the properties of qGR as follows:

Theorem III.14. *Consider $X, Y, Z \in \Sigma^{\leq n}$ and $k \in \mathbb{Z}_+$.*

(a) The encoding $\mathsf{qGR}_k(X, Y)$ occupies $\mathcal{O}(k^2 \log^4 n)$ space.
(b) Given $\mathsf{qGR}_k(X, Y)$, we can compute $\min(k+1, \mathsf{ed}(X, Y))$ in $\tilde{\mathcal{O}}(k^2)$ time and space.
(c) Assume $X = X_p X_s$ and $Y = Y_p Y_s$, where $\big||X_p| - |Y_p|\big|$, $\big||X_s| - |Y_s|\big| = \mathcal{O}(k)$. Given $\mathsf{qGR}_{k+||X_s|-|Y_s||}(X_p, Y_p)$ and $\mathsf{qGR}_{k+||X_p|-|Y_p||}(X_s, Y_s)$, the encoding $\mathsf{qGR}_k(X, Y)$ can be computed in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.
(d) Given $\mathsf{qGR}_d(X, Y)$ and $\mathsf{qGR}_d(Y, Z)$ for $d \geq \mathsf{ed}(X, Y) + 2k$, the encoding $\mathsf{qGR}_k(X, Y)$ can be computed in $\tilde{\mathcal{O}}(d^5)$ time using $\tilde{\mathcal{O}}(d^2)$ space.
(e) Given $\mathsf{qGR}_k(X, Y)$ and an integer $k' \leq k$, the encoding $\mathsf{qGR}_{k'}(X, Y)$ can be computed in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.
(f) Given $X, Y \in \Sigma^{\mathcal{O}(k^2)}$, the encoding $\mathsf{qGR}_k(X, Y)$ can be constructed in $\tilde{\mathcal{O}}(k^3)$ time and $\tilde{\mathcal{O}}(k^2)$ space.

C. Edit Distance Sketches

Recall that we need an edit-distance sketch sk^E allowing to retrieve $\mathsf{qGR}_k(X, Y)$ 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], [36] are designed just 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], [36], 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 III.15 (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 III.16 (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 provides a complete black-box interface of the properties of the CGK algorithm utilized in our sketches. It also encapsulates Nisan's pseudorandom generator [46] that reduces the number of (shared) random bits.

Proposition III.17. *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 $\mathsf{W}(n,r,Y)$:*

$$\Pr_r \begin{bmatrix} \mathcal{A}_{\mathsf{W}} \in \mathsf{GA}_{c\cdot\mathsf{ed}(X,Y)^2}(X,Y) \\ \textit{and} \\ \mathsf{width}(\mathcal{A}_{\mathsf{W}}) \le c \cdot \mathsf{ed}(X,Y) \end{bmatrix} \ge 1 - \delta.$$

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 III.17, 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 III.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 each position $t \in [1\,..\,m]$ (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[6] containing $[x_t\,..\,x'_t)$. 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.

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 each forward context 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], [36]. 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 developed tools are then used to compute $\mathsf{ed}(X,Y)$ (or certify $\mathsf{ed}(X,Y) > k$) and retrieve the greedy encoding $\mathsf{GR}_k(X,Y)$. We summarize the properties of the edit distance sketches below:

Theorem III.18. *For every constant $\delta \in (0,1)$, there is a sketch sk^E_k (parameterized 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 $\mathsf{sk}^E_k(S)$ of $S \in \Sigma^{\leq n}$ takes $\mathcal{O}(k^2\log^3 n)$ bits. Given streaming access to S, it can be constructed in $\tilde{\mathcal{O}}(k)$ amortized time per character 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 $X, Y \in \Sigma^{\leq n}$, 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.*

As a corollary, we show the following result:

Corollary III.19. *There is a sketch $\mathsf{sk}^{\mathsf{q}}_k$ (parameterized 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 $\mathcal{O}(\log^2 n \log\delta^{-1})$ random bits) such that:*

- *(a) The sketch $\mathsf{sk}^{\mathsf{q}}_k(S)$ of $S \in \Sigma^{\leq n}$ takes $\mathcal{O}(k^2\log^3 n\log\delta^{-1})$ bits. Given streaming access to S, it can be constructed in $\tilde{\mathcal{O}}(k\log\delta^{-1})$ amortized time per character using $\tilde{\mathcal{O}}(k^2\log\delta^{-1})$ 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 $\tilde{\mathcal{O}}(k^5\log\delta^{-1})$ time and uses $\tilde{\mathcal{O}}(k^2\log\delta^{-1})$ space.*

[6] A dyadic interval is of the form $[i2^j\,..\,(i+1)2^j)$ for integers $i, j \geq 0$.

IV. Pattern Matching with k Edits

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

A. Periodicity under Edit Distance

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

Definition III.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 IV.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 (where $|Q| \leq |X|/128k$), then either Y is not k-periodic, or Y is k-periodic with k-period Q.*

Note that Claim IV.1 implies, in particular, that a string can have at most one k-period.

