Online Approximate Circular Pattern Matching in Small Space

Panagiotis Charalampopoulos
King's College London, UK

Taha El Ghazi
DIENS, École normale supérieure de Paris, PSL Research University, France

Jonas Ellert
University of Southern Denmark, Odense, Denmark

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

In approximate circular pattern matching the goal is to compute all approximate occurrences of all rotations of a pattern P in a text T. We study this problem under the two most fundamental string distance metrics, the Hamming distance and the edit distance, in the setting where the text arrives online and the available space is limited. Specifically, we wish to report each ending position j of an approximate occurrence before symbol $T[j+1]$ arrives, using sublinear space on top of having read-only access to P and (the seen prefix of) T. For both variants, we present algorithms that use $O(\text{poly}(k))$ extra space and process each arriving symbol in $O(\text{poly}(k))$ time. Notably, with an overhead, our algorithms can be lifted to the asymmetric streaming setting, where we only have read-only access to the pattern for free and account for all extra space.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Pattern matching

Keywords and phrases Small-space algorithms, approximate pattern matching, circular pattern matching

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.99

Funding *Jonas Ellert*: Partially supported by an ERCIM Alain Bensoussan fellowship (during his time at CWI Amsterdam, The Netherlands).

1 Introduction

In the circular pattern matching (CPM) problem, one is given a pattern P of length m and a text T of length n, and must report all positions j of the text such that a suffix of $T[1\,..\,j]$ is similar to a rotation of P. This is motivated, e.g., by applications in bioinformatics and image processing: in bioinformatics, the starting position of a circular sequence can vary significantly, as circular molecular structures are not sequenced and linearized consistently (see [17]); in image processing, the contour of a shape can be represented using a directional chain code (a circular sequence), particularly when the orientation of the image is irrelevant [2].

In *exact* CPM, we must find all positions j such that $T(j-m\,..\,j]$ equals a rotation of P. Exact CPM has been extensively studied in the classical offline setting, where the input is given in full and the space complexity accounts for all the space used [12, 16, 18–20, 24]. In the read-only setting, one assumes read-only random access to the input string(s) and only accounts for the extra space, that is, the space used by the algorithm/data structure on top of the space needed to store the input. For strings over an alphabet of size σ, the

classical read-only solution for CPM via the suffix automaton of $P \cdot P$ runs in $\mathcal{O}(n \log \sigma)$ time and uses $\mathcal{O}(m)$ extra space [22]. Recently, Charalampopoulos et al. [7] showed a simple $\mathcal{O}(n)$-time $\mathcal{O}(m)$-space solution. Finally, Bathie et al. [3] showed that exact CPM can be solved optimally *online* using $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space.

With these optimal bounds in the online read-only model, a natural next step is to investigate more restricted models, such as streaming. Unfortunately, Bathie et al. [3] showed that achieving $o(m)$ space in the streaming model is impossible. This further motivated them to study the complexity of the problem in the asymmetric streaming model proposed by Saks and Seshadhri [23] (see also a work of Andoni, Krauthgamer, and Onak [1]). In this model, the pattern is read-only and the text arrives as a stream. Formally, the text arrives symbol by symbol, and the algorithm must account for any space used to store information about it and perform computations, while the space required to store the read-only string is not counted toward the space complexity. Bathie et al. [3] showed that, for every $\tau \in [\sqrt{m} \log m (\log \log m)^3 \,..\, m]$, there exists an asymmetric streaming algorithm that solves exact CPM in time $\tilde{O}(mn/\tau)$[1] using $\mathcal{O}(\tau)$ extra space. This was further improved to $\tilde{O}((m^3/\tau^6 + 1)n)$ time and $\tilde{O}(\tau)$ extra space in [6] via a black box for converting online read-only pattern matching algorithms to asymmetric streaming algorithms via function inversion.

In most applications, searching for exact occurrences is insufficient, as noise in the data may obscure them. To address this, we relax the notion of closeness by allowing the suffix to be at distance at most k from a rotation of the pattern, where k is a parameter of the problem. We focus on the two most fundamental string distances: the Hamming distance and the edit distance. Edit distance is particularly relevant in applications such as bioinformatics, while the more tractable Hamming distance serves as a useful stepping stone toward efficient edit distance solutions. CPM under Hamming and edit distances has been considered in the offline setting [7, 8, 11]. However, no online read-only algorithms or asymmetric streaming algorithms have existed prior to this work.

Our contribution. We continue the study of CPM under the Hamming and the edit distance. Formally, for a positive integer k, a position $j \in [1 \,..\, n]$ of a text T is called a *circular k-mismatch (resp., k-edit) occurrence* of P if the Hamming (resp., edit) distance between some suffix of $T[1 \,..\, j]$ and a rotation of P is at most k.

k-mismatch (resp., k-edit) CPM
Input: Integers $n \geq m \geq k > 0$, alphabet Σ, a string $T \in \Sigma^n$, a pattern $P \in \Sigma^m$.
Output: All circular k-mismatch (resp., k-edit) occurrences of P in T.

We propose the first online read-only solutions for these two versions of CPM:

▶ **Theorem 1.** *There is an online read-only algorithm that solves the k-mismatch* CPM *problem using $\mathcal{O}(k^4)$ worst-case time per symbol and $\mathcal{O}(k^5)$ space.*

▶ **Theorem 2.** *There is an online read-only algorithm that solves the k-edit* CPM *problem using $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)})$ amortized time per symbol and $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)})$ space.*

[1] Hereafter, $\tilde{\mathcal{O}}$ means that we hide factors polylogarithmic in n.

Our algorithms are similar in spirit to the algorithms for the offline setting [7, 8, 11]. However, due to the more restrictive online setting, we naturally have to overcome several challenges. Our main technical contribution is for the edit distance case. Locked fragments (see [9, 11, 13]) provide a way of dealing with the computation of approximate matches of an almost periodic rotation P' of P in an almost periodic fragment F of T. We introduce the notion of super-locked fragments, which are more robust with respect to the operation of appending symbols, as needed for processing T. Then, we use the super-locked fragments of P' and F to structurally analyze these strings and their edit distance, leveraging the combinatorial insights to obtain an efficient algorithm.

By applying the black box for converting online read-only algorithms to asymmetric streaming algorithms [6] (see Appendix A), we obtain the following:

▶ **Corollary 3.** *Consider positive integers τ and k with $k = \tilde{\mathcal{O}}(\tau)$, read-only access to a pattern P of length m, and a streaming text T of length n. After preprocessing P, we can*

1. *Compute all circular k-mismatch occurrences of P in T using space $\tilde{\mathcal{O}}(k^5 + \tau)$ and worst-case time $\tilde{\mathcal{O}}(k^4 + k^3m^3/\tau^6)$ per symbol of T; and*
2. *Compute all circular k-edit occurrences of P in T using space $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)} + \tau)$ and amortized time $\tilde{\mathcal{O}}(k^{\mathcal{O}(1)} + k^3m^3/\tau^6)$ per symbol of T.*

Technical Overview

We now present the main technical ideas behind our contribution.

Online Read-only Algorithm for k-Mismatch CPM. If a position $j \in [m\,..\,n]$ of a text T is a circular k-mismatch occurrence of P, then a substring of P^2 is at Hamming distance at most k from $T(j-m\,..\,j]$. We partition P^2 into $\Theta(k)$ blocks of length $\Theta(m/k)$. By the pigeonhole principle, if j is a circular k-mismatch occurrence of P, then $T(j-m\,..\,j]$ contains an exact occurrence of at least one of the blocks, and this block is far from the end of the substring. The algorithm uses these occurrences as "anchors".

In the simple case when an anchor is not periodic, by the Fine–Wilf periodicity lemma [14] (see also Section 2) it has $\mathcal{O}(k)$ occurrences in any $2m$-length fragment of T. It follows that we can check whether each occurrence of the anchor generates a circular occurrence of P in T by extending it naively in both directions.

