Approximate Circular Pattern Matching Under Edit Distance

Panagiotis Charalampopoulos
Birkbeck, University of London, UK

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski
University of Warsaw, Poland

Wojciech Rytter
University of Warsaw, Poland

Tomasz Waleń
University of Warsaw, Poland

Wiktor Zuba
CWI, Amsterdam, The Netherlands

Abstract

In the k-Edit Circular Pattern Matching (k-Edit CPM) problem, we are given a length-n text T, a length-m pattern P, and a positive integer threshold k, and we are to report all starting positions of the substrings of T that are at edit distance at most k from some cyclic rotation of P. In the decision version of the problem, we are to check if any such substring exists. Very recently, Charalampopoulos et al. [ESA 2022] presented $\mathcal{O}(nk^2)$-time and $\mathcal{O}(nk\log^3 k)$-time solutions for the reporting and decision versions of k-Edit CPM, respectively. Here, we show that the reporting and decision versions of k-Edit CPM can be solved in $\mathcal{O}(n+(n/m)\,k^6)$ time and $\mathcal{O}(n+(n/m)\,k^5\log^3 k)$ time, respectively, thus obtaining the first algorithms with a complexity of the type $\mathcal{O}(n+(n/m)\,\mathrm{poly}(k))$ for this problem. Notably, our algorithms run in $\mathcal{O}(n)$ time when $m=\Omega(k^6)$ and are superior to the previous respective solutions when $m=\omega(k^4)$. We provide a meta-algorithm that yields efficient algorithms in several other interesting settings, such as when the strings are given in a compressed form (as straight-line programs), when the strings are dynamic, or when we have a quantum computer.

We obtain our solutions by exploiting the structure of approximate circular occurrences of P in T, when T is relatively short w.r.t. P. Roughly speaking, either the starting positions of approximate occurrences of rotations of P form $\mathcal{O}(k^4)$ intervals that can be computed efficiently, or some rotation of P is almost periodic (is at a small edit distance from a string with small period). Dealing with the almost periodic case is the most technically demanding part of this work; we tackle it using properties of locked fragments (originating from [Cole and Hariharan, SICOMP 2002]).

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases circular pattern matching, approximate pattern matching, edit distance

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.24

Related Version *Full Version*: https://arxiv.org/abs/2402.14550

Funding *Solon P. Pissis*: Supported by the PANGAIA and ALPACA projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.
Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.
Wiktor Zuba: Received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement Grant Agreement No 101034253.

Acknowledgements We thank Tomasz Kociumaka for helpful discussions.

41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024).
Editors: Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov;
Article No. 24; pp. 24:1–24:22

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the classic pattern matching (PM) problem, we are given a length-n text T and a length-m pattern P, and we are to report all starting positions (called occurrences) of the fragments of T that are identical to P. This problem can be solved in the optimal $\mathcal{O}(n)$ time by, e.g., the famous Knuth-Morris-Pratt algorithm [29]. In many real-world applications, we are interested in locating not only the fragments of T which are identical to P, but also the fragments of T which are identical to any cyclic rotation of P. In this setting, the rotations of P form an equivalence class, represented by a single circular string. In the circular PM (CPM) problem, we are to report all occurrences of the fragments of T that are identical to some cyclic rotation of P. The CPM problem can also be solved in $\mathcal{O}(n)$ time [14].

Applications where circular strings are considered include the comparison of DNA sequences in bioinformatics [23, 4] as well as the comparison of shapes represented through directional chain codes in image processing [36, 35]. In both applications, it is not sufficient to look for exact (circular) matches. In bioinformatics, we need to account for DNA sequence divergence (e.g., in the comparison of different species or individuals); and in image processing, we need to account for small differences in the comparison of images (e.g., in classifying handwritten digits). This gives rise to the notion of edit distance on circular strings [34, 3].

We say that string U is a *(cyclic) rotation* of string V if $U = XY$ and $V = YX$ for some strings X, Y, and write $V = \mathsf{rot}^i(U)$, where $i = |X|$; e.g., $U = \mathtt{abcde}, X = \mathtt{ab}, Y = \mathtt{cde}, V = \mathtt{cdeab} = \mathsf{rot}^2(U)$. The *edit (Levenshtein) distance* $\delta_E(U, V)$ of two strings U and V is the minimal number of letter insertions, deletions and substitutions required to transform U to V. For two strings U and V and an integer $k > 0$, we write $U =_k V$ if $\delta_E(U, V) \leq k$ and we write $U \approx_k V$ if there exists a rotation U' of U such that $U' =_k V$.

For a string U composed of letters $U[0], \ldots, U[|U| - 1]$, by $U[i\,..\,j] = U[i\,..\,j + 1)$ we denote the *fragment of* U corresponding to the *substring* $U[i] \cdots U[j]$. We say that $T[p\,..\,p']$ is a *circular k-edit occurrence* of pattern P if $P \approx_k T[p\,..\,p']$. By $\mathsf{CircOcc}_k(P, T)$ we denote the set of starting positions of circular k-edit occurrences of P in T. Let us define k-Edit CPM (cf. Figure 1).

> ***k*-Edit CPM**
>
> **Input:** A text T of length n, a pattern P of length m, and a positive integer k.
>
> **Output:** A representation of the set $\mathsf{CircOcc}_k(P, T)$. (**Reporting version**)
> Any position $i \in \mathsf{CircOcc}_k(P, T)$, if there is any. (**Decision version**)

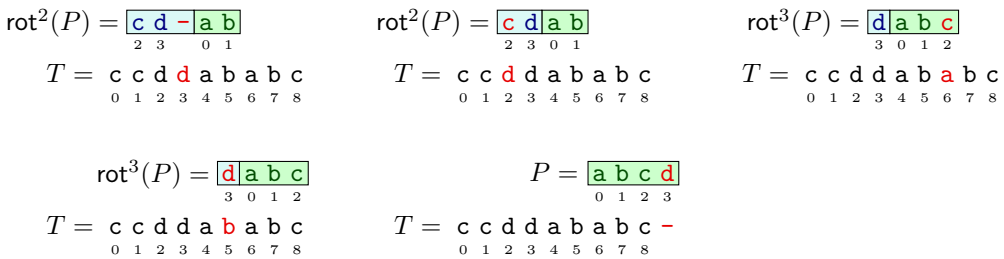

Figure 1 Illustration of the 1-edit circular occurrences of pattern $P = \mathtt{abcd}$ in text $T = \mathtt{ccddababc}$. We have $\mathsf{CircOcc}_1(P, T) = \{1, 2, 3, 5, 6\}$. The letters involved in an edit operation are coloured red.

Related work. The *Hamming distance* of two equal-length strings U and V is the number of mismatches between U and V; that is, the minimal number of letter substitutions required to transform U to V. Accounting for surplus or missing letters on top of substitutions poses significant challenges. For example, the Hamming distance of two length-n strings can be computed in $\mathcal{O}(n)$ time with a trivial algorithm, while it is known that their edit distance cannot be computed in $\mathcal{O}(n^{2-\epsilon})$ time, for any $\epsilon > 0$, under the Strong Exponential Time Hypothesis [5]. The situation is similar for (non-circular) approximate pattern matching. The k-Mismatch PM problem is quite well-understood as the upper bound of $\tilde{\mathcal{O}}(n + kn/\sqrt{m})$ due to Gawrychowski and Uznański [22], who provided a smooth tradeoff between the algorithms of Amir et al. [2] with running time $\tilde{\mathcal{O}}(n\sqrt{k})$ and Clifford et al. [18] with running time $\tilde{\mathcal{O}}(n + (n/m)k^2)$, is matched by a lower bound for so-called "combinatorial" algorithms.[1] Algorithms that are faster by polylogarithmic factors have been presented in [11, 12, 16]. In contrast, the complexity of the k-Edit PM problem is not yet settled: the current records are the classic $\mathcal{O}(nk)$-time algorithm of Landau and Vishkin [33] and the very recent $\tilde{\mathcal{O}}(n + (n/m)k^{3.5})$-time algorithm of Charalampopoulos et al. [17] improving the classic $\mathcal{O}(n + (n/m)k^4)$-time algorithm of Cole and Hariharan [20]. However, there is no known lower bound for k-Edit PM ruling out an $\mathcal{O}(n + (n/m)k^2)$-time algorithm.

Recent results in pattern matching under both the Hamming distance and the edit distance for various settings [8, 9, 13, 15, 16, 17, 19, 27, 30, 39] were fuelled by a novel characterization of the structure of approximate occurrences. It is folklore knowledge that if $n \leq 3m/2$, either pattern P has a single exact occurrence in T or both P and the portion of T spanned by occurrences of P are periodic (with the same period). In 2019, Bringmann et al. [9] showed that either P has *few* approximate occurrences (under the Hamming distance) or it is *approximately periodic*. Later, Charalampopoulos et al. [16] tightened this result and proved an analogous statement for approximate occurrences under the edit distance.

Let us now focus on approximate circular pattern matching. The CPM problem under the Hamming distance is called the *k-Mismatch CPM* problem. An $\mathcal{O}(nk)$-time algorithm and an $\tilde{\mathcal{O}}(n + (n/m)k^3)$-time algorithm were proposed for the reporting version of k-Mismatch CPM by Charalampopoulos et al. in [13] and [15], respectively, whereas an $\tilde{\mathcal{O}}(n + (n/m)k^2)$-time algorithm for its decision version was given in [15]. Further, the authors of [7, 26] presented efficient average-case algorithms for k-Mismatch CPM. The k-Edit CPM problem was considered in [15], where an $\mathcal{O}(nk^2)$-time algorithm and an $\mathcal{O}(nk \log^3 k)$-time algorithm were presented for the reporting and decision version, respectively. Until now, no algorithm with worst-case runtime $\mathcal{O}(n + (n/m)k^{\mathcal{O}(1)})$ was known for k-Edit CPM. Such an algorithm is superior over $\mathcal{O}(nk^{\mathcal{O}(1)})$-time algorithms when the number of allowed errors is small in comparison to the length of the pattern. Here, we propose the first such algorithms.

Our result. In order to represent the output of our algorithm compactly, we need the notion of an *interval chain*. For two integer sets A and B, let $A \oplus B = \{a + b : a \in A, b \in B\}$. We extend this notation for an integer b to $A \oplus b = b \oplus A = A \oplus \{b\}$. An interval chain for an interval I and non-negative integers a and q is a set of the form

$$\text{Chain}(I, a, q) = I \cup (I \oplus q) \cup (I \oplus 2q) \cup \cdots \cup (I \oplus aq).$$

Here q is called the *difference* of the interval chain. For example the set of underlined intervals in Figure 4 corresponds to $\text{Chain}([3\,..\,8], 2, 8) = [3\,..\,8] \cup [11\,..\,16] \cup [19\,..\,24]$.