B. Semi-streaming Algorithm

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

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

Let $\Pi = \{P_1, P_2, \ldots, P_z\}$ be the resulting set of prefixes, with $1 = |P_1| < \cdots < |P_z| = |P|$. 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 the endpoints of its occurrence in P_i. 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 (see Corollary III.5), the encodings $\mathsf{qGR}_{29k}(D, D)$ and $\mathsf{qGR}_{29k}(P(\ell_{i-1}\,..\,\ell_i], D^\infty[1\,..\,\Delta])$ for all $\Delta \in [\max(0, \Delta_i - 2k)\,..\,\Delta_i]$. The total size of the precomputed data is $\tilde{\mathcal{O}}(k^4)$ by Theorem III.14(a).

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:

Corollary IV.2. *There is an algorithm that, given an integer $k \in \mathbb{Z}_+$ and strings X, Y, with read-only random access to X and streaming access to Y, can construct $\mathsf{qGR}_k(X, Y[1\,..\,y])$ and $\mathsf{qGR}_k(X[1\,..\,y], Y[1\,..\,y])$ (if $y \leq |X|$) as soon as $Y[y]$ arrives. The algorithm uses $\tilde{\mathcal{O}}(k^2)$ space and takes $\tilde{\mathcal{O}}(k^3)$ time per character of Y plus $\tilde{\mathcal{O}}(k^5)$ time per encoding constructed.*

Proof. We maintain a buffer of $2k^2 + k$ trailing characters of Y. Moreover, for each $x \in [0\,..\,|X|]$ divisible by k^2, we construct $\mathsf{qGR}_k(X[1\,..\,x], Y[1\,..\,x])$ while scanning $Y(x\,..\,x+k^2]$. This is trivial if $x = 0$. Otherwise, we construct $\mathsf{qGR}_k(X(x-k^2\,..\,x], Y(x-k^2\,..\,x])$ using Theorem III.14(f) and combine this encoding with the already constructed $\mathsf{qGR}_k(X[1\,..\,x-k^2], Y[1\,..\,x-k^2])$ using Theorem III.14(c). This process uses $\tilde{\mathcal{O}}(k^2)$ space and takes $\tilde{\mathcal{O}}(k^5)$ time, which is distributed evenly among k^2 scanned characters.

As soon as $Y[y]$ arrives, the algorithm has access to $\mathsf{qGR}_k(X[1\,..\,x], Y[1\,..\,x])$ for some x such that $x \le \min(|X|, y) \le x + 2k^2$. Upon a query for $\mathsf{qGR}_k(X[1\,..\,y], Y[1\,..\,y])$ (valid if $y \le |X|$), we construct $\mathsf{qGR}_k(X(x\,..\,y], Y(x\,..\,y])$ using Theorem III.14(f) and then obtain $\mathsf{qGR}_k(X[1\,..\,y], Y[1\,..\,y])$ using Theorem III.14(c). Upon a query for $\mathsf{qGR}_k(X, Y[1\,..\,y])$ we proceed similarly, but we immediately return $\perp$ whenever $\big|y - |X|\big| > k$. The latter guarantees that $x \le \max(|X|, y) \le x+2k^2+k$ whenever $\mathsf{qGR}_k(X(x\,..\,|X|], Y(x\,..\,y])$ needs to be constructed. □

a) 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 $i \in [2\,..\,z]$, we define *active* k-edit occurrences of P_{i-1} as those whose right endpoints belong to $[r - \Delta_i\,..\,r]$, and denote $\mathsf{aOCC}^E_k(P_{i-1}, T) := \mathsf{OCC}^E_k(P_{i-1}, T) \cap [r - \Delta_i\,..\,r]$. By Corollaries III.3 and III.5, $\mathsf{aOCC}^E_k(P_{i-1}, T)$ forms $\mathcal{O}(k^3)$ chains. For each chain $\mathcal{C}$, we store the following information:

1) The rightmost position p^* 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 $T[1\,..\,p]$ for every endpoint $p \in \mathcal{C}$;
3) If $|\mathcal{C}| \ge 2$, a shift s such that the difference of $\mathcal{C}$ is $D = \mathsf{rot}^s(Q_{i-1})$.

Moreover, we maintain an instance of the algorithm of Corollary IV.2 with $X = P(\ell_{i-1}\,..\,\ell_i]$, $Y = T(p^*\,..]$, and threshold k. If $|\mathcal{C}| \ge 2$ and the difference if $\mathcal{C}$ is D, we also maintain another instance with $X = D^\infty$, $Y = T(p^*\,..]$, and threshold $29k$. We run these subroutines as long as p^* remains the rightmost position in $\mathcal{C}$.

b) 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 $|P_i| = 1$ implies $r \in \mathsf{OCC}^E_1(P_i, T)$, and we naively check whether $r \in \mathsf{OCC}^E_0(P_i, T)$. Below, we consider three possible cases for $i \ge 2$.

c) Case 1: P_{i-1} is k-periodic and P_i is not k-periodic: By construction, in this case $\ell_{i-1} + 1 = \ell_i$. We have $r \in \mathsf{OCC}^E_k(P_i, T)$ if and only if at least one of the following conditions is satisfied:

1) $r \in \mathsf{OCC}^E_{k-1}(P_{i-1}, T)$: 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) $r - 1 \in \mathsf{OCC}^E_{k-1}(P_{i-1}, T)$ and $P[\ell_i] \ne T[r]$: 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) $r - 1 \in \mathsf{OCC}^E_k(P_{i-1}, T)$ and $P[\ell_i] = T[r]$: 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) $r - 1 \in \mathsf{OCC}^E_{k-1}(P_i, T)$: 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 $r \in \mathsf{OCC}^E_k(P_i, T)$, in $\mathcal{O}(k^3)$ time using $\mathsf{aOCC}^E_k(P_{i-1}, T)$ and $\mathsf{aOCC}^E_k(P_i, T) \setminus \{r\}$. 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 final two cases, we use the following simple observation that is a consequence of Fact II.2:

Observation IV.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*

$$d := \min_{r'}(\mathsf{ed}_{i-1}(r') + \mathsf{ed}(P(\ell_{i-1}\,..\,\ell_i], T(r'\,..\,r])),$$

where r' is taken over $\mathsf{OCC}^E_k(P_{i-1}, T) \cap [r-\Delta_i\,..\,r-\Delta_i+2k]$. If $d \le k$, then $\mathsf{ed}_i(r) = d$; otherwise, $\mathsf{ed}_i(r) > k$.

Hence, to decide whether $r \in \mathsf{OCC}^E_k(P_i, T)$, it suffices to compute the value $\min(d, k+1)$, where d is as defined above.

d) Case 2: P_{i-1} is not k-periodic: In this case, $\mathsf{aOCC}^E_k(P_{i-1}, T)$ is stored as $\mathcal{O}(k^3)$ chains of size one. Therefore, we can find the positions $r' \in \mathsf{OCC}^E_k(P_{i-1}, T) \cap [r - \Delta_i\,..\,r - \Delta_i + 2k]$ in $\mathcal{O}(k^3)$ time. For each such position r', we construct $\mathsf{qGR}_k(P(\ell_{i-1}\,..\,\ell_i], T(r'\,..\,r])$ using the algorithm of Corollary IV.2, in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space. We then retrieve $\min(k+1, \mathsf{ed}(P(\ell_{i-1}\,..\,\ell_i], T(r'\,..\,r])))$ using Theorem III.14(b) in $\tilde{\mathcal{O}}(k^2)$ time and space. As we also know the smallest edit distance between P_{i-1} and a suffix of $T[1\,..\,r']$, we can compute d in $\tilde{\mathcal{O}}(k^6)$ time and $\tilde{\mathcal{O}}(k^2)$ space.

e) Case 3: P_{i-1} and P_i are k-periodic: We can identify all positions $r' \in \mathsf{OCC}^E_k(P_{i-1}, T) \cap [r - \Delta_i\,..\,r - \Delta_i + 2k]$ 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 $\mathsf{ed}(P(\ell_{i-1}\,..\,\ell_i], T(r'\,..\,r])$ as we already know the smallest edit distance from P_{i-1} to a suffix of $T[1\,..\,r']$. If $|\mathcal{C}| = 1$, we proceed as in Case 2. Otherwise, we use quasi-greedy encodings. On a high level, our goal is to compute the edit distance between $P(\ell_{i-1}\,..\,\ell_i]$ and $T(r'\,..\,r]$ via a string $D^\infty[1\,..\,r - r']$, where D is the difference of $\mathcal{C}$.

Lemma IV.4. $\mathsf{ed}(P(\ell_{i-1}\,..\,\ell_i], D^\infty[1\,..\,r - r']) \le 27k$.

Proof. As P_{i-1} and P_i are k-periodic, by Claim IV.1, we obtain that $P_i = P[1\,..\,\ell_i]$ is k-periodic with k-period $Q_i = Q_{i-1}$, that is, $\mathsf{ed}(P_i, Q_i^\infty[1\,..\,t]) \le 2k$ for some $t \in \mathbb{Z}_{\ge 0}$; in particular, $|t - \ell_i| \le 2k$. By Fact II.2, there exists $t' \in [0\,..\,t]$ such that $|t' - \ell_{i-1}| \le 2k$ and $\mathsf{ed}(Q_i^\infty(t'\,..\,t], P(\ell_{i-1}\,..\,\ell_i]) \le 2k$. Hence, the triangle inequality yields $\mathsf{ed}(Q_i^\infty(\ell_{i-1}\,..\,\ell_i], P(\ell_{i-1}\,..\,\ell_i]) \le 6k$.