The case when the anchor has a period Q is more involved. Again, by the Fine–Wilf periodicity lemma [14], the exact occurrences of an anchor in any $2m$-length fragment of T span a periodic substring with period Q. We extend the anchor in P^2 and the periodic substring in T in both directions. We then consider several subcases depending on how the periodic regions in P^2 and T are aligned. In more detail, we extend periodicity of the anchor in the pattern and in the text until we encounter $k+1$ mismatches on both sides (we refer to such mismatches as "misperiods"). We can test each alignment where a pair of misperiods in the pattern and in the text are aligned individually (they are few). The principal technical difficulty is that we need to treat k-mismatch occurrences where a pair of misperiods is aligned on the left and on the right of the anchor separately: in the first case, the pair is far from the endpoint of the occurrence and we can de-amortize necessary computations, while in the second case the pair of aligned misperiods is close to the endpoint of the occurrence, so we need to start necessary computations in advance. When no two misperiods are aligned, the Hamming distance is defined as a sum of the number of misperiods in the m-length fragments of P^2 and the text, and since the number of misperiods is bounded by $\mathcal{O}(k)$, we can compute the distance efficiently.

Online Read-only Algorithm for k-Edit CPM. Our solution for the edit distance resembles the one for Hamming distance, but, as is often the case, it is significantly more involved. It also follows the offline solution of [11] with several technical differences. Again, we cut P^2 into $\Theta(k)$ non-overlapping blocks of length $\Theta(m/k)$. Then, every circular k-edit occurrence of P in T has an exact occurrence of one of the blocks, allowing us to use them as anchors.

If an anchor is not periodic, then every $2m$-length fragment of T contains $\mathcal{O}(k)$ exact occurrences of the anchor. Using a procedure similar to the Landau–Vishkin algorithm [21], we can extend each of them in both directions until we encounter k edits, allowing us to compute the implied circular k-edit occurrences.

The case when the anchor has a period Q requires much more work. In the spirit of the Hamming distance solution, we extend periodic fragments in the pattern and in the text to allow for $\mathcal{O}(k)$ edits, and would like to consider two cases depending on whether the edits between the text and some substring of Q^∞ and the edits between the pattern and some substring of Q^∞ are aligned or not. However, there is an extra technical challenge caused by the fact that the edits between the pattern and the text might imply shifts. To overcome this challenge, we define a novel variant of so-called locked fragments introduced by Cole and Hariharan in [13], adapted by Charalampopoulos, Kociumaka, and Wellnitz [9], and exploited for offline k-edit CPM by Charalampopoulos et al. [11]. We call these fragments *super-locked fragments* as they give us stronger guarantees and are easier to maintain for an online text. Intuitively, a super-locked fragment includes an edit and $\Omega(k)$ copies of Q before and after it. We then consider two cases based on whether two super-locked fragments in the pattern and in the text overlap or not similarly to [11]. The most challenging case is that of super-locked fragments not overlapping. We give an overview of our approach for this case in Section 4, after relevant notation has been introduced.

2 Preliminaries

An *alphabet* Σ is a finite set of *symbols*. An n-length string T over alphabet Σ is a sequence of n symbols. For an integer $\ell \geq 0$, we write Σ^ℓ for the set of strings of length ℓ over Σ, and $\Sigma^{\leq \ell}$ for the set of strings over Σ of length at most ℓ. We write $T[1\,..\,n]$ to denote that T is a string of length $|T| = n$. For $i, j \in [1\,..\,n]$, we write $T[i]$ to denote the i-th symbol in T, and $T[i\,..\,j] = T[i\,..\,j+1) = T(i-1\,..\,j] = T(i-1\,..\,j+1)$ to denote the *fragment* $T[i]T[i+1]\cdots T[j]$ if $i \leq j$, and the *empty string* otherwise. We say that another string $P[1\,..\,m]$ is a *substring* of T, or equivalently that it *occurs* in T, if there is $i \in [m+1\,..\,n]$ such that $\forall j \in [0\,..\,m) : T[i-j] = P[j]$. Any such fragment $T(i-m\,..\,i]$ is an *occurrence* of P in T, and we also call the position i an occurrence of P in T. *Suffixes* and *prefixes* of T respectively have the form $T[i\,..\,n]$ and $T[1\,..\,i]$, and we may simply write $T[i\,..]$ and $T[..\,i]$ instead. The *reverse* of T is $\mathsf{rev}(T) = T[n]T[n-1]\cdots T[2]T[1]$.

The *concatenation* $S[1]S[2]\cdots S[m]\,T[1]T[2]\cdots T[n]$ of strings $S[1\,..\,m]$ and $T[1\,..\,n]$ is denoted by ST or $S \cdot T$. The concatenation of $k \in \mathbb{Z}_{>0}$ copies of S is denoted by S^k, while the infinite string obtained by concatenating infinitely many copies of S is denoted by S^∞.

A string T is *primitive* if it cannot be expressed as S^k for any string S and any integer $k > 1$. The *rotation* operation $\mathrm{rot}(\star)$ takes a string as input and moves its first symbol to the end; that is, $\mathrm{rot}(T) := T[2\,..\,|T|]T[1]$. We say that a string $\mathrm{rot}^i(T)$ is a *rotation* of T. A primitive string does not match any of its non-trivial rotations, that is, $T = \mathrm{rot}^i(T)$ iff $i \equiv 0 \pmod{|T|}$. An integer $p > 0$ is a *period* of T if $T[i] = T[i+p]$ for all $i \in [1\,..\,n-p]$. The smallest period of T is referred to as *the period* of T and is denoted by $\mathsf{per}(T)$. We may also call the string $T[1\,..\,\mathsf{per}(T)]$ the period of T. A string T is called *periodic* if $\mathsf{per}(T) \leq |T|/2$.

▶ **Fact 4** ([5]). *There is an online read-only algorithm that reports all (exact) occurrences of a pattern $P[1\,..\,m]$ in a text $T[1\,..\,n]$ using constant space and $\mathcal{O}(n)$ time.*

▶ **Fact 5** (Corollary of Fine–Wilf periodicity lemma [14]). *Consider strings X of length m and Y of length at most $2m-1$. If X is not periodic, then it has at most two occurrences in Y, and otherwise its occurrences in Y form one arithmetic progression with difference equal to $\mathsf{per}(X)$.*

▶ **Fact 6** ([15, Lemma 6]). *Given a read-only string X one can decide in $\mathcal{O}(|X|)$ time and constant space if X is periodic and if so, compute $\mathsf{per}(X)$.*

Hamming and edit distances. The Hamming distance between two strings X, Y is the number of positions i such that $X[i] \neq Y[i]$ (*mismatches*) if $|X| = |Y|$, and ∞ otherwise. For convenience, we may say suffix (resp. prefix) of Y^∞ instead of suffix (resp. prefix) of Y^t for some t. We denote the Hamming distance between a string X and any prefix and any suffix of Y^∞ by $\mathsf{hd}(X, Y^*)$ and $\mathsf{hd}(X, {}^*Y)$, respectively.

For a formal definition of edit distance, we first define an *alignment* between strings.

▶ **Definition 7** ([10, full version, Definition 2.1]). *A sequence $\mathcal{A} = (x_i, y_i)_{i=0}^m$ is an* alignment *of $X[x\,..\,x')$ onto $Y[y\,..\,y')$, denoted by $\mathcal{A} : X[x\,..\,x') \to Y[y\,..\,y')$, if we have $(x_0, y_0) = (x, y)$, $(x_m, y_m) = (x', y')$, and $\forall i \in [0\,..\,m] : (x_{i+1}, y_{i+1}) \in \{(x_i+1, y_i+1), (x_i+1, y_i), (x_i, y_i+1)\}$.*

If $(x_{i+1}, y_{i+1}) = (x_i+1, y_i)$, we say that $\mathcal{A}$ deletes *$X[x_i]$. If $(x_{i+1}, y_{i+1}) = (x_i, y_i+1)$, we say that $\mathcal{A}$* inserts *$Y[y_i]$. If $(x_{i+1}, y_{i+1}) = (x_i+1, y_i+1)$, we say that $\mathcal{A}$* aligns *$X[x_i]$ and $Y[y_i]$. If additionally $X[x_i] = Y[y_i]$, we say that $\mathcal{A}$* matches *$X[x_i]$ and $Y[y_i]$. Otherwise, we say that $\mathcal{A}$* substitutes *$X[x_i]$ with $Y[y_i]$.*

The *cost* of an alignment $\mathcal{A}$ of $X[x\,..\,x')$ onto $Y[y\,..\,y')$, denoted $\mathrm{ed}^{\mathcal{A}}(X[x\,..\,x'), Y[y\,..\,y'))$, is the total number of symbols that $\mathcal{A}$ inserts, deletes, or substitutes. We define the edit distance $\mathrm{ed}(X, Y)$ as the minimum cost of an alignment of $X[1\,..\,|X|]$ onto $Y[1\,..\,|Y|]$. An alignment of X onto Y is *optimal* if its cost is equal to $\mathrm{ed}(X, Y)$.