Our main algorithmic result can be stated as follows (cf. Table 1).

[1] Throughout this work, the $\tilde{\mathcal{O}}(\cdot)$ notation hides factors polylogarithmic in the length of the input strings.

Table 1 The upper-bound landscape of pattern matching (PM) and circular PM (CPM) with k edits. In the decision version of k-Edit CPM, the algorithms only find if there exists at least one occurrence and return a witness; otherwise the algorithms report all the occurrences.

k-Edit PM	Reference	Note	k-Edit CPM	Reference	Note
$\mathcal{O}(n^2)$	[38]	for any k	$\mathcal{O}(nk^2)$	[15]	
$\mathcal{O}(nk^2)$	[32]		$\tilde{\mathcal{O}}(nk)$	[15]	decision
$\mathcal{O}(nk)$	[33]		$\mathcal{O}(n + k^6 \cdot n/m)$	**This work**	
$\tilde{\mathcal{O}}(n + k^{\frac{25}{3}} \cdot n/m^{\frac{1}{3}})$	[37]		$\tilde{\mathcal{O}}(n + k^5 \cdot n/m)$	**This work**	decision
$\mathcal{O}(n + k^4 \cdot n/m)$	[20]				
$\tilde{\mathcal{O}}(n + k^{3.5} \cdot n/m)$	[17]				

▶ **Theorem 1.** *The reporting version of the k-Edit CPM problem can be solved in $\mathcal{O}(n + (n/m)k^6)$ time, with the output represented as a union of $\mathcal{O}((n/m)k^6)$ interval chains. The decision version of the k-Edit CPM problem can be solved in $\mathcal{O}(n + (n/m)k^5 \log^3 k)$ time.*

The following notion of an *anchor* (see also Figure 2) is crucial for understanding the structure of (approximate) circular pattern matching.

▶ **Definition 2.** *A circular k-edit occurrence $T[p\,..\,p']$ of P is* anchored *at position i (called* ***anchor****) if $\delta_E(T[p\,..\,i), Y) + \delta_E(T[i\,..\,p'], X) \leq k$, where $P = XY$ for some X, Y. We denote*

$$\texttt{Anchored}_k(P, T, i) = \{\, p : T[p\,..\,p'] \text{ is anchored at } i \text{ for some } p' \,\}.$$

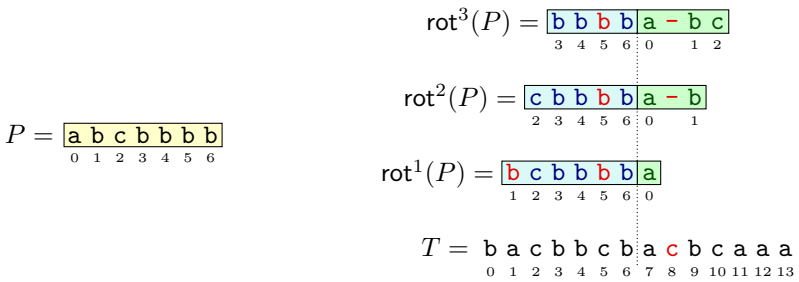

Figure 2 The starting positions of circular 2-edit occurrences of pattern P anchored at position 7 in text T are $\texttt{Anchored}_2(P, T, 7) = \{0, 1, 2, 3, 4\}$; the occurrences at positions $1, 2, 3$ are shown.

▶ **Example 3.** Let $P = \mathtt{a}^{99}\,\mathtt{b}$ and $T = P^2$. Then $|\mathsf{CircOcc}_0(P, T)| = 101$, while we have only two anchors (0 and 100).

Our algorithm exploits the approximate periodic structure of the two strings in scope. On the way to our main algorithmic result we prove (in the end of Section 2) the following structural result for k-Edit CPM:

▶ **Theorem 4.** *Consider a pattern P of length m, a positive integer threshold k, and a text T of length $n \leq cm + k$, for a constant $c \geq 1$. Then, either there are only $\mathcal{O}(k^2)$ anchors of circular k-edit occurrences of P in T or some rotation of P is at edit distance $\mathcal{O}(k)$ from a string with period $\mathcal{O}(m/k)$.*

The PILLAR model. We work in the PILLAR model that was introduced in [16] with the aim of unifying approximate pattern matching algorithms across different settings. In this model, we assume that the following primitive PILLAR operations can be performed efficiently, where the argument strings are fragments of strings in a given collection $\mathcal{X}$:

- $\mathsf{Extract}(S, \ell, r)$: Retrieve string $S[\ell\,..\,r)$.
- $\mathsf{LCP}(S, T)$, $\mathsf{LCP}_R(S, T)$: Compute the length of the longest common prefix/suffix of S, T.
- $\mathsf{IPM}(S, T)$: Assuming that $|T| \leq 2|S|$, compute the starting positions of all exact occurrences of S in T, expressed as an arithmetic progression.
- $\mathsf{Access}(S, i)$: Retrieve the letter $S[i]$; $\mathsf{Length}(S)$: Compute the length $|S|$ of the string S.

The runtime of algorithms in this model can be expressed in terms of the number of primitive PILLAR operations. The result underlying Theorem 1 can be stated as follows.

▶ **Theorem 5.** *If $n \leq m \leq 2n$, the reporting and decision versions of the k-Edit CPM problem can be solved in $\mathcal{O}(k^6)$ time and $\mathcal{O}(k^5 \log^3 k)$ time in the* PILLAR *model, respectively.*

Theorem 5 implies Theorem 1 as well as efficient algorithms for k-Edit CPM in internal, dynamic, fully compressed, and quantum settings based on known implementations of the PILLAR model in these settings, as discussed in Appendix C.

Our approach. Every circular k-edit occurrence of P in T is anchored at some position i of T. In the reporting and decision version of the problem, we use the following respective results.

▶ **Lemma 6** ([14, Lemma 30])**.** *Given a text T of length n, a pattern P of length m, an integer $k > 0$, and a position i of T, we can compute in $\mathcal{O}(k^2)$ time in the* PILLAR *model the set $\mathtt{Anchored}_k(P, T, i)$, represented as a union of $\mathcal{O}(k^2)$ intervals, possibly with duplicates.*

For an interval I denote by $\mathtt{AnyAnchored}_k(P, T, I)$ an arbitrarily chosen position in the set $\bigcup_{i \in I} \mathtt{Anchored}_k(P, T, i)$; if this set is empty then the result is *none*.

▶ **Lemma 7** ([15, Section 4])**.** *Given a text T of length n, a pattern P of length m, an integer $k > 0$, and an interval I containing up to k positions of T, we can compute $\mathtt{AnyAnchored}_k(P, T, I)$ in $\mathcal{O}(k^2 \log^3 k)$ time in the* PILLAR *model.*

It will be convenient and sufficient to deal separately with fragments of T of length $\mathcal{O}(m)$, so we can assume w.l.o.g. that $n = \mathcal{O}(m)$. Let $P = P_1P_2$ be a decomposition of the pattern with $|P_1| = \lfloor m/2 \rfloor$. By using Lemma 6 to compute k-edit circular occurrences that are anchored at one of $\mathcal{O}(k^2)$ carefully chosen anchors, we reduce our problem to searching for k-edit (non-circular) occurrences of any length-m substring of a certain fragment V of $P_2P_1P_2$ in a suitable fragment U of T, where both V and U are approximately periodic (there is also a symmetric case where V is a substring of $P_1P_2P_1$).

We achieve this as follows. Let us denote the set of standard (non-circular) *k-edit occurrences* of a string X in a string Y by

$$\mathsf{Occ}_k(X, Y) = \{i \in [0\,..\,|Y|) : Y[i\,..\,i'] =_k X \text{ for some } i' \geq i\}.$$

We compute the set $\mathsf{Occ}_k(P_1, T)$ using an algorithm for pattern matching with k edits [16]. If this set is small, it yields a small set of *anchors* for k-edit occurrences of rotations of P that contain P_1. We also do the same for P_2. Then, we can apply Lemma 6 to each anchor.

The challenging case is when $\mathsf{Occ}_k(P_1, T)$ is large. The structural result for k-Edit PM then implies that P_1 and the portions of T spanned by approximate occurrences of P_1 are *almost periodic*, i.e., they are at small edit distance from a substring of string Q^∞, where

Q is a short string. We extend the periodicity in each of $P_2P_1P_2$ and T, allowing for more edits. The reduction is then completed by accounting for some technical considerations and, possibly, calling Lemma 6 $\mathcal{O}(k^2)$ more times.

In order to develop some intuition for how to deal with the almost periodic case, let us briefly discuss how it is dealt with in the case where we are looking for approximate (circular) occurrences under the Hamming distance. The mismatches of each of the two strings (P and T or U and V) with a substring of Q^∞ are called *misperiods*. Now, consider some candidate starting position i of P in T, assuming that both $P[0\,..\,|Q|)$ and $T[i\,..\,i+|Q|)$ are approximate copies of Q: the number of mismatches of P and $T[i\,..\,i+m)$ can be inferred by just looking at the misperiods: it is just the total number of misperiods in P and $T[i\,..\,i+m)$ minus the misperiods that are aligned and thus "cancel out".

For approximate PM under the edit distance, the situation is much more complicated as deletions and insertions can be applied, and hence we cannot have an analogous statement about misperiods "cancelling out". Following works on (non-circular) k-edit PM, we employ so-called *locked* fragments (see [16, 20]).

Roughly speaking, we partition each of U and V into locked fragments and powers of Q, such that the total length of locked fragments is small and, if a locked fragment is to be aligned with a substring of Q^∞, we would rather align it with a power of Q. Then, intuitively, one has to overcome technical challenges arising from the nature of the overlap of the locked fragments with a specific circular k-edit occurrence.

We consider different cases depending on whether the fragments of U and V that yield a match imply that any pair of locked fragments (one in U and one in V) overlap. A crucial observation is that, roughly speaking, as we slide a length-m fragment of V over U, $|Q|$ positions at a time, such that the locked fragments in the window in U remain unchanged and do not overlap with locked fragments in V, the edit distance remains unchanged.