By Corollary III.5, there is $s \in \mathbb{Z}$ such that $|s - \ell_i| \le 10k$ and $D = \mathsf{rot}^s(Q_i)$. The latter implies $D^\infty = Q_i^\infty(s\,..]$

and, in particular, $D^\infty[1\,..\,r-r'] = Q_i^\infty(s\,..\,s+r-r']$. Due to $|(r-r')-(\ell_i-\ell_{i-1})| \le k$, we get $\mathsf{ed}(D^\infty[1\,..\,r-r'], Q_i^\infty(\ell_{i-1}\,..\,\ell_i]) \le 21k$. Applying the triangle inequality one more time, we derive the claim. □

By Theorem III.14(d), the encodings $\mathsf{qGR}_{29k}(P(\ell_{i-1}\,..\,\ell_i], D^\infty[1\,..\,r-r'])$ and $\mathsf{qGR}_{29k}(D^\infty[1\,..\,r-r'], T(r'\,..\,r])$ are sufficient to derive $\mathsf{qGR}_k(P(\ell_{i-1}\,..\,\ell_i], T(r'\,..\,r])$. Whereas $\mathsf{qGR}_{29k}(P(\ell_{i-1}\,..\,\ell_i], D^\infty[1\,..\,r-r'])$ has been precomputed, we must construct $\mathsf{qGR}_{29k}(D^\infty[1\,..\,r-r'], T(r'\,..\,r])$ using the available information. Let p be the rightmost position in $\mathcal{C}$.

1) Recall that at position $(p+1)$ we started an instance of Corollary IV.2. Therefore, upon reaching r, we can retrieve $\mathsf{qGR}_{29k}(T(p\,..\,r], D^\infty[1\,..\,r-p])$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.
2) By the definition of chains, $T(r'\,..\,p] = D^j$ for $j = (p-r')/q_i$. By Theorem III.14(c), we can use the precomputed encoding $\mathsf{qGR}_{29k}(D, D)$ to retrieve $\mathsf{qGR}_{29k}(T(r'\,..\,p], D^j)$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.
3) Applying Theorem III.14(c) again, we construct $\mathsf{qGR}_{29k}(T(r'\,..\,r], D^\infty[1\,..\,r-r'])$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space.

f) 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 $i \in [1\,..\,z]$, 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 III.5). For each rotation R, we run a constant-space linear-time deterministic pattern matching algorithm [49]. 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 $\mathcal{O}(1)$ amortized 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 $r-q_i$, if it exists, by checking each chain in turn. We then check whether the smallest edit distance from a suffix of $T[1\,..\,r]$ to P_i equals to $\mathsf{ed}(\mathcal{C})$ and whether $T(r-q_i\,..\,r]$ 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 $\mathcal{C}$ and update its rightmost position). To finalize the update of the chains, we must delete the k-edit occurrences that becomes inactive: for each i and for each chain of k-edit occurrences of P_i, we "delete" the first k-edit occurrence if it ends at position $r-\Delta_i$. For this, we simply update the total number of occurrences in the chain (and delete the entire chain if it becomes empty).

3) Analysis: We summarize the results of this section.

Theorem IV.5. *Consider 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 $\tilde{\mathcal{O}}(k^6)$ amortized time per character of the text T.*

Proof. Let us first bound the space complexity of the algorithm. For each $i \in [1\,..\,z]$, with $z = \mathcal{O}(\log m)$, we maintain the set $\mathsf{aOCC}_k(P_i, T)$ as $\mathcal{O}(k^3)$ chains. For each chain, we run one or two instances of the algorithm of Corollary IV.2, which take $\tilde{\mathcal{O}}(k^2)$ space each. 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 analyze the running time. Updating the chains takes $\tilde{\mathcal{O}}(1)$ time. At any time, we maintain $\tilde{\mathcal{O}}(k^3)$ instances of the algorithm of Corollary IV.2 that take $\mathcal{O}(k^3)$ time per character. To test each position r, we spend $\tilde{\mathcal{O}}(k \cdot k^5)$ time. The pattern matching algorithms for the rotations of Q_i take $\tilde{\mathcal{O}}(k)$ amortized time in total. □

C. Streaming Algorithm

We now modify the algorithm for the semi-streaming model to develop a fully streaming algorithm. W.l.o.g., assume $k \le m$ and take $\delta = n^{-c}$ for sufficiently large constant c.

1) Preprocessing: We define the prefixes P_i and their string periods Q_i exactly in the same way as in Section IV-B. Recall that $\ell_i = |P_i|$ and, if defined, $q_i = |Q_i|$. For every $i > 1$, we store the following information, where all sketches sk^{q} (of Corollary III.19) are parameterized by probability δ, maximal length n, the alphabet of P and T, and a seed of $\mathcal{O}(\log^2 n \log \delta^{-1})$ random bits:

1) The character $P[\ell_i]$ and the sketch $\mathsf{sk}^{\mathsf{q}}_{3k}(P(\ell_{i-1}\,..\,\ell_i])$.
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 encodings $\mathsf{qGR}_{29k}(P(\ell_{i-1}\,..\,\ell_i], D^\infty[1\,..\,\Delta])$ for all $\Delta \in [\max(0, \Delta_i - 2k)\,..\,\Delta_i]$.
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 $\mathsf{sk}^{\mathsf{q}}_{29k}(D)$ and $\mathsf{sk}^{\mathsf{q}}_{29k}(D[1\,..\,(\Delta_i - 2k) \bmod q_i])$, as well as the string $D^\infty(\Delta_i - 2k\,..\,\Delta_i]$.