Given an alignment $\mathcal{A} : X[x\,..\,x') \to Y[y\,..\,y')$ and a fragment $X[\bar{x}\,..\,\bar{x}')$ of $X[x\,..\,x')$, we write $\mathcal{A}(X[\bar{x}\,..\,\bar{x}'))$ for the fragment $Y[\bar{y}\,..\,\bar{y}')$ of $Y[y\,..\,y')$ that $\mathcal{A}$ aligns against $X[\bar{x}\,..\,\bar{x}')$. Given that insertions and deletions may render this definition ambiguous, we formally set $\bar{y} := \min\{\hat{y} : (\bar{x}, \hat{y}) \in \mathcal{A}\}$ and $\bar{y}' = y'$ if $\bar{x}' = x'$ and to $\min\{\hat{y}' : (\bar{x}', \hat{y}') \in \mathcal{A}\}$ otherwise. This particular choice satisfies the following decomposition property.

▶ **Fact 8** ([10, full version, Fact 2.2]). *For any alignment $\mathcal{A}$ of X onto Y and any decomposition $X = X_1 \cdots X_t$ of X into t fragments, $Y = \mathcal{A}(X_1) \cdots \mathcal{A}(X_t)$ is a decomposition of Y into t fragments such that $\mathrm{ed}^{\mathcal{A}}(X, Y) \geq \sum_{i=1}^t \mathrm{ed}(X_i, \mathcal{A}(X_i))$.*

Further, if $\mathcal{A}$ is an optimal alignment, we have $\mathrm{ed}^{\mathcal{A}}(X, Y) = \sum_{i=1}^t \mathrm{ed}(X_i, \mathcal{A}(X_i))$.

▶ **Definition 9** ([10, full version, Definition 2.3]). *For an alignment $\mathcal{A} : X[x\,..\,x') \to Y[y\,..\,y')$, and a pair $(x_i, y_i) \in \mathcal{A}$ where $\mathcal{A}$ does not match x_i and y_i, define a* breakpoint *B as*

$$B := \begin{cases} (x_i, y_i, \texttt{INS}), & \text{if } \mathcal{A} \text{ inserts } Y[y_i], \\ (x_i, y_i, \texttt{DEL}), & \text{if } \mathcal{A} \text{ deletes } X[x_i], \\ (x_i, y_i, \texttt{SUB}), & \text{if } \mathcal{A} \text{ substitutes } Y[y_i] \text{ for } X[x_i], \text{ and} \\ (x_i, y_i, \perp), & \text{if } x_i = x' \text{ and } y_i = y' \text{ or if } x_i = x-1 \text{ and } y_i = y-1. \end{cases}$$

We write $\mathsf{B}_{X,Y}(\mathcal{A})$ for the set of breakpoints.

Observe that we can uniquely reconstruct $\mathcal{A}$ from $\mathsf{B}_{X,Y}(\mathcal{A})$. Our algorithms use $\mathsf{B}_{X,Y}(\mathcal{A})$ (with elements stored in a sorted array) to represent an alignment $\mathcal{A}$; if the cost of $\mathcal{A}$ is d, then this *breakpoint representation* takes $\mathcal{O}(d+1)$ space.

For brevity, we use the following shorthands. For an integer $k \geq 0$, we denote the *k-bounded edit distance* of X and Y by $\mathrm{ed}_k(X,Y) = \min\{\mathrm{ed}(X,Y), k+1\}$. By $\mathrm{ed}(X,Y^*)$, $\mathrm{ed}(X,{}^*Y)$, and $\mathrm{ed}(X,{}^*Y^*)$ we denote the minimum edit distance between string X and any prefix, suffix, and substring of a string Y^∞, respectively.

3 CPM under the Hamming Distance

In this section, we give a sketch of the proof of Theorem 1. We first partition P^2 into $8(k+1)$ blocks of length $\lfloor m/(8(k+1))\rfloor$ or $\lceil m/(8(k+1))\rceil$. A simple observation, formalized below, is that in every circular k-mismatch occurrence of P, we can find a block that is matched exactly. Further, such a block can be found in the initial quarter of the occurrence.

▶ **Observation 10.** *Let $m \leq j \leq n$ and $m < i \leq 2m$ be such that $\mathsf{hd}(T(j-m\,..\,j], P^2(i-m\,..\,i]) \leq k$. There exist integers ℓ and r with $3m/4 \leq r \leq \ell \leq m$ such that fragment $P' := P^2(i-\ell\,..\,i-r]$ is one of the blocks in P^2 and $T(j-\ell\,..\,j-r] = P'$.*

If P' is a block satisfying the two conditions in Observation 10, we say that j is a circular k-mismatch occurrence of P in T with *anchor* P', *anchored* at position $j-r$. Below, we fix one block $P' = P^2[a\,..\,b]$ and show how to compute all circular k-mismatch occurrences of P in T with anchor P'. We denote $|P'|$ by m'. In a preprocessing phase, we check whether P' is periodic, and if it is, we compute its period $\mathsf{per}(P')$ in $\mathcal{O}(m')$ time using Fact 6.

Assume first that P' is not periodic. Using real-time constant space pattern matching (Fact 4), we discover that $T(j-m'\,..\,j] = P'$ as soon as $T[j]$ arrives. Then, by Fact 5, there are $\mathcal{O}(k)$ exact occurrences of P' in any $\Theta(m)$-length fragment of T. We will spend $\mathcal{O}(m)$ time to treat each occurrence of P', that is, to extend the occurrence of P' into all circular k-mismatch occurrences that it anchors. The overall time complexity is $\mathcal{O}(nk)$. Crucially, since an anchor lies in the initial $m/4$ symbols of an occurrence, it is easy to de-amortize the extension algorithm during the arrival of the upcoming symbols.

For the extension of an occurrence of P', we merely extend it to the left and to the right until encountering k mismatches, see Figure 1a. These extensions and the mismatches can then be used to decide whether a given position is a circular k-mismatch occurrence, leading to the following intermediate result.

▶ **Lemma 11.** *If P' is not periodic, there is an online read-only algorithm that, upon reading $T[j]$, reports whether j is a circular k-mismatch occurrence of P with anchor P', processing each arriving symbol in $\mathcal{O}(k)$ time and using $\mathcal{O}(k^2)$ extra space.*

From now on assume that P' is periodic, and let $Q := P'[1\,..\,\mathsf{per}(P')]$. By Fact 5, occurrences of P' can be partitioned in arithmetic progressions with differences $|Q| = \mathsf{per}(P')$ so that every $\Theta(m)$-length substring of T is overlapped by $\mathcal{O}(k)$ of those. Consider one such progression $\{p_i\}_{i=0}^{s}$, where $p_i = p + i\cdot|Q|$. We start by introducing the following notation. For $i \in [0\,..\,s]$ and $j \in [1\,..\,n]$, define $a_{i,j}$ and $b_{i,j}$ so that $P^2(a_{i,j}\,..\,b_{i,j}]$ and $T(j-m\,..\,j]$ align P' with $T(p_i-m'\,..\,p_i]$. Formally, $a_{i,j} = a - [(p_i-m') - (j-m)]$ and $b_{i,j} = b + j - p_i$. During preprocessing, we compute a fragment $P'' := P^2[a'\,..\,b']$ by extending P' so that:

$$a' = \min\{i \in [1\,..\,a) : \mathsf{hd}(P^2[i\,..\,a), {}^*Q) \leq k+1\},$$
$$b' = \max\{i \in [a\,..\,2m] : \mathsf{hd}(P^2[a\,..\,i], Q^*) \leq k+1\}.$$

Additionally, we compute the following associated sets of mismatches: $M(P'') := \{i \in [a' \,..\, b'] : P[i] \neq Q^{\infty}[1 + (i - a) \pmod{|Q|}]\}$, $M_\ell(P'') := M(P'') \cap [a' \,..\, b]$, and $M_r(P'') = M(P'') \cap (b \,..\, b']$. We store $M(P'')$, $M_\ell(P'')$, and $M_r(P'')$ as sorted lists in $\mathcal{O}(k+1)$ space each. Similarly, we define an extension $T' := T[\ell' \,..\, r']$ of $T(p_0 - m' \,..\, p_0]$ such that:

$$\ell' = \min\{p_0 - m' - \min\{m/2, a - a'\} \leq i \leq p_0 - m' : \mathsf{hd}(T[i \,..\, p_0 - m'], {}^*Q) \leq k+1\},$$
$$r' = \max\{p_0 - m' < i \leq n : \mathsf{hd}(T(p_0 - m' \,..\, i], Q^*) \leq k+1\}.$$