2 Reduction of k-Edit CPM to the PeriodicSubMatch Problem

A *string* $S = S[0\,..\,|S|-1]$ is a sequence of *letters* over some alphabet. The string $S[i]S[i+1]\cdots S[j]$, for any indices i, j such that $i \le j$, is called a *substring* of S. By $S[i\,..\,j] = S[i\,..\,j+1) = S(i-1\,..\,j]$ we denote a *fragment* of S that can be viewed as a positioned substring $S[i]S[i+1]\cdots S[j]$ (it is represented in $\mathcal{O}(1)$ space). We also denote $S^{(j)} = S[j\,..\,j+m)$. An integer p such that $0 < p \le |S|$ is called a *period* of S if $S[i] = S[i+p]$, for all $i \in [0\,..\,|S|-p)$. We define *the period* of S as the smallest such p. A string Q is called *primitive* if $Q = W^k$ for a string W and a positive integer k implies that $k = 1$. By $\mathsf{rot}^j(X)$ we denote the string $X[j\,..\,|X|)X[0\,..\,j)$. We generalize the rotation operation rot to arbitrary integer exponents r as $\mathsf{rot}^r(X) = \mathsf{rot}^{r \bmod |X|}(X)$.

By $\delta_E(X, Y^*)$, $\delta_E(X, {}^*Y)$ and $\delta_E(X, {}^*Y^*)$ we denote the minimum edit distance between string X and any prefix, suffix and substring of string $Y^{|X|+|Y|}$, respectively.

We say that a string U is *almost Q-periodic* if $\delta_E(U, Q^*) \le 112k$. We write $a \equiv_d b \pmod q$ if $a - b \equiv i \pmod q$, where $\min(i, q-i) \le d$ (in other words, a and b are d-approximately congruent modulo q). For example, $11 \equiv_3 21 \pmod 8$, but $11 \equiv_1 21 \pmod 8$ does not hold.

A pair of indices (p, x) satisfying $p \in \mathsf{Occ}_k(V^{(x)}, U)$ and $p \equiv_{77k} x + r \pmod q$ will be called an approximate match (*app-match*, in short).

The following auxiliary problem, PeriodicSubMatch, is illustrated in Figure 3.

PeriodicSubMatch

Input: A primitive string Q, integers m, r, k, α, β, and strings U, V such that

- $m \le |U| \le \frac{7}{4}m + 3(k+1)$, $m \le |V| \le \frac{3}{2}m$, $q = |Q| \le \frac{m}{256\,k}$, $r \in [0\,..\,q)$,
- U is almost Q-periodic,
- $V = P^2[\alpha\,..\,\beta]$ (hence, length-m substrings of V are rotations of P),
- V is almost Q'-periodic, where $Q' := \mathsf{rot}^r(Q)$.

Output: $\{\, p \in \mathsf{Occ}_k(V^{(x)}, U) \;:\; p \equiv_{77k} x + r \pmod q,\ x \le |V| - m \,\}$.

▶ Remark 8. Due to the condition that V is a fragment of P^2, we can apply the operation $\texttt{Anchored}_k$ to compute efficiently the output of PeriodicSubMatch in the case when a position j_1 in V is *aligned* with a position i_1 in U. The efficiency of the whole approach is based on the efficiency of the operation $\texttt{Anchored}_k$.

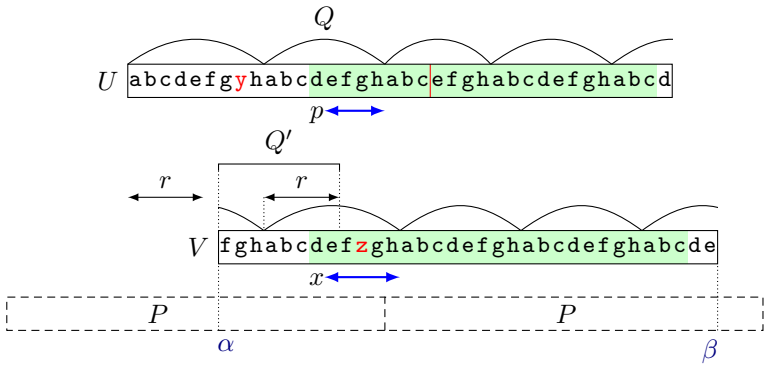

Figure 3 We have $m = 25$, $k = 2$ and $r = 5$. Edits with respect to the approximate periodicity are marked in red. Green rectangles show that $V^{(x)} =_2 U[p\,..\,p+23)$. We have $p = x + r + 1$, so $p \equiv_1 x + r \pmod q$. The distances (in blue) from p and x to the starts of next approximate periods Q are the same up to $\Theta(k)$. For the example purposes, we waive the constraint $q = |Q| \le \frac{m}{256\,k}$.

The strings U and V are both close to substrings of Q^∞. The condition $p \equiv_{\Theta(k)} x + r \pmod q$ means that we are only interested in k-edit occurrences $U[p\,..\,p']$ of $V^{(x)}$ such that the two substrings are approximately synchronized with respect to the approximate period Q; see Figure 3. (In particular, no other k-edit occurrences exist.) The constants originate from Theorem 10 and some additional requirements imposed in the proof of Lemma 12.

▶ **Example 9.** A very simple *double fully periodic* case, where both U and V are substrings of Q^∞, is depicted in Figure 4. Again, we waive the constraint $q = |Q| \le \frac{m}{256\,k}$.

The following theorem follows as a combination of several results of [16], see Appendix A.

▶ **Theorem 10** ([16]). *If $|T| = n < \frac{3}{2}m + k$, then in $\mathcal{O}(k^4)$ time in the* PILLAR *model we can compute a representation of the set $\mathsf{Occ}_k(P, T)$. If $\lfloor |\mathsf{Occ}_k(P, T)|/k \rfloor > 642045 \cdot (n/m) \cdot k$, the algorithm also returns:*

- *a primitive string Q satisfying $|Q| \le m/(256k)$, $\delta_E(P, {}^*Q^*) = \delta_E(P, Q^*) < 2k$, and*
- *a fragment $\bar{T}$ of T such that $\delta_E(\bar{T}, {}^*Q^*) \le \delta_E(\bar{T}, Q^*) \le 24k$, $|\mathsf{Occ}_k(P, T)| = |\mathsf{Occ}_k(P, \bar{T})|$.*

Moreover, $i \equiv_{24k} 0 \pmod{|Q|}$ for each $i \in \mathsf{Occ}_k(P, \bar{T})$.

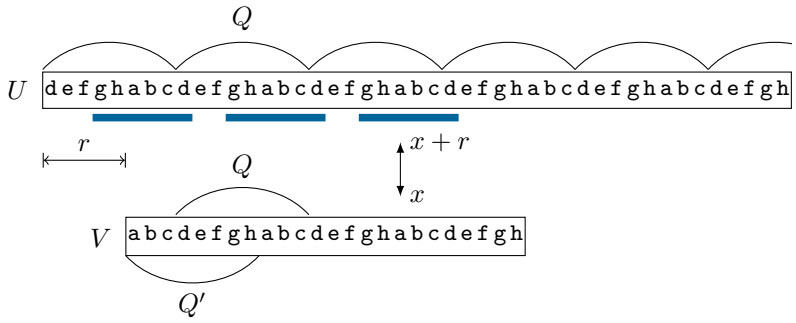

Figure 4 A double fully periodic case. Let $k = 2$, $q = |Q| = 8$, and $r = 4$. For $m = 23$, the set of k-edit occurrences of any length-m fragment of V (2 possibilities) in U is the (underlined) interval chain. For $m = 16$ it is a single interval. Position x in V is synchronized with respect to the periodicity with any position p in U such that $p \equiv x + r \pmod{q}$.

▶ Remark 11. An $\mathcal{O}((n/m)k^{3.5}\sqrt{\log k \log m})$-time algorithm for computing a representation of the set $\mathsf{Occ}_k(P, T)$ using $\mathcal{O}(k^3)$ arithmetic progressions was presented in [17]. The simpler result from [16] is sufficient for our needs.

▶ **Lemma 12.** *If $n = \mathcal{O}(m)$, then k-Edit CPM can be reduced in $\mathcal{O}(k^4)$ time in the* PILLAR *model to at most two instances of the* PeriodicSubMatch *problem. The output to k-Edit CPM is a union of the outputs of the two* PeriodicSubMatch *instances and $\mathcal{O}(k^4)$ intervals.*

Sketch of the proof. The proof resembles the proof of [15, Lemma 12] for Hamming distance. Let us partition P to two (roughly) equal chunks, P_1 of length $\lfloor m/2 \rfloor$ and P_2 of length $\lceil m/2 \rceil$. Each circular k-edit occurrence of P in T implies a standard k-edit occurrence of at least one of P_1 and P_2. We focus on the case when it implies such an occurrence of P_1, noting that the computations for P_2 are symmetric.

For a fragment T' of T, we denote by $\mathsf{Implied}_k(P_1, T')$ the set of circular k-edit occurrences of P in T in which a k-edit occurrence of P_1 is contained in T'.

We cover T with fragments of length $\lfloor \frac{3}{2}|P_1| \rfloor + k$ starting at multiples of $\lfloor \frac{1}{2}|P_1| \rfloor$. (The last fragments can be shorter.) For each of the fragments T' of T, we will compute a representation of a set A such that $\mathsf{Implied}_k(P_1, T') \subseteq A \subseteq \mathsf{CircOcc}_k(P, T)$. If $|\mathsf{Occ}_k(P_1, T')| = \mathcal{O}(k^2)$, we use the following fact whose proof is based on anchors.

▷ Claim 13. If the set $\mathsf{Occ}_k(P_1, T')$ for a fragment T' of T has size $\mathcal{O}(k^2)$ and is given, then a set of positions A such that $\mathsf{Implied}_k(P_1, T') \subseteq A \subseteq \mathsf{CircOcc}_k(P, T)$, represented as a union of $\mathcal{O}(k^4)$ intervals, can be computed in $\mathcal{O}(k^4)$ time in the PILLAR model.

Proof. We compute the set $\mathtt{Anchored}_k(P, T, i + s)$ for each position $i \in \mathsf{Occ}_k(P_1, T')$, where s is the starting position of T' in T (i.e., $T' = T[s \,..\, s + |T'|)$). By Lemma 6, this set is represented as a union of $\mathcal{O}(k^2)$ intervals and can be computed in $\mathcal{O}(k^2)$ time in the PILLAR model. Since $|\mathsf{Occ}_k(P_1, T')| = \mathcal{O}(k^2)$, the union A of all these sets contains $\mathcal{O}(k^4)$ intervals and is computed in $\mathcal{O}(k^4)$ total time. Clearly, A satisfies the required inclusions. ◁

If $|\mathsf{Occ}_k(P_1, T')| = \mathcal{O}(k^2)$, Claim 13 can be applied. Otherwise, we can assume that $|\mathsf{Occ}_k(P_1, T')| > ck^2$ for a sufficiently large constant c. Then, by Theorem 10, P_1 and the relevant part $\bar{T}'$ of T' are both almost Q-periodic.