2) Main Stage: As in Section IV-B, 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 rightmost position p^* in $\mathcal{C}$, the size $|\mathcal{C}|$, and the smallest edit distance, $\mathsf{ed}(\mathcal{C})$, from a suffix of $T[1\,..\,p^*]$ to P_{i-1}. If $|\mathcal{C}| \ge 2$, we also store the difference of $\mathcal{C}$ (defined by the shift of the rotation of Q_{i-1}).

Moreover, we run the following procedures as long as p^* remains the rightmost position in $\mathcal{C}$. First, we maintain an instance of Corollary III.19(a) for computing the sketch $\mathsf{sk}^{\mathsf{q}}_{3k}(T(p^*\,..\,p^*+\Delta_i-2k])$. If $|\mathcal{C}| \ge 2$ and the difference of $\mathcal{C}$ is D, we also build the encoding $\mathsf{qGR}_{29k}(T(p^*\,..\,r], D^\infty[1\,..\,r-p^*])$ whenever $r \bmod q_i = p^* \bmod q_i$ or $r \bmod q_i \in [p^* + \Delta_i - 2k\,..\,p^* + \Delta_i] \bmod q_i$. For this, for every $\ell \in \mathbb{Z}_{\ge 0}$, we construct $\mathsf{sk}^{\mathsf{q}}_{29k}(T(p^*+q_i\ell\,..\,p^*+q_i(\ell+1)])$ and $\mathsf{sk}^{\mathsf{q}}_{29k}(T(p^*+q_i\ell\,..\,p^*+q_i\ell+(\Delta_i-2k) \bmod q_i])$ using Corollary III.19(a) as the algorithm scans the underlying fragments of T. Once

such a scan is completed, we apply Corollary III.19(b) to derive either $\mathsf{qGR}_{29k}(T(p^*+q_i\ell\,..\,p^*+q_i(\ell+1)],D)$ or $\mathsf{qGR}_{29k}(T(p^*+q_i\ell\,..\,p^*+q_i\ell+(\Delta_i-2k) \bmod q_i],D[1\,..\,(\Delta_i-2k) \bmod q_i])$ using the precomputed sketches $\mathsf{sk}^{\mathsf{q}}_{29k}(D)$ and $\mathsf{sk}^{\mathsf{q}}_{29k}(D[1\,..\,(\Delta_i-2k) \bmod q_i])$. Combined with the already available $\mathsf{qGR}_{29k}(T(p^*\,..\,p^*+q_i\ell],D^\infty[1\,..\,q_i\ell])$, this lets us derive $\mathsf{qGR}_{29k}(T(p^*\,..\,r],D^\infty[r-p^*])$ (via Theorem III.14(c)) whenever $r \bmod q_i = p^* \bmod q_i$ or $r \bmod q_i = (p^*+\Delta_i-2k) \bmod q_i$. In the remaining case of $r \bmod q_i \in (p^*+\Delta_i-2k\,..\,p^*+\Delta_i] \bmod q_i$, we use $\mathsf{qGR}_{29k}(T(p^*\,..\,r),D^\infty[1\,..\,r-p^*))$ and combine it (via Theorem III.14(c)) with $\mathsf{qGR}_{29k}(T[r],D^\infty[r-p^*])$, constructed using Theorem III.14(f); note that $D^\infty[r-p^*]$ occurs within $D^\infty(\Delta_i-2k\,..\,\Delta_i]$ because $(r-p^*) \bmod q_i \in (\Delta_i-2k\,..\,\Delta_i] \bmod q_i$.

a) 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. In Case 2, instead of Corollary IV.2, we use Corollary III.19(b) to construct $\mathsf{qGR}_{3k}(P(\ell_{i-1}\,..\,\ell_i],T(r'\,..\,r'+\Delta_i-2k])$ from $\mathsf{sk}^{\mathsf{q}}_{3k}(P(\ell_{i-1}\,..\,\ell_i])$ and $\mathsf{sk}^{\mathsf{q}}_{3k}(T(r'\,..\,r'+\Delta_i-2k])$. Next, we derive $\mathsf{qGR}_{3k+(r-r'-\Delta_i)}(P(\ell_{i-1}\,..\,\ell_i],T(r'\,..\,r'+\Delta_i-2k])$ using Theorem III.14(e), build $\mathsf{qGR}_{2k}(\varepsilon,T(r'+\Delta_i-2k\,..\,r])$ using Theorem III.14(f), and finally obtain $\mathsf{qGR}_k(P(\ell_{i-1}\,..\,\ell_i],T(r'\,..\,r])$ via Theorem III.14(c). It remains to explain how we modify the algorithm for Case 3.

