

Approximate Circular Pattern Matching

Panagiotis Charalampopoulos  

Birkbeck, University of London, UK
Reichman University, Herzliya, Israel

Tomasz Kociumaka  

Max Planck Institute for Informatics, Saarbrücken, Germany

Jakub Radoszewski  

University of Warsaw, Poland
Samsung R&D, Warsaw, Poland

Solon P. Pissis  

CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Wojciech Rytter  

University of Warsaw, Poland

Tomasz Waleń  

University of Warsaw, Poland

Wiktor Zuba  

CWI, Amsterdam, The Netherlands

Abstract

We investigate the complexity of approximate *circular pattern matching* (CPM, in short) under the *Hamming* and *edit* distance. Under each of these two basic metrics, 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 (called occurrences) of fragments of T that are at distance at most k from some cyclic rotation of P . In the decision version of the problem, we are to check if there is any such occurrence. All previous results for approximate CPM were either average-case upper bounds or heuristics, with the exception of the work of Charalampopoulos et al. [CKP⁺, JCSS'21], who considered only the Hamming distance. For the reporting version of the approximate CPM problem, under the Hamming distance we improve upon the main algorithm of [CKP⁺, JCSS'21] from $\mathcal{O}(n + (n/m) \cdot k^4)$ to $\mathcal{O}(n + (n/m) \cdot k^3 \log \log k)$ time; for the edit distance, we give an $\mathcal{O}(nk^2)$ -time algorithm. Notably, for the decision versions and wide parameter-ranges, we give algorithms whose complexities are almost identical to the state-of-the-art for standard (i.e., non-circular) approximate pattern matching:

- For the decision version of the approximate CPM problem under the Hamming distance, we obtain an $\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$ -time algorithm, which works in $\mathcal{O}(n)$ time whenever $k = \mathcal{O}(\sqrt{m \log \log m / \log m})$. In comparison, the fastest algorithm for the standard counterpart of the problem, by Chan et al. [CGKKP, STOC'20], runs in $\mathcal{O}(n)$ time only for $k = \mathcal{O}(\sqrt{m})$. We achieve this result via a reduction to a geometric problem by building on ideas from [CKP⁺, JCSS'21] and Charalampopoulos et al. [CKW, FOCS'20].
- For the decision version of the approximate CPM problem under the edit distance, the $\mathcal{O}(nk \log^3 k)$ runtime of our algorithm near matches the $\mathcal{O}(nk)$ runtime of the Landau–Vishkin algorithm [LV, J. Algorithms'89] for approximate pattern matching under edit distance; the latter algorithm remains the fastest known for $k = \Omega(m^{2/5})$. As a stepping stone, we propose an $\mathcal{O}(nk \log^3 k)$ -time algorithm for solving the Longest Prefix k' -Approximate Match problem, proposed by Landau et al. [LMS, SICOMP'98], for all $k' \in \{1, \dots, k\}$. Our algorithm is based on Tiskin's theory of *seaweeds* [Tiskin, Math. Comput. Sci.'08], with recent advancements (see Charalampopoulos et al. [CKW, FOCS'22]), and on exploiting the seaweeds' relation to *Monge matrices*.

In contrast, we obtain a conditional lower bound that suggests a polynomial separation between approximate CPM under the Hamming distance over the binary alphabet and its non-circular counterpart. We also show that a strongly subquadratic-time algorithm for the decision version of approximate CPM under edit distance would refute the Strong Exponential Time Hypothesis.



© Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 35; pp. 35:1–35:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Pattern matching

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

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.35

Related Version *Full Version*: <http://arxiv.org/abs/2208.08915>

Funding *Panagiotis Charalampopoulos*: Supported by Israel Science Foundation (ISF) grant 810/21 when the work that led to this paper was conducted.

Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

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.

Tomasz Walęń: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Wiktor Zuba: Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

Acknowledgements We thank Paweł Gawrychowski and Oren Weimann for suggesting the strategy for the proof of Lemma 29.

1 Introduction

Pattern matching is one of the most widely-studied problems in computer science. Given a string P of length m , known as the *pattern*, and a string T of length $n \geq m$, known as the *text*, the task is to compute all occurrences of P in T . In the standard setting, the matching relation between P and the fragments of T assumes that the leftmost and rightmost positions of the pattern are conceptually important. In many real-world applications, however, any rotation (cyclic shift) of P is a relevant pattern. For instance, in bioinformatics [3, 6, 27, 33], the position where a sequence starts can be totally arbitrary due to arbitrariness in the sequencing of a circular molecular structure or due to inconsistencies introduced into sequence databases as a result of different linearisation standards [3]. In image processing [2, 40, 41, 42], the contours of a shape may be represented through a directional chain code; the latter can be interpreted as a cyclic sequence if the orientation of the image is not important [2].