Computing V. String V is obtained by extending the approximate periodicity of the middle fragment P_1 in $P_2P_1P_2$ towards both directions. (Note that all rotations of P that contain its first half P_1 are substrings of $P_2P_1P_2$.) In each direction, we stop extending when either $c'k$ errors to a prefix (suffix) of $Q^{|V|}$ are accumulated, for a specified constant c', or we reach the end of the string. In the former case, we obtain a so-called repetitive region R_{right} with a prefix P_1 (R_{left} with a suffix P_1, respectively); see Figure 5.

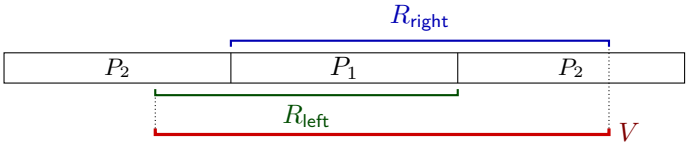

Figure 5 String V (shown in brown) and repetitive regions R_{left} and R_{right} in $P_2P_1P_2$.

Intuitively, a repetitive region is a fragment that is sufficiently long and almost periodic, but also sufficiently far from being periodic. Thus a rotation of P that contains P_1 either contains one of the repetitive regions or it is contained in V. A repetitive region is known [16] to have $\mathcal{O}(k^2)$ occurrences in a string of length $\mathcal{O}(m)$, so the former case can be solved in $\mathcal{O}(k^4)$ time with the aid of anchors as in Claim 13. The latter case will lead to PeriodicSubMatch.

Computing U. We obtain U by extending the approximate periodicity of $\bar{T}'$ to T towards both directions. We extend it to the left until one of the following conditions is satisfied: the appended substring is at edit distance at least $c'k$ from all suffixes of $(\mathsf{rot}^x(Q))^{2n}$, for all $x \in [-34k\,..\,34k]$, the beginning of T is reached, or roughly $m/2 + k$ letters have been inspected.

The extension to the left is symmetric. Finally, we prove that all occurrences in $\mathsf{Implied}_k(P_1, T')$ that correspond to length-m substrings of V are contained in U. The approximate congruence mod$|Q|$ in PeriodicSubMatch follows from the analogous condition in Theorem 10. ◀

Let us now restate and prove our structural result.

▶ **Theorem 4.** *Consider a pattern P of length m, a positive integer threshold k, and a text T of length $n \leq cm + k$, for a constant $c \geq 1$. Then, either there are only $\mathcal{O}(k^2)$ anchors of circular k-edit occurrences of P in T or some rotation of P is at edit distance $\mathcal{O}(k)$ from a string with period $\mathcal{O}(m/k)$.*

Proof. Theorem 4 readily follows from the proof of Lemma 12. If $|V| \geq m$, then some rotation of P is almost periodic. Otherwise, we only have $\mathcal{O}(k^2)$ anchors for approximate circular occurrences (stemming from occurrences of some of P_1, P_2, or a repetitive region obtained by extending either of P_1 or P_2 in some direction). ◀

3 Locked Fragments

The notion of locked fragments originates from [20]. We use them as defined in [16]. Let us state [16, Lemma 6.9] [2] with $d_S = 112k$, for $k > 0$; this characterization of locked fragments will be sufficient for our purposes. See Figure 6 for an illustration.

[2] The original lemma also concluded that L_1 is a so-called k-locked prefix; however, this property is not needed here (and, in particular, a k-locked string is also locked).

▶ **Lemma 14** (see [16, Lemmas 5.6 and 6.9]). *Let S denote a string, Q denote a primitive string, $q = |Q|$, and suppose that $\delta_E(S, {}^*Q^*) \leq 112k$ and $|S| \geq 225kq$ for some positive integer k.*

Then there is an algorithm which in $\mathcal{O}(k^2)$ time in the PILLAR *model computes disjoint* locked fragments *$L_1, \ldots, L_\ell$ of S satisfying:*

(a) *$S = L_1 Q^{\alpha_1} L_2 Q^{\alpha_2} \cdots L_{\ell-1} Q^{\alpha_{\ell-1}} L_\ell$, where $\alpha_i \in \mathbb{Z}_{>0}$ for all i,*

(b) *$\delta_E(S, {}^*Q^*) = \sum_{i=1}^{\ell} \delta_E(L_i, {}^*Q^*)$ and $\delta_E(L_i, {}^*Q^*) > 0$ for all $i \in (1\,..\,\ell)$,*

(c) *$\ell = \mathcal{O}(k)$ and $\sum_{i=1}^{\ell} |L_i| \leq 676kq$.*

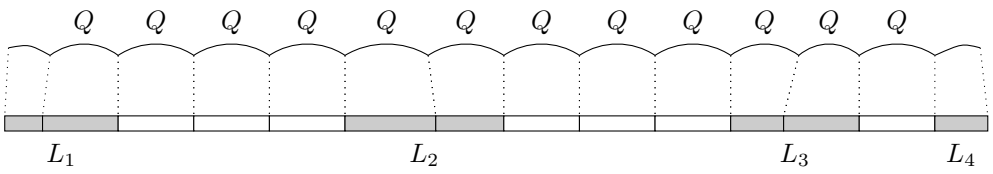

Figure 6 Illustration of Lemma 14. We have a decomposition $S = L_1 \cdot Q^3 \cdot L_2 \cdot Q^3 \cdot L_3 \cdot Q^1 \cdot L_4$. L_1 is an approximate suffix of $Q^{|S|}$, L_4 is an approximate prefix of Q^∞, and internal gray parts are *approximate* powers of Q. The remaining (white) fragments are *exact* powers of Q.

Let us consider the decompositions obtained by applying Lemma 14 to strings U and V from PERIODICSUBMATCH w.r.t. the string Q. Strings U and V are almost Q-periodic and almost Q'-periodic, respectively, so $\delta_E(U, {}^*Q^*), \delta_E(V, {}^*Q^*) \leq 112k$. Moreover, $|U|, |V| \geq m \geq 256kq > 225kq$. Thus, U and V satisfy the assumptions of the lemma. If any of the decompositions starts with a locked prefix of length smaller than q (possibly empty) or ends with a locked suffix of length smaller than q, we extend the locked fragment by a copy of Q and possibly by a neighbouring locked fragment if this copy was the only copy separating them. The total length of the locked fragments increases by at most $2q \leq 2kq$, so it is bounded by $678kq$.

4 Overlap Case of PERIODICSUBMATCH

We consider all possible *offsets* Δ (integers $\Delta \in (-|V|\,..\,|U|)$) by which we can shift V, looking for a length-m substring of V that approximately matches a substring of U.

We denote $\mathsf{Ext}_t(X) = \bigcup_{x \in X} \{y : |x - y| \leq t\}$. Denote also by $\mathit{locked}(U)$, $\mathit{locked}(V)$ the set of positions in all locked fragments in U, V, respectively.

▶ **Definition 15.** *Δ is a* t-overlap offset *if there are positions p, x such that $p - x = \Delta$, and*

$$p \in X \oplus \{-m, 0, m\}, \quad x \in Y \oplus \{-m, 0, m\} \quad \text{where } X = \mathsf{Ext}_t(\mathit{locked}(U)),\ Y = \mathit{locked}(V).$$

Otherwise Δ is a t-non-overlap offset.

An integer Δ is called a *valid offset* if $\Delta \equiv_{77k} r \pmod q$. (Recall the definition of r in PERIODICSUBMATCH.) For two integer sets A and B, let $A \ominus B = \{a - b : a \in A, b \in B\}$.

▶ **Observation 16.** *For any intervals I, J, the set $\mathsf{Ext}_t(I \ominus J)$ is an interval of size $|I| + |J| - 1 + 2t$ that can be computed in $\mathcal{O}(1)$ time.*

▶ **Lemma 17.** *The set of valid t-overlap offsets can be represented as a union of $\mathcal{O}(k^2 + k^2 t/q)$ intervals of length $\mathcal{O}(k)$ each. This representation can be computed in $\mathcal{O}(k^2 + k^2 t/q)$ time in the* PILLAR *model.*

Proof. Let $\ell_1, \ldots, \ell_{n_1}$ and $\ell'_1, \ldots, \ell'_{n_2}$ be the lengths of locked fragments in U and V, respectively, and $s_1 = \sum_{i=1}^{n_1} \ell_i$, $s_2 = \sum_{i=1}^{n_2} \ell'_i$. By point (c) in Lemma 14, we have $n_1 + n_2 = \mathcal{O}(k)$ and $s_1 + s_2 = \mathcal{O}(kq)$. By Observation 16, the set of t-overlap offsets is a union of $\mathcal{O}(k^2)$ intervals of total length proportional to:

$$\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}(\ell_i+\ell'_j+t) = n_1n_2t+n_2\sum_{i=1}^{n_1}\ell_i+n_1\sum_{j=1}^{n_2}\ell'_j \leq n_1n_2t+(n_1+n_2)(s_1+s_2) = \mathcal{O}(k^2(t+q)).$$

The intervals can be computed in $\mathcal{O}(k^2)$ time. An interval of length ℓ contains $\mathcal{O}(k + \ell k/q)$ valid offsets grouped into $\mathcal{O}(1 + \ell/q)$ intervals of length $\mathcal{O}(k)$ each. These maximal intervals of offsets can be computed in $\mathcal{O}(1+\ell/q)$ time via elementary modular arithmetics. Therefore, the number of intervals of t-overlap offsets that are valid is proportional to

$$(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} 1) + \mathcal{O}(k^2 + k^2t/q) = \mathcal{O}(k^2 + k^2t/q)$$

and all of them can be computed in $\mathcal{O}(k^2 + k^2t/q)$ time. ◀

An app-match (p, x) is called a ***t*-overlap app-match** if and only if $p - x$ is a t-overlap offset. In this section, we consider t-overlap app-matches. In Section 5, we consider t-non-overlap app-matches: app-matches (p, x) such that $p - x$ is a t-non-overlap offset, for $t = \Theta(qk)$.