We exploit Observation IV.3 again. We can find all positions r' in $\mathcal{O}(k^3)$ time. To test a position r', it suffices to construct $\mathsf{qGR}_k(P(\ell_{i-1}\,..\,\ell_i],T(r'\,..\,r])$. Let $\mathcal{C}$ be the chain with difference D that contains r'. If $|\mathcal{C}|=1$, we proceed as in Case 2; otherwise, we use the quasi-greedy encodings as follows. By Lemma IV.4, $\mathsf{ed}(P(\ell_{i-1}\,..\,\ell_i],D^\infty[1\,..\,r-r'])\le 27k$ and hence, by Theorem III.14(d), $\mathsf{qGR}_k(P(\ell_{i-1}\,..\,\ell_i],T(r'\,..\,r])$ can be computed from the encodings $\mathsf{qGR}_{29k}(P(\ell_{i-1}\,..\,\ell_i],D^\infty[1\,..\,r-r'])$ and $\mathsf{qGR}_{29k}(D^\infty[1\,..\,r-r'],T(r'\,..\,r])$. The former encoding has been precomputed and is stored explicitly. Hence, we only need to explain how to compute latter one. Let p be the rightmost position in the chain $\mathcal{C}$.

1) Recall that $T(r'\,..\,p]=D^j$ for $j=(p-r')/q_i$. We first compute $\mathsf{qGR}_{29k}(D,D)$ from $\mathsf{sk}^{\mathsf{q}}_{29k}(D)$ via Corollary III.19(b), and then $\mathsf{qGR}_{29k}(T(r'\,..\,p],D^j)=\mathsf{qGR}_{29k}(D^j,D^j)$ in $\tilde{\mathcal{O}}(k^5)$ time and $\tilde{\mathcal{O}}(k^2)$ space as in Section IV-B.
2) At the position $(p+1)$ we launched a procedure computing $\mathsf{qGR}_{29k}(T(p\,..\,r],D^\infty[1\,..\,r-p])$ whenever $r \bmod q_i \in [p+\Delta_i-2k\,..\,p+\Delta_i] \bmod q_i$. This condition is satisfied due to $r-r'\in[\Delta_i-2k\,..\,\Delta_i]$ and $r' \bmod q_i = p \bmod q_i$, so the encoding $\mathsf{qGR}_{29k}(T(p\,..\,r],D^\infty[1\,..\,r-p])$ is available.
3) We finally concatenate $\mathsf{qGR}_{29k}(T(r'\,..\,p],D^j)$ and $\mathsf{qGR}_{29k}(T(p\,..\,r],D^\infty[1\,..\,r-p])$ via Theorem III.14(c), obtaining $\mathsf{qGR}_{29k}(T(r'\,..\,r],D^\infty[1\,..\,r-r'])$.

b) 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. For this, we use the algorithm of Section IV-B, but replace the constant-space pattern matching algorithm with the streaming pattern matching algorithm [47] 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 $1-\delta$.

3) Analysis: We summarize the results of this section.

Theorem IV.6. *Consider 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 $\tilde{\mathcal{O}}(k^8)$ amortized time per character of the text T. The algorithm computes $\mathsf{OCC}^E_k(P,T)$ correctly with high probability.*

Proof. By Corollary III.5, for a fixed index i, only $\mathcal{O}(k)$ rotations of Q_i can be a difference of a chain of occurrences of P_i. Each sketch $\mathsf{sk}^{\mathsf{q}}_{29k}(\cdot)$ takes $\tilde{\mathcal{O}}(k^2)$ space (Corollary III.19(a)) and $\mathsf{qGR}_{29k}(\cdot,\cdot)$ takes $\tilde{\mathcal{O}}(k^2)$ space as well (Theorem III.14(a)). Therefore, the information computed during the preprocessing stage occupies $\tilde{\mathcal{O}}(k^4)$ space. During the main stage, we store $\tilde{\mathcal{O}}(k^3)$ chains. For each chain, we run the algorithm of Corollary III.19(a) that takes $\tilde{\mathcal{O}}(k^2)$ space. The algorithm that computes the quasi-greedy encoding (Corollary III.19(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 Theorem III.14(c)(d), which requires 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 III.19(a) that takes $\tilde{\mathcal{O}}(k)$ amortized time per character. In addition, for every character we run $\tilde{\mathcal{O}}(k^3)$ instances of the algorithms of Theorem III.14(c)(f), taking $\tilde{\mathcal{O}}(k^8)$ time in total. For every character of T, updating the chains involves $\tilde{\mathcal{O}}(k)$ calls to the algorithms of Corollary III.19(b) and Theorem III.14(c)(d)(e), taking $\tilde{\mathcal{O}}(k^6)$ 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 [47] and that of Corollary III.19(b) that computes quasi-greedy encodings. These procedures are called $\mathrm{poly}(n,k)=\mathrm{poly}(n)$ times. By choosing the constant c in $\delta=n^{-c}$ large enough, we use the union bound to guarantee that the entire algorithm is correct with high probability. □

ACKNOWLEDGMENTS

Tomasz Kociumaka was partially supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship. Ely Porat was partially supported by ISF grants no. 1278/16 and 1926/19, by a BSF grant no. 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Program (grant no. 683064). Tatiana Starikovskaya was partially supported by the ANR-20-CE48-0001 grant from the French National Research Agency (ANR).