Note that we extend $(p_0 - m' \,..\, p_0]$ to the left by at most $m/2$ symbols and at most as much as P' was extended to the left. T' is not constructed explicitly, it is only used to describe the algorithm. We further define the associated sets of mismatches: $M(T', j) = \{i \in [\max\{\ell', j - m + 1\} \,..\, \min\{r', j\}] : T[i] \neq Q[1 + (i - (p_0 - m')) \pmod{|Q|}]\}$, $M_\ell(T', j) = M(T', j) \cap (j - m \,..\, p_0]$, and $M_r(T', j) = M(T', j) \cap (p_0 \,..\, j]$. We call the elements of $M(P'')$ and $M(T', j)$ the *misperiods* of P'' and T', respectively. (See Figure 1 for an illustration.)

We next define a set $\mathcal{N}(j)$ that, intuitively, contains all integers i such that no two misperiods (one in $P^2(a_{i,j} \,..\, b_{i,j}]$ and one in $T(j - m \,..\, j]$) are aligned (see Figure 1b).

$$\mathcal{N}(j) = \left\{ i \in [0 \,..\, s] \;\middle|\; \begin{array}{l} (1)\ j - m \leq p_i - m' + 1 \leq p_i \leq j - \frac{3m}{4}, \\ (2)\ \forall x \in M(P'') \cap (a_{i,j}, b_{i,j}],\ x - a_{i,j} + (j - m) \notin M(T', j) \end{array} \right\}$$

Furthermore, let $\mathcal{A}(j)$ denote the "complement" of $\mathcal{N}(j)$, that is, the set of integers i such that at least one misperiod in $P(a_{i,j} \,..\, b_{i,j}]$ is aligned with a misperiod in $T(j - m \,..\, j]$.

$$\mathcal{A}(j) = \left\{ i \in [0 \,..\, s] \;\middle|\; \begin{array}{l} (1)\ j - m \leq p_i - m' + 1 \leq p_i \leq j - \frac{3m}{4}, \\ (2)\ \exists x \in M(P'') \text{ such that } x - a_{i,j} + (j - m) \in M(T', j) \end{array} \right\}$$

In what follows, we design two algorithms that decide whether j is a circular k-mismatch occurrence of P with anchor P' anchored at p_i, for $i \in \mathcal{N}(j)$ and for $i \in \mathcal{A}(j)$, respectively.

First consider $i \in \mathcal{N}(j)$. We show that in this case $P^2(a_{i,j} \,..\, b_{i,j}]$ is a fragment of $P^2[a' \,..\, b']$ and $T(j - m \,..\, j]$ is a fragment of $T[\ell' \,..\, r']$. Intuitively, this is because (since no misperiods are aligned) the Hamming distance equals the total number of misperiods spanned, and hence neither $P^2(a_{i,j} \,..\, b_{i,j}]$ nor $T(j - m \,..\, j]$ can span more than k misperiods.

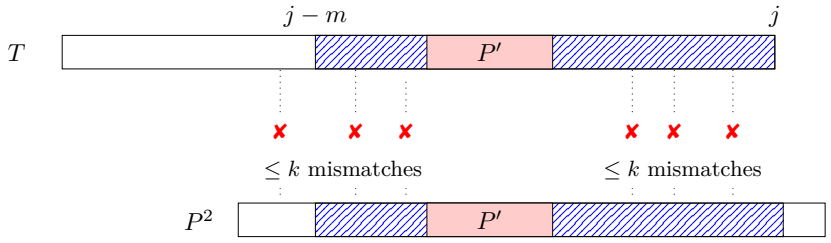
▶ **Lemma 12.** *Assume $i \in \mathcal{N}(j)$.*
1. *If $a' > a_{i,j}$, or $\ell' > j - m$, then $\mathsf{hd}(P^2(a_{i,j} \,..\, b_{i,j}], T(j - m \,..\, j]) > k$.*
2. *If $b' < b_{i,j}$, or $r' < j$, then $\mathsf{hd}(P^2(a_{i,j} \,..\, b_{i,j}], T(j - m \,..\, j]) > k$.*

It follows that it is enough to know the sets of misperiods in $P^2[a' \,..\, b']$ and $T[\ell' \,..\, r']$ to find all circular k-mismatch occurrence of P with anchor P' anchored at p_i, for $i \in \mathcal{N}(j)$: on a high level, the idea of the algorithm is to count the number of misperiods in $T(j - m \,..\, j]$ and $P^2(a_{i,j} \,..\, b_{i,j}]$ and take their sum, which equals the Hamming distance.

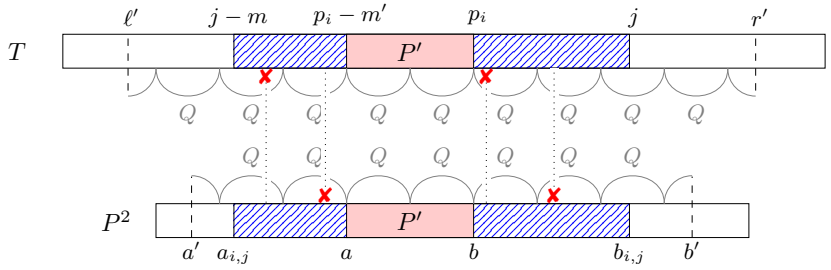
▶ **Lemma 13.** *Given $M(P'')$ and $M(T', j)$ there is an algorithm that, upon reading $T[j]$:*
1. *If there exists $i \in \mathcal{N}(j)$ such that $\mathsf{hd}(P^2(a_{i,j} \,..\, b_{i,j}], T(j - m \,..\, j]) \leq k$, answers YES;*
2. *If the algorithm answers YES, j is a circular k-mismatch occurrence of P.*

The algorithm takes $\mathcal{O}(k+1)$ time and $\mathcal{O}(1)$ extra space.

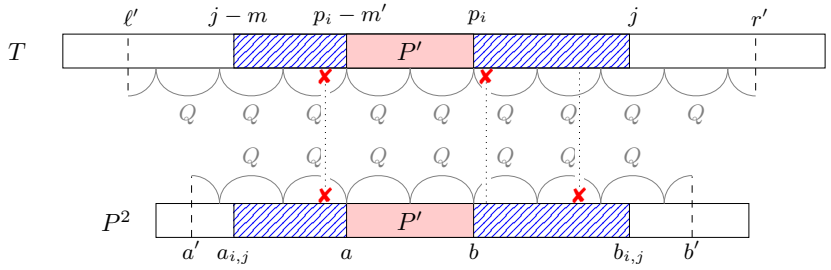
We now augment the above algorithm with the algorithm for detecting circular k-mismatch occurrences j with anchor P' anchored at p_i, for $i \in \mathcal{A}(j)$ to obtain our final algorithm for the case when P' is periodic. Intuitively, this subcase is easy, because the number of misperiods is $\mathcal{O}(k)$ both for the text and for the pattern, and hence the number of aligned pairs of misperiods is $\mathcal{O}(k^2)$. For each pair, we can extend to the left and to the right until



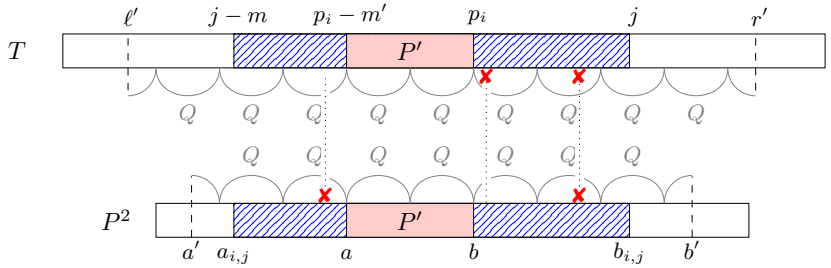
(a) Aperiodic P'.



(b) Periodic P': No aligned misperiods.



(c) Periodic P': Left-aligned misperiods.