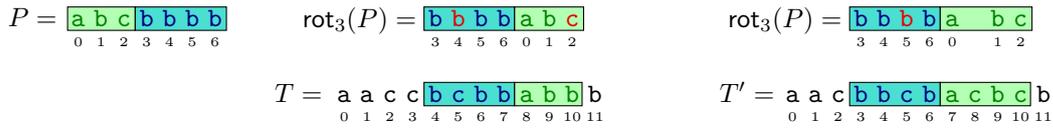
With such scenarios in mind, when matching a pattern P against a text T , one is interested in computing all fragments of T that match some rotation of P . Let us introduce necessary basic notation. The positions of a string U are numbered from 0 to $|U| - 1$, with $U[i]$ denoting the i -th letter, and $U[i..j] = U[i..j+1)$ denoting the substring $U[i] \cdots U[j]$, which is empty if $i > j$. We now formally define the circular pattern matching problem.

CIRCULAR PATTERN MATCHING (CPM)

Input: A text T of length n and a pattern P of length m .

Output: The set $\{i : T[i..i+m) = P' \text{ for some rotation } P' \text{ of } P\}$.

A textbook solution for CPM works in $\mathcal{O}(n \log \sigma)$ time (or $\mathcal{O}(n)$ time with randomization), where σ is the alphabet’s size, using the suffix automaton of $P \cdot P$ [39]. There is a simple deterministic $\mathcal{O}(n)$ -time algorithm which we discuss in the full version. Many practically fast algorithms for CPM also exist; see [17, 23, 43] and references therein. For the indexing version of the CPM problem (searching a collection of circular patterns), see [31, 32, 33].



■ **Figure 1** For a pattern $P = UV$, with $|U| = x$, we denote the rotation VU of P by $\text{rot}_x(P)$. Middle: a circular 2-mismatch occurrence of pattern P (left) at position 4 in text T . Right: a circular 2-edit occurrence of pattern P with the same rotation at position 3 in text T' (note that there is no circular 2-mismatch occurrence of P in T' at this position).

As in the standard pattern matching setting, a single surplus or missing letter in P or in T may result in many occurrences being missed. In bioinformatics, this may correspond to a single-nucleotide polymorphism; in image processing, this may correspond to data corruption. Thus, a relatively large body of works has been devoted to practically fast algorithms for *approximate* CPM; see [2, 4, 5, 7, 8, 24, 29, 30] and references therein. All previous results for approximate CPM were either average-case upper bounds or heuristics, with the exception of the work of Charalampopoulos et al. [14].

Here, like in the previous works on approximate CPM, we consider two well-known approximate matching relations of two strings U and V : the Hamming distance, denoted as $\delta_H(U, V)$ (the number of mismatches for two equal-length strings, otherwise equal to ∞), and the edit distance $\delta_E(U, V)$ (the minimum number of insertions, deletions and substitutions required to transform U to V). For two strings U and V , an integer $k > 0$, and a distance function d on strings, we write $U \stackrel{d}{=} V$ if $d(U, V) \leq k$ and $U \stackrel{d}{\approx} V$ if there exists a rotation U' of U such that $U' \stackrel{d}{=} V$. We define $\text{CircOcc}_k^d(P, T) = \{i : P \stackrel{d}{\approx} T[i..p] \text{ for some } p \geq i\}$; we call it the set of *circular k -mismatch (k -edit) occurrences of P in T* if $d = \delta_H$ ($d = \delta_E$, respectively). We omit the d -superscript when it is clear from the context. We next formally define four variants of k -approximate CPM; see Figure 1 for an example.

k -APPROXIMATE CPM: k -MISMATCH CPM AND k -EDIT CPM

Input: A text T of length n , a pattern P of length m , a positive integer k , and a distance function d : $d = \delta_H$ for k -Mismatch CPM and $d = \delta_E$ for k -Edit CPM.

Output: (**Reporting**) $\text{CircOcc}_k^d(P, T)$.
 (**Decision**) Any position $i \in \text{CircOcc}_k^d(P, T)$, if it exists.

Our upper bounds for k -Mismatch CPM. A summary of the best previous and our new worst-case upper bounds on approximate CPM for strings over a polynomially-bounded integer alphabet (we omit “polynomially-bounded” henceforth) is provided in Table 1.

■ **Table 1** Comparison of previous upper bounds and our results on k -approximate CPM.

Distance Metric	Time Complexity	Version	Reference
Hamming distance	$\mathcal{O}(nk)$	reporting	[14]
	$\mathcal{O}(n + (n/m) \cdot k^4)$		
	$\mathcal{O}(n + (n/m) \cdot k^3 \log \log k)$	reporting	this work
	$\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$	decision	
Edit distance	$\mathcal{O}(nk^2)$	reporting	this work
	$\mathcal{O}(nk \log^3 k)$	decision	

35:4 Approximate Circular Pattern Matching

In Section 2 we prove the following results.