REFERENCES

[1] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams, "Simulating branching programs with edit distance and friends: Or: A polylog shaved is a lower bound made," in *STOC 2016*. ACM, 2016, pp. 375–388.

[2] K. R. Abrahamson, "Generalized string matching," *SIAM Journal on Computing*, vol. 16, no. 6, pp. 1039–1051, 1987.

[3] A. Amir, M. Lewenstein, and E. Porat, "Faster algorithms for string matching with k mismatches," *Journal of Algorithms*, vol. 50, no. 2, pp. 257–275, 2004.

[4] A. Babu, N. Limaye, J. Radhakrishnan, and G. Varma, "Streaming algorithms for language recognition problems," *Theoretical Computer Science*, vol. 494, pp. 13–23, 2013.

[5] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," *SIAM Journal on Computing*, vol. 47, no. 3, pp. 1087–1097, 2018.

[6] D. Belazzougui and Q. Zhang, "Edit distance: Sketching, streaming, and document exchange," in *FOCS 2016*. IEEE, 2016, pp. 51–60.

[7] D. Breslauer and Z. Galil, "Real-time streaming string-matching," *ACM Transactions on Algorithms*, vol. 10, no. 4, pp. 22:1–22:12, 2014.

[8] K. Bringmann and M. Künnemann, "Quadratic conditional lower bounds for string problems and dynamic time warping," in *FOCS 2015*. IEEE, 2015, pp. 79–97.

[9] D. Chakraborty, D. Das, and M. Koucký, "Approximate online pattern matching in sublinear time," in *FSTTCS 2019*, ser. LIPIcs, vol. 150. Schloss Dagstuhl, 2019, pp. 10:1–10:15.

[10] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *STOC 2016*. ACM, 2016, pp. 712–725.

[11] T. M. Chan, S. Golan, T. Kociumaka, T. Kopelowitz, and E. Porat, "Approximating text-to-pattern Hamming distances," in *STOC 2020*. ACM, 2020, pp. 643–656.

[12] P. Charalampopoulos, T. Kociumaka, and P. Wellnitz, "Faster approximate pattern matching: A unified approach," in *FOCS 2020*. IEEE, 2020, pp. 978–989.

[13] R. Clifford, A. Fontaine, E. Porat, B. Sach, and T. Starikovskaya, "Dictionary matching in a stream," in *ESA 2015*, ser. LNCS, vol. 9294. Springer, 2015, pp. 361–372.

[14] ——, "The k-mismatch problem revisited," in *SODA 2016*. SIAM, 2016, pp. 2039–2052.

[15] R. Clifford, M. Jalsenius, and B. Sach, "Cell-probe bounds for online edit distance and other pattern matching problems," in *SODA 2015*. SIAM, 2015, pp. 552–561.

[16] R. Clifford, T. Kociumaka, and E. Porat, "The streaming k-mismatch problem," in *SODA 2019*. SIAM, 2019, pp. 1106–1125.

[17] R. Cole and R. Hariharan, "Approximate string matching: A simpler faster algorithm," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1761–1782, 2002.

[18] F. Ergün, E. Grigorescu, E. S. Azer, and S. Zhou, "Streaming periodicity with mismatches," in *APPROX 2017*, ser. LIPIcs, vol. 81. Schloss Dagstuhl, 2017, pp. 42:1–42:21.

[19] ——, "Periodicity in data streams with wildcards," *Theory of Computing Systems*, vol. 64, no. 1, pp. 177–197, 2020.

[20] F. Ergün, H. Jowhari, and M. Sağlam, "Periodicity in streams," in *APPROX 2010*, ser. LNCS, vol. 6302. Springer, 2010, pp. 545–559.

[21] N. François, F. Magniez, M. de Rougemont, and O. Serre, "Streaming property testing of visibly pushdown languages," in *ESA 2016*, ser. LIPIcs, vol. 57. Schloss Dagstuhl, 2016, pp. 43:1–43:17.

[22] M. Ganardi, D. Hucke, D. König, M. Lohrey, and K. Mamouras, "Automata theory on sliding windows," in *STACS 2018*, ser. LIPIcs, vol. 96. Schloss Dagstuhl, 2018, pp. 31:1–31:14.

[23] M. Ganardi, D. Hucke, and M. Lohrey, "Querying regular languages over sliding windows," in *FSTTCS 2016*, ser. LIPIcs, vol. 65. Schloss Dagstuhl, 2016, pp. 18:1–18:14.

[24] ——, "Randomized sliding window algorithms for regular languages," in *ICALP 2018*, ser. LIPIcs, vol. 107. Schloss Dagstuhl, 2018, pp. 127:1–127:13.

[25] ——, "Sliding window algorithms for regular languages," in *LATA 2018*, ser. LNCS, vol. 10792. Springer, 2018, pp. 26–35.

[26] M. Ganardi, D. Hucke, M. Lohrey, and T. Starikovskaya, "Sliding window property testing for regular languages," in *ISAAC 2019*, ser. LIPIcs, vol. 149. Schloss Dagstuhl, 2019, pp. 6:1–6:13.