(d) Periodic P': Right-aligned misperiods.

Figure 1 $T(j-m\,..\,j]$ is a candidate circular k-mismatch occurrence of P.

encountering k mismatches, similar to the non-periodic case. The only complication is that the aligned misperiods may be close to j, so we cannot de-amortize the left extension. To overcome this, we consider the following two subsets of $\mathcal{A}(j)$:

$$\mathcal{A}_\ell(j) = \{0 \le i \le s : \exists x, x \in M_\ell(P'') \text{ and } x - a_{i,j} + (j-m) \in M_\ell(T', j)\},$$
$$\mathcal{A}_r(j) = \{0 \le i \le s : \exists x, x \in M_r(P'') \text{ and } x - a_{i,j} + (j-m) \in M_r(T', j)\}.$$

If $\mathcal{A}_\ell(j) \neq \emptyset$, j is called *left-aligned*. (See Figure 1c.) Analogously, if $\mathcal{A}_r(j) \neq \emptyset$, j is called *right-aligned*. (See Figure 1d.) Left-aligned occurrences j can be treated analogously to the non-periodic case. For right-aligned occurrences, assume that the misperiods to the left of the fixed aligned pair of misperiods are not aligned. It follows that the Hamming distance between the left halves of the text and the pattern fragments equals the sum of the number of misperiods in the said fragments and hence can be computed efficiently.

▶ **Lemma 14.** *If P' is periodic, there is an online read-only algorithm that, upon reading $T[j]$:*
1. *If j is a circular k-mismatch occurrence of P with anchor P', returns YES;*
2. *If j is not a circular k-mismatch occurrence of P, returns NO.*

The algorithm takes $\mathcal{O}(k^3)$ time per symbol and $\mathcal{O}(k^4)$ extra space.

Theorem 1 follows directly from Lemmas 11 and 14.

4 CPM under the Edit Distance

In this section, we give a sketch of the proof of Theorem 2. We start by introducing necessary tools. We denote by $\mathsf{Occ}(X, Y)$ the set of *starting* positions of exact occurrences of a string X in a string Y. Further, we denote by $\mathsf{Occ}_k^E(X, Y)$ the set of *ending* positions of fragments of Y that are at edit distance at most k from X. (This asymmetry in notation simplifies the exposition.) Our algorithms use the following result of Bathie et al. [4] as a sub-routine:

▶ **Fact 15** ([4]). *Consider a pattern $P \in \Sigma^m$ and a text $T \in \Sigma^n$. There is an online read-only algorithm that computes $\mathsf{Occ}_k^E(P, T)$ using $\tilde{O}(k^4)$ space and $\tilde{O}(k^4)$ amortized time per symbol.*

During the course of the algorithm, we maintain optimal alignments between suffixes of T and fragments of a string Q^∞ for a primitive string Q using the following lemma, which is obtained by adapting the Landau–Vishkin algorithm [21].

▶ **Lemma 16.** *Consider an integer $d > 0$, a string R to which we have constant-time random access, and a string S arriving as a stream. There is a data structure that uses $\mathcal{O}(d^2)$ space, can be updated in $\mathcal{O}(d)$ time per arriving symbol of S, and supports the following queries in $\mathcal{O}(d)$ time when the j-th symbol of S is the last symbol that has arrived: given any $s \in [1\,..\,j]$ and any $r \in [1\,..\,|R|]$, return $\mathrm{ed}_d(S[1\,..\,s], R[1\,..\,r])$ and, if the answer is at most d, the breakpoint representation of an optimal alignment from $S[1\,..\,s]$ onto $R[1\,..\,r]$.*

In our algorithm, we compute occurrences of substrings of P^2 in T and extend them to k-edit circular occurrences of P. The following definition formalizes this concept:

▶ **Definition 17** ([11]). *A circular k-edit occurrence $T[p\,..\,p']$ of P is* anchored *at a position $i \in [p\,..\,p']$ (called an* anchor*) if $\mathrm{ed}(T[p\,..\,i), Y) + \mathrm{ed}(T[i\,..\,p'], X) \le k$, where $P = XY$ for some X, Y. We denote*

$$\mathtt{Anchored}_k(P, T, i) := \{p' : \text{there exists a circular } k\text{-edit occ. } T[p\,..\,p'] \text{ of } P \text{ anchored at } i\}.$$

Further, for an integer j we denote $\mathtt{Anchored}_k(P, T, i, j) = \mathtt{Anchored}_k(P, T, i) \cap [j, \infty)$.

Using Lemma 16, we can efficiently compute anchored occurrences:

▶ **Lemma 18** (Computing anchored occurrences). *Suppose that at the arrival of $T[j]$ we are given an integer $i \le j$. We can then report each position $j' \in \texttt{Anchored}_k(P, T, i, j)$ before symbol $T[j'+1]$ arrives in total time $\mathcal{O}(k^3 \cdot m)$ using $\mathcal{O}(k^2)$ extra space.*

4.1 Preprocessing the Pattern

We decompose the pattern into fragments $P_1 P_2 \cdots P_{k+2} = P$, where, for all i, $|P_i| = \lfloor m/(k+2) \rfloor$ or $|P_i| = \lceil m/(k+2) \rceil$. Let $\pi_i := \sum_{j<i} |P_i|$ and $\Pi_i := \mathrm{rot}^{\pi_i}(P)$. For each i such that $q_i := \mathsf{per}(P_i) \le m/999k^2$, we set $Q_i := P_i[1\,..\,q_i]$ and do the following:

1. We compute the longest prefix $R_i^{\mathrm{right}} = \Pi_i^2(m\,..\,r_i)$ of $\Pi_i^2(m\,..\,2m]$ that is at edit distance at most $8k$ to a prefix of Q_i^∞ using Lemma 16. We next argue that if $|R_i^{\mathrm{right}}| < m$, then we have $\mathrm{ed}(R_i^{\mathrm{right}}, {}^*Q_i^*) = 8k$. First, by the definitions, we have $\mathrm{ed}(R_i^{\mathrm{right}}, {}^*Q_i^*) \le \mathrm{ed}(R_i^{\mathrm{right}}, Q_i^*) = 8k$. Now, since R_i^{right} has a prefix consisting of at least $\lfloor \lfloor m/(k+2) \rfloor / (m/999k^2) \rfloor \ge 331k$ copies of Q_i, any alignment of cost $\mathrm{ed}(R_i^{\mathrm{right}}, {}^*Q_i^*) \le 8k$ matches at least one of said $331k$ copies exactly, and hence also matches exactly all the copies of Q_i preceding it. This implies that $\mathrm{ed}(R_i^{\mathrm{right}}, {}^*Q_i^*) = \mathrm{ed}(R_i^{\mathrm{right}}, Q_i^*)$.
2. We symmetrically compute the longest suffix $R_i^{\mathrm{left}} = \Pi_i^2[\ell_i\,..\,m+|P_i|]$ of $\Pi_i^2(|P_i|\,..\,m+|P_i|]$ that is at edit distance at most $8k$ to a suffix of $Q_i^{2m} Q_i^\infty[1\,..\,|P_i|]$. Symmetrically to before, if $|R_i^{\mathrm{left}}| < m$, then $\mathrm{ed}(R_i^{\mathrm{left}}, {}^*Q_i^*) = 8k$.
3. We set $V_i := \Pi_i[\ell_i\,..\,r_i)$ if $r_i - \ell_i \ge m$ and $V_i := \varepsilon$ otherwise.

▶ **Observation 19.** *Consider an m-length fragment F of Π_i^2 that contains $\Pi_i(m\,..\,m+|P_i|]$. Either F contains R_i^{right} or R_i^{left}, or F is contained in V_i.*

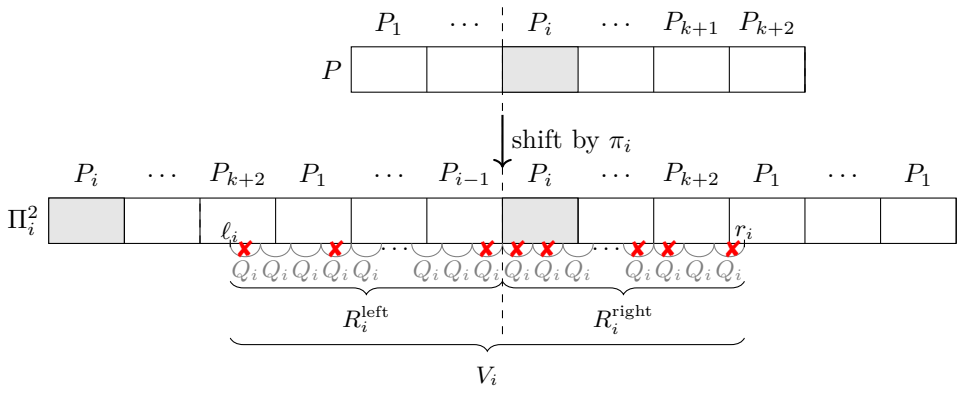


Figure 2 An illustration of fragments R_i^{left}, R_i^{right}, and V_i.