It follows from the statement of PeriodicSubMatch that if (p, x) is an app-match, then $p - x$ is a valid offset. The following fact, together with Lemma 6, implies a fast algorithm for computing the following set for a given offset Δ:

$$\{p \in \mathsf{Occ}_k(V^{(x)}, U) \,:\, \Delta = p - x,\, \Delta \equiv_{77k} r \pmod q\}.$$

▶ **Fact 18.** *If (p, x) is an app-match, $\Delta = p - x$ and $\Delta' = m - \alpha + \Delta$, then the corresponding circular k-edit occurrence $U[p\,..\,p']$ is anchored at a position in $[\Delta' - k\,..\,\Delta' + k]$; see Figure 7.*

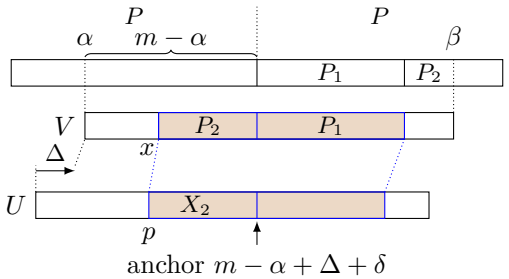

Figure 7 The anchor in U is at position $m - \alpha + \Delta + \delta$, where $\delta = |X_2| - |P_2| \in [-k\,..\,k]$ (since $\delta_E(X_2, P_2) \leq k$).

Using Lemmas 6 and 7 we obtain the following corollary.

▶ **Corollary 19.** *Let I be an interval of size $\mathcal{O}(k)$. All positions p for which there exists an app-match (p, x) such that $p - x \in I$, represented as a union of $\mathcal{O}(k^3)$ intervals, can be computed in $\mathcal{O}(k^3)$ time in the* PILLAR *model. Moreover, one can check if there is any app-match (p, x) with $p - x \in I$ in $\mathcal{O}(k^2 \log^3 k)$ time in the* PILLAR *model.*

The solution of the overlap case is presented in Algorithm 1. Lemma 17 together with Fact 18 and Corollary 19 imply the following lemma.

Algorithm 1 Overlap case: reporting version.

Compute the decompositions of U and V into locked fragments;

// Compute the set Λ of $(t+k)$-overlap offsets, being a union of $\mathcal{O}(k^2)$ intervals:
foreach *locked fragment* $U[i_{\min}\,..\,i_{\max}]$ **do**
  foreach *locked fragment* $V[j_{\min}\,..\,j_{\max}]$ **do**
    $\Lambda := \Lambda \cup ([i_{\min} - j_{\max} - (t+k)\,..\,i_{\max} - j_{\min} + (t+k)] \oplus \{-m, 0, m\})$;

// Compute the set Γ of valid $(t+k)$-overlap offsets,
// represented as a union of $\mathcal{O}(k^2 + k^2(t+k)/q)$ intervals of size $\mathcal{O}(k)$ each:
foreach *interval* I *of offsets* **in** Λ **do**
  $\Gamma := \Gamma \cup \{$maximal intervals representing $\{i \in I : i \equiv_{77k} r \pmod q\}\}$;

foreach *interval* $[i_{\min}\,..\,i_{\max}]$ *of offsets* **in** Γ, *with* $i_{\max} - i_{\min} = \mathcal{O}(k)$ **do**
  $J := [i_{\min}\,..\,i_{\max}] \oplus (m - \alpha)$;
  report $\bigcup_{a \in J} \texttt{Anchored}_k(P, U, a)$;

▶ **Lemma 20.** *Let B be the output of Algorithm 1. Then $B \subseteq \mathsf{CircOcc}_k(P, U)$ and every t-overlap app-match occurrence p is in B.*

Moreover, if $t = \mathcal{O}(kq)$, Algorithm 1 works in $\mathcal{O}(k^6)$ time in the PILLAR *model with the output represented as a union of $\mathcal{O}(k^6)$ intervals.*

Proof. Consider a t-overlap app-match (p, x). Then, there exists an anchor a such that $p \in \texttt{Anchored}_k(P, U, a)$, and $y = a - (m - \alpha)$ is a $(t+k)$-overlap offset, since we have

$$\delta_E(V[x\,..\,m-\alpha), U[p\,..\,a)) + \delta_E(V[m-\alpha\,..\,x+m), U[a\,..\,p']) \le k.$$

Now, y is in some interval $[i_{\min}\,..\,i_{\max}] \in \Gamma$, as the union of the elements of Γ comprises the set of valid $(t+k)$-overlap offsets. Then, since $y \in [i_{\min}\,..\,i_{\max}]$, we have $a = y + (m - \alpha) \in [i_{\min}\,..\,i_{\max}] \oplus (m - \alpha)$, and hence a is in one of the sets J constructed in the penultimate line of Algorithm 1. In the case when $t = \mathcal{O}(kq)$, using Lemma 17, we compute, in $\mathcal{O}(k^3)$ time, $\mathcal{O}(k^3)$ intervals of anchors, of size $\mathcal{O}(k)$ each. The time complexity and the fact that the algorithm returns the output as a union of $\mathcal{O}(k^6)$ intervals follows by a direct application of Corollary 19 to each interval of anchors. ◀

To obtain the next corollary, we replace the last line of Algorithm 1 by:

if $\texttt{AnyAnchored}_k(P, U, J) \neq$ *none* **then return** $\texttt{AnyAnchored}_k(P, U, J)$;

▶ **Corollary 21.** *If $t = \mathcal{O}(kq)$, one can check if $B \neq \emptyset$ and, if so, return an arbitrary element of B, in $\mathcal{O}(k^5 \log^3 k)$ time in the* PILLAR *model.*

5 Non-Overlap Case of PeriodicSubMatch

Recall that an app-match (p, x) is called a *t-non-overlap app-match* if and only if $p - x$ is a t-non-overlap offset. In this section we assume $t = \Theta(kq)$. The set of t-non-overlap offsets is too large, but it has a short representation.

▶ **Lemma 22.** *The set of t-non-overlap offsets can be partitioned into $\mathcal{O}(k^2)$ maximal intervals in $\mathcal{O}(k^2 \log\log k)$ time in the* PILLAR *model.*

Proof. There are $\mathcal{O}(k)$ locked fragments in U and V. By Observation 16, every pair of locked fragments, one from U and one from V, induces an interval of t-overlap offsets that can be computed in $\mathcal{O}(1)$ time. The complement of the union of these offsets can be computed in $\mathcal{O}(k^2 \log\log k)$ time by sorting the endpoints of the intervals using integer sorting [24]. ◀

We denote by $\mathbf{NonOv}(t)$ the set of maximal intervals yielded by the above lemma. For simplicity, we mostly discuss the decision version of the problem in this section; the correctness proof for the reporting version requires a few further technical arguments.

Let $\lambda_k = (112k+3)\cdot(3k+10)\cdot q + 678kq$.

▶ **Lemma 23.** *If $\lambda_k > \frac{m}{2}$,* PeriodicSubMatch *can be solved in $\mathcal{O}(k^5)$ time in the* PILLAR *model, with the output represented as a union of $\mathcal{O}(k^5)$ intervals.*

Proof. We have $m = \mathcal{O}(k^2 q)$. As $\mathcal{O}(k)$ out of every q consecutive offsets are valid and they can be grouped in at most two intervals, there are $\mathcal{O}(mk/q) = \mathcal{O}(k^3)$ valid offsets, which are grouped into $\mathcal{O}(k^2)$ intervals of size $\mathcal{O}(k)$ each. Let the set of such intervals be $\mathcal{J}$. The time complexity and output size follow from an application of Corollary 19 to the $\mathcal{O}(k)$-size interval of anchors corresponding to each $J \in \mathcal{J}$, as in the last three lines of Algorithm 1. ◀

Henceforth we assume that $\lambda_k \le \frac{m}{2}$. Let W be the longest fragment of V such that each length-m fragment of V contains W, i.e., $W = V[|V| - m \,..\, m)$.

▶ **Observation 24.** *If $\lambda_k \le \frac{m}{2}$, then W contains a fragment equal Q^{3k+9} that is disjoint from locked fragments in V.*

Proof. We have $|W| \ge \frac{m}{2}$ since $|V| \le \frac{3}{2}m$. By Lemma 14, V contains at most $112k+2$ locked fragments. Their total length does not exceed $678kq$. By the pigeonhole principle, as $\lambda_k = (112k+3)\cdot((3k+10)\cdot q) + 678kq \le |W|$, string W contains a substring of length at least $(3k+10)q$ that is disjoint from locked fragments. By Lemma 14, this substring is a substring of Q^∞ and thus contains a copy of Q^{3k+9}. ◀

▶ **Definition 25** (sample)**.** *We select an arbitrary fragment $V[j \,..\, j']$ equal Q^{3k+9} of W that is disjoint from locked fragments in V; then the middle fragment $V[j_1 \,..\, j_2]$ of $V[j \,..\, j']$ equal Q^{k+1} becomes an additional locked fragment. The fragment $V[j_1 \,..\, j_2]$ is called the* sample*.*

When computing t-overlap offsets with the algorithm of Section 4, we treat the sample as a locked fragment; the total length of the locked fragments is then still $\mathcal{O}(kq)$.

Henceforth we replace P by its rotation $\mathsf{rot}^y(P)$, where $y = (j_1 + \alpha) \bmod m$. Let us note that after this change, the sets $\texttt{Anchored}_k$ can be computed equally efficiently as the sets $\texttt{Anchored}_k$ for the original P. This follows from the fact that the algorithm underlying Lemma 6 does not use IPM queries, and the remaining queries from the PILLAR model can easily be implemented in $\mathcal{O}(1)$ time if an input string is given by its cyclic rotation.

For an interval $I = [i_1 \,..\, i_2]$ and a string S, by $S[I]$ we denote $S[i_1 \,..\, i_2]$. We denote $\hat{q} = 2(k+6)(q+3)$; the constants originate from the proof of Lemma 29.