[27] M. Ganardi, A. Jeż, and M. Lohrey, "Sliding windows over context-free languages," in *MFCS 2018*, ser. LIPIcs, vol. 117. Schloss Dagstuhl, 2018, pp. 15:1–15:15.

[28] P. Gawrychowski, O. Merkurev, A. M. Shur, and P. Uznański, "Tight tradeoffs for real-time approximation of longest palindromes in streams," *Algorithmica*, vol. 81, no. 9, pp. 3630–3654, 2019.

[29] P. Gawrychowski, J. Radoszewski, and T. Starikovskaya, "Quasi-periodicity in streams," in *CPM 2019*, ser. LIPIcs, vol. 128. Schloss Dagstuhl, 2019, pp. 22:1–22:14.

[30] P. Gawrychowski and P. Uznański, "Towards unified approximate pattern matching for Hamming and L_1 distance," in *ICALP 2018*, ser. LIPIcs, vol. 107. Schloss Dagstuhl, 2018, pp. 62:1–62:13.

[31] P. Gawrychowski and T. Starikovskaya, "Streaming dictionary matching with mismatches," in *CPM 2019*, ser. LIPIcs, vol. 128. Schloss Dagstuhl, 2019, pp. 21:1–21:15.

[32] S. Golan, T. Kociumaka, T. Kopelowitz, and E. Porat, "The streaming k-mismatch problem: Tradeoffs between space and total time," in *CPM 2020*, ser. LIPIcs, vol. 161. Schloss Dagstuhl, 2020, pp. 15:1–15:15.

[33] S. Golan, T. Kopelowitz, and E. Porat, "Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams," in *ICALP 2018*, ser. LIPIcs, vol. 107. Schloss Dagstuhl, 2018, pp. 65:1–65:16.

[34] S. Golan and E. Porat, "Real-time streaming multi-pattern search for constant alphabet," in *ESA 2017*, ser. LIPIcs, vol. 87. Schloss Dagstuhl, 2017, pp. 41:1–41:15.

[35] R. Impagliazzo and R. Paturi, "On the complexity of k-SAT," *Journal of Computer and System Sciences*, vol. 62, no. 2, pp. 367–375, 2001.

[36] C. Jin, J. Nelson, and K. Wu, "An improved sketching algorithm for edit distance," in *STACS 2021*, ser. LIPIcs, vol. 187. Schloss Dagstuhl, 2021, pp. 45:1–45:16.

[37] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 249–260, 1987.

[38] S. Kosaraju, "Efficient string matching," 1987, unpublished manuscript.

[39] G. M. Landau and U. Vishkin, "Efficient string matching with k mismatches," *Theoretical Computer Science*, vol. 43, pp. 239–249, 1986.

[40] ——, "Fast parallel and serial approximate string matching," *Journal of Algorithms*, vol. 10, no. 2, pp. 157–169, 1989.

[41] F. Magniez, C. Mathieu, and A. Nayak, "Recognizing well-parenthesized expressions in the streaming model," *SIAM Journal on Computing*, vol. 43, no. 6, pp. 1880–1905, 2014.

[42] W. J. Masek and M. S. Paterson, "A faster algorithm computing string edit distances," *Journal of Computer and System Sciences*, vol. 20, no. 1, pp. 18–31, 1980.

[43] O. Merkurev and A. M. Shur, "Searching long repeats in streams," in *CPM 2019*, ser. LIPIcs, vol. 128. Schloss Dagstuhl, 2019, pp. 31:1–31:14.

[44] ——, "Searching runs in streams," in *SPIRE 2019*, ser. LNCS, vol. 11811. Springer, 2019, pp. 203–220.

[45] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001.

[46] N. Nisan, "Pseudorandom generators for space-bounded computation," *Combinatorica*, vol. 12, no. 4, pp. 449–461, 1992.

[47] B. Porat and E. Porat, "Exact and approximate pattern matching in the streaming model," in *FOCS 2009*. IEEE, 2009, pp. 315–323.

[48] J. Radoszewski and T. Starikovskaya, "Streaming k-mismatch with error correcting and applications," *Information and Computation*, vol. 271, p. 104513, 2020.

[49] W. Rytter, "On maximal suffixes and constant-space linear-time versions of KMP algorithm," *Theoretical Computer Science*, vol. 299, no. 1, pp. 763–774, 2003.

[50] S. C. Sahinalp and U. Vishkin, "Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract)," in *FOCS 1996*. IEEE, 1996, pp. 320–328.

[51] P. H. Sellers, "The theory and computation of evolutionary distances: Pattern recognition," *Journal of Algorithms*, vol. 1, no. 4, pp. 359–373, 1980.

[52] T. Starikovskaya, "Communication and streaming complexity of approximate pattern matching," in *CPM 2017*, ser. LIPIcs, J. Kärkkäinen, J. Radoszewski, and W. Rytter, Eds., vol. 78. Schloss Dagstuhl, 2017, pp. 13:1–13:11.

[53] A. Tiskin, "Semi-local string comparison: algorithmic techniques and applications," *Mathematics in Computer Science*, vol. 1, no. 4, pp. 571–603, 2008.