If $V_i \neq \varepsilon$, we also compute an alignment $\mathcal{A}_i$ of cost $\mathrm{ed}(V_i, {}^*Q_i^*)$ from V_i to a fragment of Q_i^∞ as follows. Observe that when the computations of R_i^{right} and R_i^{left} are concluded, Lemma 16 allows us to retrieve alignments from each of R_i^{right} and R_i^{left} to fragments of Q_i^∞ of costs $\mathrm{ed}(R_i^{\mathrm{right}}, {}^*Q_i^*)$ and $\mathrm{ed}(R_i^{\mathrm{left}}, {}^*Q_i^*)$, respectively. The facts that P_i contains at least $331k$ copies of Q_i and that $\mathrm{ed}(V_i, {}^*Q_i^*) \le 16k$ (which follows by the triangle inequality), imply that we can glue the two returned alignments together, as they must both align a common copy of Q_i in P_i exactly, that is, without any edits. The described preprocessing takes $\mathcal{O}(m \cdot k^4)$ time and uses $\mathcal{O}(k^4)$ extra space.

4.2 Reduction to the Approximately Periodic Case

Consider the following partition of P into $k+2$ blocks: $P = P_1P_2\cdots P_{k+2}$, where, for all i, $|P_i| = \lfloor m/(k+2)\rfloor$ or $|P_i| = \lceil m/(k+2)\rceil$. Any alignment of cost at most k between an m-length fragment of P^2 and a fragment of string T aligns one of the blocks exactly. This is because any m-length fragment of P^2 fully contains at least $k+1$ blocks and hence an alignment of cost at most k with a fragment of T must match at least one of these blocks exactly. In what follows, we focus on computing all k-edit circular occurrences of P in T for which there exists a witness alignment that matches P_1 exactly. To complete the solution, it suffices to perform a symmetric computation for each of the other P_is and merge the results (in a naive manner). Formally, we show how to compute a subset of $\mathsf{CircOcc}_k^E(P,T)$ that is a superset of

$$\mathcal{C} := \{y : \exists x,\, \exists a \in (|P_1|\,..\,m+1],\, \exists \text{ alignment } \mathcal{A} : P^2[a\,..\,a+m) \to T[x\,..\,y] \text{ of cost at most } k \text{ with } \mathcal{A}(P^2(m\,..\,m+|P_1|]) = P_1\}.$$

▶ **Observation 20.** $\mathcal{C} \subseteq \cup_{i\in\mathsf{Occ}(P_1,T)}\mathtt{Anchored}_k(P,T,i) \subseteq \mathsf{CircOcc}_k^E(P,T)$.

Henceforth, we drop the subscript from q_1, Q_1, V_1, R_1^{right}, and R_1^{left}.

In the case when $\mathsf{per}(P_1) > m/999k^2$, P_1 only has a few exact occurrences in every $\mathcal{O}(m)$-length fragment of T and we can efficiently extend those to k-edit circular occurrences of P using Lemma 18.

▶ **Lemma 21.** *If $\mathsf{per}(P_1) > m/999k^2$, we can compute $\mathcal{C}$ in $\mathcal{O}(k^5)$ amortized time using $\mathcal{O}(k^4)$ extra space.*

We now turn our attention to the case when $\mathsf{per}(P_1) \le m/999k^2$. Our goal is to reduce the k-EDIT CPM to the problem of computing the ending positions of the k-edit occurrences of all m-length substrings of V in T. The latter problem is formalized below as the PERIODICFRAG problem. Our reduction crucially relies on the following fact and is similar in spirit to a reduction from [11].

▶ **Fact 22** ([9, Corollary 5.18])**.** *Let P denote a pattern of length m, let T denote a string of length n, and let $k \le m$ denote a non-negative integer. Suppose that there are a positive integer $d \ge 2k$ and a primitive string Q with $|Q| \le m/8d$ and $\mathrm{ed}(P, {}^*Q^*) = d$. Then $\lfloor|\mathsf{Occ}_k^E(P,T)|/d\rfloor \le 1216\cdot n/m\cdot d$.*

▶ **Lemma 23.** *If R^{right} (R^{left}) is of length strictly smaller than m, we can compute a set $A \subseteq \mathsf{CircOcc}_k^E(P,T)$ that is a superset of all k-edit occurrences of m-length fragments of P^2 that contain R^{right} (resp. R^{left}) in $\tilde{\mathcal{O}}(k^7)$ amortized time per symbol using $\tilde{\mathcal{O}}(k^6)$ extra space.*

Proof. Let us describe the computations for R^{right}, assuming that it is of length smaller than m. The computations for R^{left} are symmetric.

We follow a procedure similar to that in the proof of Lemma 21. Let us set $d := 8k$, noting that $|Q| \le m/8d$ and $\mathrm{ed}(R^{\text{right}}, {}^*Q^*) = d$, and hence the conditions of Fact 22 are satisfied. A direct application of Fact 22 then implies that R^{right} has $\mathcal{O}(k^3)$ k-edit occurrences in any $\mathcal{O}(m)$-length fragment of T since R^{right} contains P_1 and is hence of length $\Omega(|P_1|) = \Omega(m/k)$.

We compute the right endpoints of all k-edit occurrences of R^{right} using Fact 15 in time and space $\tilde{\mathcal{O}}(k^4)$. Then, for each computed $x \in \mathsf{Occ}_k^E(R^{\text{right}}, T)$, we invoke Lemma 18 for each $i \in x - |R^{\text{right}}| + 1 + [-k\,..\,k]$; the considered positions are the only possible starting positions of suffixes of $T[1\,..\,x]$ that are at edit distance at most k from R^{right}.

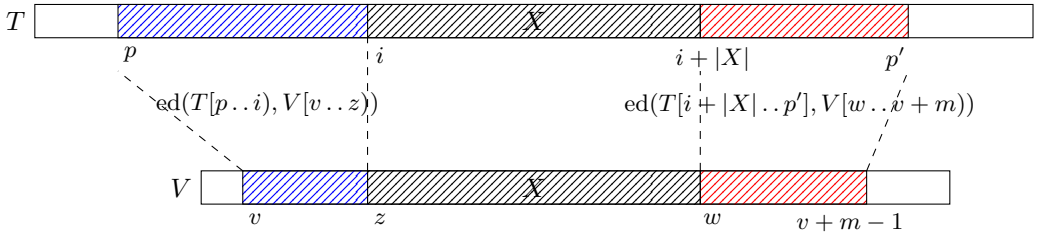


Figure 3 An illustration of Definition 24 with $\mathrm{ed}(T[p\,..\,i), V[v\,..\,z)) + \mathrm{ed}(T[i+|X|\,..\,p'), V[w\,..\,v+m)) \leq k$. We have $p' \in \mathsf{blockAnchored}_{k,m}(V, T, X)$ and, if we also have $i \in I$, then $p' \in \mathsf{blockAnchored}_{k,m}(V, T, X, I)$.

Overall, we invoke Lemma 18 $\mathcal{O}(k^4)$ times for every consecutive $2m$ symbols of the text that arrive, while each invocation becomes inactive after the arrival of at most m symbols, and the first invocation happens after reading $\Omega(m/k)$ symbols of the text. Thus, the described algorithm spends $\mathcal{O}(k^7)$ amortized time per symbol of the text and uses $\mathcal{O}(k^6)$ space throughout its course. This concludes the proof of the lemma. ◀

Next, we formalize our reduction to the PeriodicFrag problem, which is a variant of the PeriodicSubMatch problem from [11]. For convenience, we first introduce a variant of the definition of anchored occurrences, illustrated in Figure 3.