▶ **Observation 26.** *Let $[d_1 \,..\, d_2] \in \mathbf{NonOv}(t)$. If $V[j_1 \,..\, j_2]$ is the sample in V, then $U[j_1 + d_1 - t \,..\, j_2 + d_2 + t]$ does not contain a position in a locked fragment, since we defined the sample as an (exceptional) locked fragment.*

▶ **Definition 27.** *For an interval $\mathsf{D} = [d_1 \,..\, d_2]$, denote*

$$\mathsf{scope}(\mathsf{D}) = \mathsf{Ext}_k([j_1 \,..\, j_2] \oplus \mathsf{D}), \quad \mathsf{CritPos}(\mathsf{D}) = \mathit{Occ}_0(Q^{k+1}, U[\mathsf{scope}(\mathsf{D})]).$$

The positions in $\bigcup_{\mathsf{D}\in\mathbf{NonOv}(\hat{q})} \mathsf{CritPos}(\mathsf{D})$ are called critical positions*; see Figure 8.*

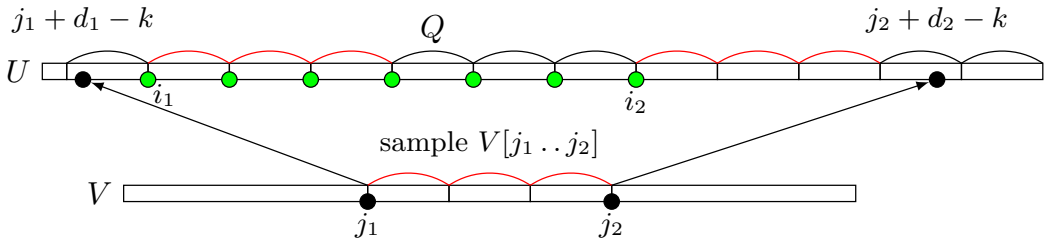

Figure 8 Illustration of basic parameters in the algorithm: $\mathsf{D} = [d_1 \,..\, d_2]$, $I = \mathsf{Ext}_k([j_1 \,..\, j_2] \oplus \mathsf{D})$. We have $I \cap \mathit{locked}(U) = \emptyset$. $\mathsf{CritPos}(\mathsf{D})$ consists of critical positions shown as green circles.

The main idea of the proof of the next lemma is as follows: in an app-match for an offset from D, at least one copy of Q from the sample must match a copy of Q in $\mathsf{scope}(\mathsf{D})$ exactly. For $\mathsf{D} \in \mathbf{NonOv}(\hat{q})$, $\mathsf{scope}(\mathsf{D})$ is a substring of Q^∞. This implies that the whole sample matches a fragment of $\mathsf{scope}(\mathsf{D})$ exactly, which is how critical positions were defined.

▶ **Lemma 28.** *For each position p for which there is a $\hat{q}$-non-overlap app-match (p, x), we have $p \in \bigcup \{ \mathtt{Anchored}_k(P, U, i) : i \text{ is a critical position} \}$.*

The lemma says that it would be enough to consider $\mathtt{Anchored}_k(P, T, i)$ for all critical positions i. Unfortunately, the total number of critical positions can be too large; however, they are grouped into $\mathcal{O}(k^2)$ arithmetic progressions and it is enough to consider the first and the last position in each such progression. In the decision version we use Algorithm 2.

Algorithm 2 Non-overlap case: decision version.

Compute decompositions of U and V into locked fragments and the sample;
Compute $\mathbf{NonOv}(\hat{q})$;
foreach *interval of non-overlap offsets* $\mathsf{D} \in \mathbf{NonOv}(\hat{q})$ **do**
  $i_1 := \min \mathsf{CritPos}(\mathsf{D})$; $i_2 := \max \mathsf{CritPos}(\mathsf{D})$;
  if $\mathtt{AnyAnchored}_k(P, U, i_1) \neq \mathit{none}$ **then return** $\mathtt{AnyAnchored}_k(P, U, i_1)$;
  if $\mathtt{AnyAnchored}_k(P, U, i_2) \neq \mathit{none}$ **then return** $\mathtt{AnyAnchored}_k(P, U, i_2)$;
return *none*;

▶ **Lemma 29.** *Assume that $\lambda_k \leq \frac{m}{2}$. Algorithm 2 works in $\mathcal{O}(k^4)$ time in the* PILLAR *model and returns a circular k-edit occurrence of P in U if any $\hat{q}$-non-overlap app-match exists.*

Proof of Theorem 5, decision version. If $\lambda_k \leq \frac{m}{2}$, Lemma 29 and Corollary 21 cover the decision version of PeriodicSubMatch for $\hat{q}$-non-overlap offsets and $\hat{q}$-overlap offsets, respectively. Together with Lemma 23 used for the corner case that $\lambda_k > \frac{m}{2}$, they yield a solution to a decision version of PeriodicSubMatch. The decision version from Theorem 5 is obtained through the reduction to PeriodicSubMatch of Lemma 12, as the time complexities of all the algorithms in the PILLAR model are $\mathcal{O}(k^5 \log^3 k)$. ◀

5.1 Overview of the proof of Lemma 29

The complexity of Algorithm 2 directly follows from Lemma 6 (computing $\mathtt{Anchored}_k$), Lemma 14 (computing decompositions into locked fragments) and Lemma 22 (computing $\mathbf{NonOv}(\hat{q})$).

For a fragment $F = U[I]$ ($F = V[I]$, respectively), we denote by $\mathit{locked}(F)$ the set $I \cap \mathit{locked}(U)$ ($I \cap \mathit{locked}(V)$, respectively).

▶ **Definition 30.** *Two fragments F_1, F_2 (both of U or both of V) are called* locked-equivalent *if $\mathit{locked}(F_1) = \mathit{locked}(F_2)$ and there are no locked positions in a prefix and a suffix of length $(k+4)q$ in F_1 and in F_2; see Figure 9.*

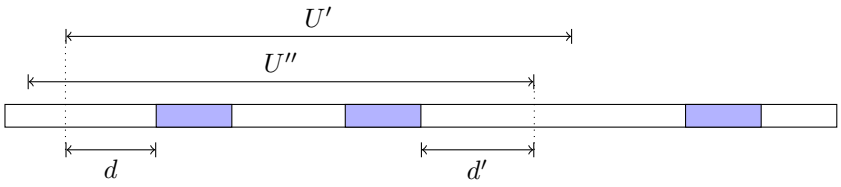

Figure 9 The gray boxes correspond to locked fragments, while $d, d' \geq (k+4)q$. The fragments U' and U'' are locked-equivalent.

We extend Definition 2 and say that a circular k-edit occurrence $T[p\,..\,p']$ of P is *x-anchored* at position i if $\delta_E(T[p\,..\,i), P[x\,..\,m)) + \delta_E(T[i\,..\,p'], P[0\,..\,x)) \leq k$. For a fragment $Y = X[i\,..\,j]$ and integer y, we denote $\mathit{shift}(Y, y) = X[i+y\,..\,j+y]$.

Let $i_1 = \min \mathsf{CritPos}(\mathsf{D})$, $i_2 = \max \mathsf{CritPos}(\mathsf{D})$ as in Algorithm 2. The next lemma shows that in many cases, if $U[p\,..\,p']$ forms a $\hat{q}$-non-overlap app-match that is anchored at a critical position i such that $i_1 < i < i_2$, then the same fragment or a fragment shifted by q positions forms a $\hat{q}$-non-overlap app-match anchored at a critical position $i \pm q$.

▶ **Lemma 31.** *Let $V[j_1\,..\,j_2] = Q^{k+1}$ be the sample, $\mathbf{C} = \mathsf{CritPos}(\mathsf{D})$ where $\mathsf{D} \in \mathbf{NonOv}(\hat{q}/2)$, and $i \in \mathbf{C}$. If $I = [p\,..\,p']$ and $U[I]$ is x-anchored at i, then for any $y \in \{q, -q\}$:*

(a) *If $U[I]$ and $U[I \oplus y]$ are locked-equivalent and $i + y \in \mathbf{C}$, then $U[I \oplus y]$ is x-anchored at $i + y$.*

(b) *If $V' = V^{(x)}$ and $\mathit{shift}(V', y)$ are locked-equivalent and $i - y \in \mathbf{C}$, then $U[I]$ is $(x+y)$-anchored at $i - y$.*

Sketch of the proof. For part (a), $y = -q$, it suffices to show that:

$$\delta_E(U[p\,..\,i), V[x\,..\,j_1)) = \delta_E(U[p-q\,..\,i-q), V[x\,..\,j_1)), \tag{1}$$

$$\delta_E(U[i\,..\,p'], V[j_1\,..\,x+m)) = \delta_E(U[i-q\,..\,p'-q], V[j_1\,..\,x+m)). \tag{2}$$

For example, in (1), fragments $X := U[p\,..\,i)$ and $X' := U[p-q\,..\,i-q)$ contain the same locked fragments of U, just shifted by q positions. Each of X, X' has a length-$\Theta(kq)$ prefix and suffix without locked positions; for the prefix, this follows from the locked-equivalence assumption, whereas for the suffix, we use Observation 26. To prove (1), we notice that an optimal alignment of X and $Y := V[x\,..\,j_1)$ can be divided into parts, such that every second part is a power of Q, and in the remaining parts, there are locked fragments in only one of the strings. This follows from the fact that $p - x$ is a $\Theta(kq)$-non-overlap offset, so the locked fragments of X and the locked fragments of Y are "well-separated". Finally, we show that with such an alignment, shifting X by q positions changes an optimal alignment in a structured manner and so the edit distance to Y stays the same. ◀

The sets $\texttt{Anchored}_k$ contain too little information for proving the correctness of the algorithm. It is important that for any of the $\mathcal{O}(k^2)$ intervals of positions of app-matches $[p_l\,..\,p_r]$ returned by a call to $\texttt{Anchored}_k(P, U, i)$, there exist positions $[p'_l\,..\,p'_r]$ and values

$[x_l\,..\,x_r]$ of cyclic rotations such that $U[p_l\,..\,p'_l]$ is x_l-anchored at i, $U[p_l+1\,..\,p'_l+1]$ is (x_l+1)-anchored at i, etc. Therefore we define

$$\texttt{Anchored}'_k(P,T,i) = \{\,(p,p',x) : T[p\,..\,p'] \text{ is } x\text{-anchored at } i\,\}.$$

For a triad (I,J,L) of intervals of the same size, we denote the combined set of triples

$$\mathsf{zip}(I,J,L) = \{(a+t,b+t,c+t) : 0 \le t < |I|\}, \text{ where } (a,b,c) = (\min(I), \min(J), \min(L)).$$

For example $\mathsf{zip}([1\,..\,3],[5\,..\,7],[2\,..\,4]) = \{(1,5,2),(2,6,3),(3,7,4)\}$. (Treating I,J,L as lists, this can be written in Python as set(zip(I,J,L)).) Just like Lemma 31 states a relation of single elements of the sets $\texttt{Anchored}_k$ for anchors at two consecutive critical positions, the next lemma shows what happens to intervals of positions in $\texttt{Anchored}_k$ (together with end-positions of app-matches and the rotations of P).