▶ **Definition 24.** *Consider integers m and $k \leq m$, strings V and T, and a fragment $X = V[z\,..\,w)$. A position p' of T is in $\mathsf{blockAnchored}_{k,m}(V, T, X)$ (in $\mathsf{blockAnchored}_{k,m}(V, T, X, I)$ for a set $I \subseteq \mathbb{Z}$) if and only if there exist positions p and i ($i \in I$, respectively) of T and a position v of V such that $T[i\,..\,i + |X|) = X$ and $\mathrm{ed}(T[p\,..\,i), V[v\,..\,z)) + \mathrm{ed}(T[i + |X|\,..\,p'], V[w\,..\,v + m)) \leq k$.*

PeriodicFrag Problem

Input:
- Positive integers m and $k \leq m$;
- a primitive string Q of length at most $m/999k^2$;
- a string P of length m, and a fragment V of P^2 satisfying $|V| \in [m\,..\,2m)$ and $\mathrm{ed}(V, {}^*Q^*) \leq 16k$;
- a fragment $P_1 = V[z\,..\,z + m_1)$ of V that is a prefix of Q^∞, where $m_1 \leq \lfloor m/(k+2) \rfloor$;
- an alignment $\mathcal{A}_\star : V \to Q^\infty[x_\star\,..\,y_\star)$ of cost $\mathrm{ed}(V, {}^*Q^*)$;
- a text T of length n.

Output: A set $\mathcal{I}$ such that $\mathsf{blockAnchored}_{k,m}(V, T, P_1) \subseteq \mathcal{I} \subseteq \mathsf{CircOcc}_k^E(P, T)$.

▶ **Lemma 25.** *Suppose that there exists an online read-only algorithm that solves the* PeriodicFrag *problem in $\mathcal{O}(u(k))$ amortized time per symbol using $\mathcal{O}(s(k))$ space. Then, there exists an online read-only algorithm for the k-edit* CPM *problem that takes $\tilde{\mathcal{O}}(k \cdot u(k) + k^8)$ amortized time per symbol and uses $\tilde{\mathcal{O}}(k \cdot s(k) + k^7)$ space.*

Proof. Any circular k-edit occurrence of P in T must match at least one of the $k+2$ blocks P_i exactly. Thus, finding all k-edit circular occurrences of P reduces to taking the union of the occurrences anchored at P_i over all $i \in [1\,..\,k+2]$. For each block P_i, we proceed as follows.

If $\mathsf{per}(P_i) > m/999k^2$, we use Lemma 21 to compute all such occurrences in $\mathcal{O}(k^5)$ amortized time per symbol using $\mathcal{O}(k^4)$ extra space.

In the case when $\mathsf{per}(P_i) \leq m/999k^2$, by combining Observation 19 and Lemma 23, we conclude that, by spending $\tilde{\mathcal{O}}(k^7)$ time per symbol and using $\tilde{\mathcal{O}}(k^6)$ extra space, we are left with computing the k-edit occurrences of the m-length fragments of V_i in T. Due to the definition of V_i and our precomputations, we arrive at an instance of the PeriodicFrag problem. By our assumption, this can be solved using $\mathcal{O}(u(k))$ amortized time per symbol and $\mathcal{O}(s(k))$ space.

We obtain the claimed complexities by summing the above bounds over all $k+2$ blocks. ◀

The following lemma, together with Lemma 25, yield our efficient online read-only algorithm for the k-edit CPM problem, encapsulated in Theorem 2. The remainder of this section is devoted to providing a high-level description of its proof.

▶ **Lemma 26.** *There is an online read-only algorithm that solves the* PeriodicFrag *problem in $\tilde{\mathcal{O}}(k^{10})$ amortized time per symbol using $\tilde{\mathcal{O}}(k^7)$ space.*

We use the pattern matching algorithm of Fact 4 to compute all exact occurrences of P_1 in T. Due to Fact 5, the exact occurrences of P_1 in any $\mathcal{O}(m)$-length substring of T form $\mathcal{O}(k^2)$ arithmetic progressions, each with period $q := |Q| = \mathsf{per}(P_1)$. In what follows, we focus on one such progression, say $\{\iota + j \cdot q : j \in [0\,..\,e]\}$ explaining how to compute a superset of $\mathsf{blockAnchored}_{k,m}(V, T, P_1, [\iota\,..\,\iota + e \cdot q])$ that is a subset of $\mathsf{CircOcc}_k^E(P, T)$ in $k^{\mathcal{O}(1)} \cdot q(e+1)$ total time using $k^{\mathcal{O}(1)}$ extra space. By performing such computations for each computed arithmetic progression of occurrences of P_1, we compute the desired output within the bounds of Theorem 2.

▶ **Definition 27.** *We call a string X* highly periodic *if $\mathrm{ed}(X, {}^*Q^*) \leq 34k$ and $|X| \geq 100qk$.*

The fact that the edit distance satisfies the triangle inequality implies the following.

▶ **Observation 28.** *Any m-length fragment F_V of V is highly periodic. Any fragment F_T of T that is at edit distance at most k from F_V satisfies $\mathrm{ed}(F_T, {}^*Q^*) \leq 17k$, and hence is highly periodic as well.*

The above observation allows us to use a strategy similar to that of Section 3 to solve the PeriodicFrag problem. However, the notion of misperiods is insufficient for dealing with approximate pattern matching under the edit distance, due to the "shifts" that insertions and deletions imply. We instead rely on a variant of locked fragments that was introduced by Cole and Hariharan in [13], adapted in [9] by Charalampopoulos et al., and exploited for circular pattern matching under the edit distance by Charalampopoulos et al. [11]. We call our stronger variant of locked fragments *super-locked* as they guarantee additional properties at the cost of a multiplicative $k^{\mathcal{O}(1)}$ factor in the complexity.

▶ **Definition 29.** *Let S be a string, Q a primitive string of length q, and k a positive integer. Let $\mathcal{A} : S \to Q^\infty[x\,..\,y)$ denote an alignment of cost $\mathrm{ed}(S, {}^*Q^*)$, where $x \in [0\,..\,|Q|)$. $\mathcal{A}$ yields a partition $S = S_0 \cdots S_s$ with $s = \lceil (y-1)/|Q| \rceil$ such that $\mathrm{ed}(S, Q^\infty[x\,..\,y)) = \sum_{i=0}^{s} \Delta_i$, where*

$$\Delta_i = \begin{cases} \mathrm{ed}(S_0, Q[x\,..\,|Q|)) & \text{if } i = 0, \\ \mathrm{ed}(S_i, Q) & \text{if } 0 < i < s, \\ \mathrm{ed}(S_s, Q[0\,..\,y - s|Q|)) & \text{if } i = s. \end{cases}$$

For each i such that $\Delta_i > 0$, mark S_{i+j} for all $j \in [-36k\,..\,36k] \cap [-i\,..\,s-i]$. Then, the maximal contiguous marked fragments of S are designated as the super-locked fragments of S with respect to Q and $\mathcal{A}$. We denote the set of these super-locked fragments as $\mathsf{Slocked}(S, k, Q, \mathcal{A})$.

The useful property of super-locked fragments is that we can always assume that an optimal alignment of S with a substring of Q^∞ aligns each super-locked fragment with an integer power of Q. In particular, the following holds:

▶ **Corollary 30.** *Let $T = Q[p_T\,..]Q^{\alpha_1}L_1Q^{\alpha_2}\cdots L_tQ^{\alpha_{t+1}}Q[..\,s_T)$ be a highly periodic string with super-locked fragments $L_1, \ldots, L_t$. We have* $\mathrm{ed}(T, {}^*Q^*) = \sum_{i=1}^{t} \mathrm{ed}(L_i, {}^*Q^*)$.

We show that super-locked fragments have total length $\mathcal{O}(k^2q)$ and can be computed efficiently both for V and for highly periodic fragments of T using $\mathcal{O}(k^3)$ amortized time per symbol and $\mathcal{O}(k^3)$ space. Our approach maintains both an optimal alignment from a suffix of (the seen prefix of) T to a substring of Q^∞ as well as the set of super-locked fragments of said suffix. The crucial observation here is that an optimal alignment is entirely defined by its restriction to the super-locked fragments.