Denote by $\mathsf{L\text{-}cut}_q(I), \mathsf{R\text{-}cut}_q(I)$ the operations of removing from the interval I its prefix/suffix of length q, possibly obtaining an empty interval. For example, $\mathsf{L\text{-}cut}_2([2\,..\,5]) = [4\,..\,5]$.

▶ **Lemma 32.** *Let* $\mathsf{D} \in \mathbf{NonOv}(\hat{q}), i_1 = \min \mathsf{CritPos(D)}$, $i_2 = \max \mathsf{CritPos(D)}$. *Assume that for some* $i \in \mathsf{CritPos(D)}$ *such that* $i \neq i_1, i_2$, *we have* $\mathsf{zip}(I_1,I_2,I_3) \subseteq \texttt{Anchored}'_k(P,U,i)$, *where* $|I_1| = |I_2| = |I_3| \ge q$. *Then:*

$$\mathsf{zip}(\mathsf{L\text{-}cut}_q(I_1), \mathsf{L\text{-}cut}_q(I_2), \mathsf{R\text{-}cut}_q(I_3)) \subseteq \texttt{Anchored}'_k(P,U,i+q),$$
$$\mathsf{zip}(\mathsf{R\text{-}cut}_q(I_1), \mathsf{R\text{-}cut}_q(I_2), \mathsf{L\text{-}cut}_q(I_3) \subseteq \texttt{Anchored}'_k(P,U,i-q).$$

For every $p \in \texttt{Anchored}_k(P,U,i)$ that satisfies the assumption of Lemma 31(b) and $i_1 < i < i_2$, that lemma immediately shows that $p \in \texttt{Anchored}_k(P,U,i-q) \cap \texttt{Anchored}_k(P,U,i+q)$. Unfortunately, this assumption does not always hold. However, Lemma 32 shows that this is true for all but at most q elements $p \in \texttt{Anchored}_k(P,U,i)$.

To prove Lemma 32, roughly speaking, we compute a *superposable partition* of intervals $I_1, I_2, I_3, I_3 \oplus m$, such that in each part, locked fragments can occur only in the parts originating from one of the strings U, V. As before, this is possible thanks to the fact that the offset is non-overlapping; here we use the fact that the definition of t-non-overlap offsets (Definition 15) covers the cases $(\Delta' \pm m) \oplus [-t\,..\,t]$. Finally, we apply the appropriate point of Lemma 31 to positions in each part in bulk.

Correctness of Algorithm 2. By Lemma 32, if $I \subseteq \texttt{Anchored}_k(P,U,i)$ for an interval I, then $\mathsf{L\text{-}cut}_q(I) \subseteq \texttt{Anchored}_k(P,U,i+q)$ and $\mathsf{R\text{-}cut}_q(I) \subseteq \texttt{Anchored}_k(P,U,i-q)$. In the proof of Lemma 29, we use Lemma 31 on positions in the first and last q positions of I to show that one of the following conditions hold:

$$(\star)\ I \subseteq \texttt{Anchored}_k(P,U,i \pm q) \text{ or } (\star\star)\ I \ominus q \subseteq \texttt{Anchored}_k(P,U,i-q).$$

In case $(\star)$, by induction we show that $I \subseteq \texttt{Anchored}_k(P,U,i_1) \cup \texttt{Anchored}_k(P,U,i_2)$. In case $(\star\star)$, we show by induction that $J := I \oplus (i_1 - i) \subseteq \texttt{Anchored}_k(P,U,i_1)$. This way we prove the correctness of Algorithm 2.

Reporting version. In the reporting version of Algorithm 2, we prove that the condition $(\star\star)$ implies an interval chain of positions $\mathrm{Chain}(J, (i_2-i_1)/q, q)$ and show a way to efficiently verify this condition given interval I (see Algorithm 3 in Appendix B). Finally, the reporting version of Theorem 5 follows from the reporting version of the overlap case (Lemma 20), the correctness and the complexity of Algorithm 3, the usage of Lemma 23 for the corner case when $\lambda_k > \frac{m}{2}$, and the reduction to PeriodicSubMatch (Lemma 12).

▶ Remark 33. In both versions (decision, reporting), the bottleneck of the algorithm's running time is the overlap case, while the most technically demanding part is the non-overlap case.

References

1 Andris Ambainis. Quantum query algorithms and lower bounds. In *Classical and New Paradigms of Computation and their Complexity Hierarchies*, pages 15–32, 2004. doi:10.1007/978-1-4020-2776-5_2.

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

3 Lorraine A. K. Ayad, Carl Barton, and Solon P. Pissis. A faster and more accurate heuristic for cyclic edit distance computation. *Pattern Recognition Letters*, 88:81–87, 2017. doi:10.1016/j.patrec.2017.01.018.

4 Lorraine A. K. Ayad and Solon P. Pissis. MARS: Improving multiple circular sequence alignment using refined sequences. *BMC Genomics*, 18(1):86, 2017. doi:10.1186/s12864-016-3477-5.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

6 Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52:3457–3467, 1995. doi:10.1103/PhysRevA.52.3457.

7 Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Fast algorithms for approximate circular string matching. *Algorithms for Molecular Biology*, 9:9, 2014. doi:10.1186/1748-7188-9-9.

8 Gabriel Bathie, Tomasz Kociumaka, and Tatiana Starikovskaya. Small-space algorithms for the online language distance problem for palindromes and squares. In *34th International Symposium on Algorithms and Computation, ISAAC 2023*, volume 283 of *LIPIcs*, pages 10:1–10:17, 2023. doi:10.4230/LIPICS.ISAAC.2023.10.

9 Karl Bringmann, Philip Wellnitz, and Marvin Künnemann. Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1126–1145. SIAM, 2019. doi:10.1137/1.9781611975482.69.

10 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

11 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 643–656. ACM, 2020. doi:10.1145/3357713.3384266.

12 Timothy M. Chan, Ce Jin, Virginia Vassilevska Williams, and Yinzhan Xu. Faster algorithms for text-to-pattern Hamming distances. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 2188–2203. IEEE, 2023. doi:10.1109/FOCS57990.2023.00136.

13 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. Circular pattern matching with k mismatches. *Journal of Computer and System Sciences*, 115:73–85, 2021. doi:10.1016/j.jcss.2020.07.003.

14 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Approximate circular pattern matching. *CoRR*, abs/2208.08915, 2022. arXiv:2208.08915, doi:10.48550/ARXIV.2208.08915.

15 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Approximate circular pattern matching. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 35:1–35:19, 2022. doi:10.4230/LIPIcs.ESA.2022.35.

16 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 978–989. IEEE, 2020. Full version: arXiv:2004.08350v2. doi:10.1109/FOCS46700.2020.00095.

17 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance: A reduction to dynamic puzzle matching and the seaweed monoid of permutation matrices. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 698–707. IEEE, 2022. Full version: arXiv:2204.03087v1. doi:10.1109/FOCS54457.2022.00072.

18 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.

19 Raphaël Clifford, Paweł Gawrychowski, Tomasz Kociumaka, Daniel P. Martin, and Przemysław Uznański. The dynamic k-mismatch problem. In *33rd Annual Symposium on Combinatorial Pattern Matching, CPM 2022*, volume 223 of *LIPIcs*, pages 18:1–18:15, 2022. doi:10.4230/LIPIcs.CPM.2022.18.

20 Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM Journal on Computing*, 31(6):1761–1782, 2002. doi:10.1137/S0097539700370527.

21 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In *29th ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99.

22 Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62.

23 Roberto Grossi, Costas S. Iliopoulos, Robert Mercas, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Circular sequence comparison: algorithms and applications. *Algorithms for Molecular Biology*, 11:12, 2016. doi:10.1186/s13015-016-0076-6.

24 Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001.

25 Ramesh Hariharan and V. Vinay. String matching in Õ(sqrt(n)+sqrt(m)) quantum time. *Journal of Discrete Algorithms*, 1(1):103–110, 2003. doi:10.1016/S1570-8667(03)00010-8.

26 Tommi Hirvola and Jorma Tarhio. Approximate online matching of circular strings. In *Experimental Algorithms - 13th International Symposium, SEA 2014*, pages 315–325. Springer, 2014. doi:10.1007/978-3-319-07959-2_27.

27 Ce Jin and Jakob Nogler. Quantum speed-ups for string synchronizing sets, longest common substring, and k-mismatch matching. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 5090–5121. SIAM, 2023. doi:10.1137/1.9781611977554.ch186.

28 Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In *STOC 2022: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1657–1670. ACM, 2022. doi:10.1145/3519935.3520061.

29 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. doi:10.1137/0206024.

30 Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small-space and streaming pattern matching with k edits. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 885–896. IEEE, 2021. doi:10.1109/FOCS52979.2021.00090.

31 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. Full version: arXiv:1311.6235. doi:10.1137/1.9781611973730.36.

32 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

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

34 Maurice Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35(2):73–78, 1990. doi:10.1016/0020-0190(90)90109-B.

35 Vicente Palazón-González and Andrés Marzal. Speeding up the cyclic edit distance using LAESA with early abandon. *Pattern Recognition Letters*, 62:1–7, 2015. doi:10.1016/j.patrec.2015.04.013.

36 Vicente Palazón-González, Andrés Marzal, and Juan Miguel Vilar. On hidden Markov models and cyclic strings for shape recognition. *Pattern Recognition*, 47(7):2490–2504, 2014. doi:10.1016/j.patcog.2014.01.018.

37 Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *37th Annual Symposium on Foundations of Computer Science, FOCS 1996*, pages 320–328. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548491.

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

39 Teresa Anna Steiner. Differentially private approximate pattern matching. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024*, volume 287 of *LIPIcs*, pages 94:1–94:18, 2024. doi:10.4230/LIPICS.ITCS.2024.94.

A Origin of Theorem 10

An algorithm that efficiently computes a representation of $\mathsf{Occ}_k(P,T)$ is encapsulated in [16, Main Theorem 9] [3]. The first step of this algorithm is the analysis of the pattern specified in [16, Lemma 6.4], which results in computing either a set of *breaks*, a set of *repetitive regions*, or a primitive string Q that is of length at most $m/(128k)$ and satisfies $\delta_E(P, {}^*Q^*) < 2k$. In the presence of breaks or repetitive regions, we have $\lfloor |\mathsf{Occ}_k(P,T)|/k \rfloor \leq 642045 \cdot (n/m) \cdot k$, see [16, Lemmas 5.21 and 5.24]. In the case where the analysis of the pattern returns an approximate period Q, we can use [16, Lemma 6.5] to find a rotation Q_1 of Q such that $\delta_E(P, {}^*Q_1^*) = \delta_E(P, Q_1^*)$. Set $Q := Q_1$. Now, let us also compute all k-edit occurrences of the reversal of P in the reversal of T. Then, we can trim T, obtaining a string $\bar{T}$ so that all k-edit occurrences of P in T are preserved in $\bar{T}$, and P has a k-edit occurrence both as a prefix and as a suffix of $\bar{T}$. We can then directly apply [16, Theorem 5.2] with $d = 8k$ to obtain the stated properties of $\bar{T}$; for the fact that $\delta_E(\bar{T}, Q^*) \leq 24k$ holds see the fourth paragraph of the proof of that theorem. The length of Q can be instead bounded by $m/(256k)$ with all other constants remaining unchanged; this is because the bottleneck for the number of occurrences in the case where P is not almost periodic stems from repetitive regions and is not sensitive to the exact length of Q. That is, it is only the number of occurrences in the case where the analysis of the pattern yields $2k$ breaks that can be larger (by a multiplicative factor of 2), but the bound stated above is dominant.