From now on, fix one progression of exact occurrences of P_1 in T, say $\{\iota + j\cdot q : j \in [0\,..\,e]\}$. We show how to compute a superset of $\mathsf{blockAnchored}_{k,m}(V, T, P_1, [\iota\,..\,\iota + e\cdot q])$ that is a subset of $\mathsf{CircOcc}_k^E(P, T)$ in $k^{\mathcal{O}(1)}\cdot q(e+1)$ total time using $k^{\mathcal{O}(1)}$ extra space. To do this, we partition $\mathsf{blockAnchored}_{k,m}(V, T, P_1, [\iota\,..\,\iota + e\cdot q])$ into two subsets. The first subset is comprised only of those positions that satisfy both of the following:

1. There exist a suffix F_1 of the text and an m-length fragment F_2 of V that admit an alignment of cost $\leq k$ that matches P_1 and $T[\iota + j\cdot q\,..\,\iota + j\cdot q + |P_1|)$ for some $j \in [0\,..\,e]$;
2. The super-locked fragments of F_1 and F_2 (almost) *overlap*.

To compute (a superset of) the first subset, we use a strategy similar to the aperiodic case for the Hamming distance. Namely, we show that the number of positions where the super-locked fragments almost overlap is $k^{\mathcal{O}(1)}$, and we verify each of them using Lemma 16 in time $k^{\mathcal{O}(1)}$.

The second set is the complement of the first one, and computing it is much more involved. The main challenge that we have to overcome is that this subset can be quite large, and we cannot afford to treat each of its elements separately. However, we crucially show that it suffices to consider a $k^{\mathcal{O}(1)}$-size subset of the elements. Conceptually, if two m-length fragments of V (a) have the same super-locked fragments, (b) do not overlap with the $(m+k)$-length suffix of $T[\iota\,..\,\zeta]$, and (c) are "synchronized" with respect to Q, it suffices to consider just one of them: in fact, the edit distance only depends on the cost of aligning the super-locked fragments against some powers of Q. For each arriving symbol, we are thus left with computing the edit distances between only $k^{\mathcal{O}(1)}$ pairs of strings, which can be achieved in $k^{\mathcal{O}(1)}$ amortized time per symbol and $k^{\mathcal{O}(1)}$ space.

References

1 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proc. of FOCS*, pages 377–386, 2010. doi:10.1109/FOCS.2010.43.

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

3 Gabriel Bathie, Panagiotis Charalampopoulos, and Tatiana Starikovskaya. Internal pattern matching in small space and applications. In *Proc. of CPM*, pages 4:1–4:20, 2024. doi:10.4230/LIPIcs.CPM.2024.4.

4 Gabriel Bathie, Tomasz Kociumaka, and Tatiana Starikovskaya. Small-space algorithms for the online language distance problem for palindromes and squares. In *Proc. of ISAAC*, pages 10:1–10:17, 2023. doi:10.4230/LIPIcs.ISAAC.2023.10.

5 Dany Breslauer, Roberto Grossi, and Filippo Mignosi. Simple real-time constant-space string matching. In *Proc. of CPM*, pages 173–183, 2011. doi:10.1007/978-3-642-21458-5_16.

6 Panagiotis Charalampopoulos, Taha El Ghazi, Jonas Ellert, Paweł Gawrychowski, and Tatiana Starikovskaya. Suffix random access via function inversion: A key for asymmetric streaming string algorithms, 2026. Accepted to ICALP'2026. doi:10.48550/arXiv.2604.19371.

7 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Circular pattern matching with k mismatches. *J. Comput. Syst. Sci.*, 115:73–85, 2021. doi:10.1016/J.JCSS.2020.07.003.

8 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Walen, and Wiktor Zuba. Approximate circular pattern matching. In *Proc. of ESA*, pages 35:1–35:19, 2022. Full version: arXiv:2208.08915. doi:10.4230/LIPIcs.ESA.2022.35.

9 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *Proc. of FOCS*, pages 978–989, 2020. Full version: arXiv:2004.08350. doi:10.1109/FOCS46700.2020.00095.

10 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 *Proc. of FOCS*, pages 698–707, 2022. Full version: arXiv:2204.03087. doi:10.1109/FOCS54457.2022.00072.

11 Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Tomasz Walen, and Wiktor Zuba. Approximate circular pattern matching under edit distance. In *Proc. of STACS*, pages 24:1–24:22, 2024. doi:10.4230/LIPIcs.STACS.2024.24.

12 Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Bit-Parallel Algorithms for Exact Circular String Matching. *Comput. J.*, 57(5):731–743, March 2013. doi:10.1093/comjnl/bxt023.

13 Richard Cole and Ramesh Hariharan. Approximate String Matching: A Simpler Faster Algorithm. *SIAM J. Comput.*, 31(6):1761–1782, 2002. doi:10.1137/S0097539700370527.

14 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proc. Am. Math. Soc.*, 16:109–114, 1965. doi:10.1090/S0002-9939-1965-0174934-9.

15 Johannes Fischer, Travis Gagie, Pawel Gawrychowski, and Tomasz Kociumaka. Approximating LZ77 via small-space multiple-pattern matching. In *Proc. of ESA*, pages 533–544, 2015. Full version available at arXiv:1504.06647. doi:10.1007/978-3-662-48350-3_45.

16 Kimmo Fredriksson and Szymon Grabowski. Average-optimal string matching. *J. Discrete Algorithms*, 7(4):579–594, 2009. doi:10.1016/j.jda.2008.09.001.

17 Roberto Grossi, Costas S. Iliopoulos, Robert Mercas, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Circular sequence comparison: algorithms and applications. *Algorithms Mol. Biol.*, 11:12, 2016. doi:10.1186/S13015-016-0076-6.

18 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, and Sharma V. Thankachan. Space-efficient construction algorithm for the circular suffix tree. In *Proc. of CPM*, pages 142–152, 2013. doi:10.1007/978-3-642-38905-4_15.

19 Wing-Kai Hon, Chen-Hua Lu, Rahul Shah, and Sharma V. Thankachan. Succinct indexes for circular patterns. In *Proc. of ISAAC*, pages 673–682, 2011. doi:10.1007/978-3-642-25591-5_69.

20 Costas S. Iliopoulos, Solon P. Pissis, and M. Sohel Rahman. Searching and indexing circular patterns. In *Algorithms for Next-Generation Sequencing Data: Techniques, Approaches, and Applications*, pages 77–90. Springer, 2017. doi:10.1007/978-3-319-59826-0_3.

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

22 M. Lothaire. *Applied Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005.
23 Michael E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proc. of SODA*, pages 1698–1709, 2013. doi:10.1137/1.9781611973105.122.
24 Robert Susik, Szymon Grabowski, and Sebastian Deorowicz. Fast and simple circular pattern matching. In *Proc. of ICMMI (Man-Machine Interactions 3)*, pages 537–544, 2013. doi:10.1007/978-3-319-02309-0_59.

A Asymmetric Streaming Algorithms

[6] showed a black box for converting online read-only algorithms to asymmetric streaming ones for a wide class of pattern matching problems:

▶ **Definition 31** (Generic pattern matching, [6, Definition 7.1]). *For an integer $z \geq 0$, a pattern matching problem is z-generic if it satisfies the following property. For a pattern $P[1\,..\,m]$ and a text $T[1\,..\,n]$, every occurrence of P in T is a fragment of T that equals a concatenation of at most $z+1$ substrings of P and at most z symbols.*

As already noted in [6], circular pattern matching is 2 generic and k-approximate pattern matching under both the Hamming and edit distances is $(2+2k)$-generic.

▶ **Fact 32** ([6, Corollary 7.4]). *Consider integers $z \geq 0$ and $\tau > 0$. Suppose that, for a z-generic pattern matching problem with pattern $P[1\,..\,m]$, there exists an online read-only algorithm $\mathcal{A}$ that uses $s(m,\tau)$ extra space and processes a text $T[1\,..\,n]$ in $g(m,\tau)$ worst-case (or amortized) time per symbol, where f and g are functions that are non-decreasing in their parameters. Then, there is an asymmetric streaming algorithm that, given τ, random access to P, and streaming access to T, solves the given z-generic pattern matching problem for P and T using $\tilde{\mathcal{O}}(s(m,\tau)+\tau)$ extra space and $\tilde{\mathcal{O}}(k^3m^3/\tau^6 + g(m,\tau))$ worst-case (resp. amortized) time per symbol of T.*

A direct application of Fact 32 to our online read-only algorithms for approximate circular pattern matching, encapsulated in Theorems 1 and 2, yields Corollary 3.