[3] When referring to statements of [16], we use their numbering in the full (arxiv) version of the paper.

B Reporting Version

Algorithm 3 is a reporting version of Algorithm 2. Algorithm 3 outputs all $\hat{q}$-non-overlap app-matches as a collection of $\mathcal{O}(k^4)$ interval chains (some of which can be single intervals).

Algorithm 3 Non-overlap case: reporting version.

foreach *interval of offsets* $\mathsf{D} \in \mathbf{NonOv}(\hat{q})$ **do**
  $i_1 := \min \mathsf{CritPos}(\mathsf{D})$; $i_2 := \max \mathsf{CritPos}(\mathsf{D})$;
  $\mathcal{Z}_1 := \mathtt{Anchored}_k(P, U, i_1)$;
  $\mathcal{Z}_2 := \mathtt{Anchored}_k(P, U, i_2)$;
  report $\mathcal{Z}_1 \cup \mathcal{Z}_2$;
  foreach *interval* $I = [p_l \,..\, p_r]$ **in** $\mathcal{Z}_1$, *with* $p_l > 0$ *and* $p_r + m + k \leq |U|$ **do**
    if $(\{p_l - 1\} \cup I) \cap \mathit{locked}(U) = \emptyset$ **then**
      report $\mathrm{Chain}(I, (i_2 - i_1)/q, q)$;

C k-Edit CPM in Other Settings

Theorem 5 is stated in the PILLAR model. In the standard setting, all PILLAR operations can be implemented in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ preprocessing [14, Section 3]; this yields Theorem 1.

We now present our results for the internal, dynamic, fully compressed, and quantum settings. In each case, in the reporting version of the problem, the output is represented as a union of $\mathcal{O}((|T|/|P|) \cdot k^6)$ interval chains.

With the same implementations of operations in the internal setting as in the standard setting, we obtain an efficient implementation.

▶ **Theorem 34** (Internal Setting). *Given two substrings P and T of a length-n string S, reporting and decision versions of k-Edit CPM for P and T can be solved in $\mathcal{O}((|T|/|P|)k^6)$ time and $\mathcal{O}((|T|/|P|)k^5 \log^3 k)$ time, respectively, after $\mathcal{O}(n)$ preprocessing on S.*

Let $\mathcal{X}$ be a growing collection of non-empty persistent strings; it is initially empty, and then undergoes updates by means of the following operations:

- $\mathtt{Makestring}(U)$: Insert a non-empty string U to $\mathcal{X}$
- $\mathtt{Concat}(U, V)$: Insert string UV to $\mathcal{X}$, for $U, V \in \mathcal{X}$
- $\mathtt{Split}(U, i)$: Insert $U[0\,..\,i)$ and $U[i\,..\,|U|)$ to $\mathcal{X}$, for $U \in \mathcal{X}$ and $i \in [0\,..\,|U|)$.

By N we denote an upper bound on the total length of all strings in $\mathcal{X}$ throughout all updates executed by an algorithm. A collection $\mathcal{X}$ of non-empty persistent strings of total length N can be dynamically maintained with operations $\mathtt{Makestring}(U)$, $\mathtt{Concat}(U, V)$, $\mathtt{Split}(U, i)$ requiring time $\mathcal{O}(\log N + |U|)$, $\mathcal{O}(\log N)$ and $\mathcal{O}(\log N)$, respectively, so that PILLAR operations can be performed in time $\mathcal{O}(\log^2 N)$. All stated time complexities hold with probability $1 - 1/N^{\Omega(1)}$; see [21, 16]. Moreover, Kempa and Kociumaka [28, Section 8 in the arXiv version] presented an alternative deterministic implementation, which supports operations $\mathtt{Makestring}(U)$, $\mathtt{Concat}(U, V)$, $\mathtt{Split}(U, i)$ in $\mathcal{O}(|U| \log^{\mathcal{O}(1)} \log N)$, $\mathcal{O}(\log |UV| \log^{\mathcal{O}(1)} \log N)$, and $\mathcal{O}(\log |U| \log^{\mathcal{O}(1)} \log N)$ time, respectively, so that PILLAR operations can be performed in time $\mathcal{O}(\log N \log^{\mathcal{O}(1)} \log N)$. With these implementations, we obtain the following result.

▶ **Theorem 35** (Dynamic Setting). *A collection $\mathcal{X}$ of non-empty persistent strings of total length N can be dynamically maintained with operations* $\texttt{Makestring}(U)$, $\texttt{Concat}(U,V)$, $\texttt{Split}(U,i)$ *requiring time $\mathcal{O}(\log N + |U|)$, $\mathcal{O}(\log N)$ and $\mathcal{O}(\log N)$, respectively, so that, given two strings $P,T \in \mathcal{X}$ and an integer threshold $k > 0$, we can solve k-Edit CPM in $\mathcal{O}((|T|/|P|)\cdot k^6 \log^2 N)$ time for the reporting variant and $\mathcal{O}((|T|/|P|)\cdot k^5 \log^3 k \log^2 N)$ time for the decision variant. All stated time complexities hold with probability $1 - 1/N^{\Omega(1)}$. Randomization can be avoided at the cost of a $\log^{\mathcal{O}(1)} \log N$ multiplicative factor in all the update times, with k-Edit CPM queries answered in $\mathcal{O}((|T|/|P|)\cdot k^6 \log N \log^{\mathcal{O}(1)} \log N)$ time (reporting version) or $\mathcal{O}((|T|/|P|)\cdot k^5 \log^3 k \log N \log^{\mathcal{O}(1)} \log N)$ time (decision version).*

A straight line program (SLP) is a context-free grammar G that consists of a set Σ of terminals and a set $N_G = \{A_1, \ldots, A_n\}$ of non-terminals such that each $A_i \in N_G$ is associated with a unique production rule $A_i \to f_G(A_i) \in (\Sigma \cup \{A_j : j < i\})^*$. We can assume without loss of generality that each production rule is of the form $A \to BC$ for some symbols B and C (that is, the given SLP is in Chomsky normal form). Every symbol $A \in S_G := N_G \cup \Sigma$ generates a unique string, which we denote by $gen(A) \in \Sigma^*$. The string $gen(A)$ can be obtained from A by repeatedly replacing each non-terminal with its production. We say that G generates $gen(G) := gen(A_n)$.

In the fully compressed setting, given a collection of straight-line programs (SLPs) of total size n generating strings of total length N, each $\texttt{PILLAR}$ operation can be performed in $\mathcal{O}(\log^2 N \log\log N)$ time after an $\mathcal{O}(n \log N)$-time preprocessing [14, Section 3]. If we applied Theorem 1 directly in the fully compressed setting, we would obtain $\Omega(N/M)$ time, where N and M are the uncompressed lengths of the text and the pattern, respectively. Instead, we can adapt an analogous procedure provided in [16, Section 7.2] for (non-circular) pattern matching with edits to obtain the following result.

▶ **Theorem 36** (Fully Compressed Setting). *Let G_T denote a straight-line program of size n generating a string T, let G_P denote a straight-line program of size m generating a string P, let $k > 0$ denote an integer threshold, and set $N := |T|$ and $M := |P|$. We can solve k-Edit CPM in $\mathcal{O}(m \log N + nk^6 \log^2 N \log\log N)$ time (counting version) or $\mathcal{O}(m \log N + nk^5 \log^3 k \log^2 N \log\log N)$ time (decision version). A representation of the occurrences in the form of interval chains can be returned in $\mathcal{O}((N/M)\cdot k^6)$ extra time.*

We say an algorithm on an input of size n succeeds *with high probability* if the success probability can be made at least $1 - 1/n^c$ for any desired constant $c > 1$.

In what follows, we assume the input strings can be accessed in a quantum query model [1, 10]. We are interested in the time complexity of our quantum algorithms [6].

▶ **Observation 37** ([27, Observation 2.3]). *For any two strings S,T of length at most n, $\mathsf{LCP}(S,T)$ or $\mathsf{LCP}_R(S,T)$ can be computed in $\tilde{\mathcal{O}}(\sqrt{n})$ time in the quantum model with high probability.*

Hariharan and Vinay [25] gave a near-optimal quantum algorithm for the decision version of exact PM. We formalize this next.

▶ **Theorem 38** ([25]). *The decision version of PM can be solved in $\tilde{\mathcal{O}}(\sqrt{n})$ time in the quantum model with high probability. If the answer is YES, then the algorithm returns a witness occurrence.*

By employing Theorem 38 and binary search to find the period of S [31] and thus its full list of occurrences expressed as an arithmetic progression in T, we obtain the following.

▶ **Observation 39.** *For any two strings S, T of length at most n, with $|T| \leq 2|S|$, $\mathsf{IPM}(S, T)$ can be computed in $\tilde{\mathcal{O}}(\sqrt{n})$ time in the quantum model with high probability.*

All other `PILLAR` operations are performed trivially in $\mathcal{O}(1)$ quantum time. Thus while all `PILLAR` operations can be implemented in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time preprocessing in the standard setting by a classic algorithm, in the quantum setting, all `PILLAR` operations can be implemented in $\tilde{\mathcal{O}}(\sqrt{m})$ quantum time *with no preprocessing*, as we always deal with strings of length $\mathcal{O}(m)$. We obtain the following results.

▶ **Theorem 40** (Quantum Setting)**.** *The reporting version of the k-Edit CPM problem can be solved in $\tilde{\mathcal{O}}((n/\sqrt{m})k^6)$ time in the quantum model with high probability. The decision version of the k-Edit CPM problem can be solved in $\tilde{\mathcal{O}}((n/\sqrt{m})k^5)$ time in the quantum model with high probability.*