► **Theorem 1.** *The reporting version of k -Mismatch CPM for strings over an integer alphabet can be solved in $\mathcal{O}(n + (n/m) \cdot k^3 \log \log k + \text{Output})$ time.*

► **Theorem 2.** *The decision version of k -Mismatch CPM for strings over an integer alphabet can be solved in $\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$ time.*

Our upper bounds for k -Edit CPM. A proof of the following theorem, based on the classic Landau–Vishkin algorithm [38], is given in Section 3.

► **Theorem 3.** *The reporting version of k -Edit CPM for strings over an integer alphabet can be solved in $\mathcal{O}(nk^2)$ time.*

We reduce the decision version of k -Edit CPM to the following problem.

LONGEST PREFIX k -APPROXIMATE MATCH (k -LPAM)

Input: A text T of length n and a pattern P .

Output: An array $\text{LPref}_k[0..n]$ such that $\text{LPref}_k[i]$ is the length of the longest prefix of P that matches a prefix of $T[i..n)$ with at most k edits.

Specifically, we introduce a problem called *All- k -LPAM* that consists in solving k' -LPAM for all $k' \in [0..k]$. We show that k -Edit CPM can be reduced to All- k -LPAM on the same pattern and text and on the reversed pattern and text. Landau et al. [37] gave an $\mathcal{O}(nk)$ -time solution to k -LPAM, which provides an $\mathcal{O}(nk^2)$ -time solution to All- k -LPAM. In Section 4 we show the following result for All- k -LPAM (Theorem 4) which implies Theorem 5; All- k -LPAM can find further applications in approximate pattern matching; for example see [9].

► **Theorem 4.** *All- k -LPAM for strings over an integer alphabet can be solved in $\mathcal{O}(nk \log^3 k)$ time.*

► **Theorem 5.** *The decision version of k -Edit CPM for strings over an integer alphabet can be solved in $\mathcal{O}(nk \log^3 k)$ time.*

The complexities of our algorithms for the decision versions of k -Mismatch and k -Edit CPM match, up to $\log^{\mathcal{O}(1)} k$ factors, the complexities of some of the fastest known algorithms for pattern matching with up to k mismatches [12, 15, 21, 26] and edits [38], respectively.

In [37], an algorithm for a weaker problem of computing, given two strings U and V each of length at most n , the rotation of U with the minimum edit distance to V is given. The algorithm works in $\mathcal{O}(ne)$ time, where e is the minimum edit distance achieved.

Our conditional lower bounds. We reduce known problems to approximate CPM, as shown in Table 2.

■ **Table 2** Our conditional lower bounds for approximate CPM for alphabets of constant size.

Problem	Conditioned on	Complexity	Reference
Mismatch-CPM (unbounded)	BJI	$\Omega(n^{1.5-\varepsilon})$ for all const. $\varepsilon > 0$	Theorem 30
k -Edit CPM (decision)	SETH	$\Omega(n^{2-\varepsilon})$ for all const. $\varepsilon > 0$	Theorem 31

For the Hamming distance, we consider the Mismatch-CPM problem where the number of allowed mismatches is unbounded (see Section 5 for a precise definition). The breakthrough work constructing a Binary Jumbled Index (BJI) in $\mathcal{O}(n^{1.859})$ time [13] was very recently improved to $\mathcal{O}(n^{1.5} \log^{\mathcal{O}(1)} n)$ time [18]. We show that obtaining an $\mathcal{O}(n^{1.5-\varepsilon})$ -time algorithm, for any constant $\varepsilon > 0$, for Mismatch-CPM over the binary alphabet would require a further improvement to BJI. A similar problem of (non-circular) pattern matching with mismatches has a classic $\mathcal{O}(n \log n)$ -time solution for constant-size alphabets using convolutions [22]; and the fastest known solution for a general alphabet works in $\mathcal{O}(n^{1.5} \sqrt{\log n})$ time [1, 36]. Our conditional lower bound for k -Edit CPM is based on the conditional hardness of computing the edit distance of two binary strings [10], which in turn is based on the Strong Exponential Time Hypothesis (SETH) [34]. It implies conditional optimality of our algorithm for the decision version of k -Edits CPM for a general $k \leq n$ up to a subpolynomial factor.

The PILLAR model. For k -Mismatch CPM, we work in the PILLAR model that was introduced in [15] 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 substrings of strings in a collection \mathcal{X} , represented via handles:

- **Extract**(S, ℓ, r): Retrieve string $S[\ell..r]$.
- **LCP**(S, T), **LCP_R**(S, T): Compute the length of the longest common prefix/suffix of S, T .
- **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 sequence.
- **Access**(S, i): Retrieve the letter $S[i]$.
- **Length**(S): Compute the length $|S|$ of the string S .

In fact, in the standard setting, where the strings are given explicitly and are over an integer alphabet, all primitive PILLAR operations can be performed in $\mathcal{O}(1)$ time after a linear-time preprocessing. The runtime of algorithms in this model can be expressed in terms of the number of primitive PILLAR operations. (Any extra time required by our algorithms is explicitly specified.)

In the full version we obtain fast algorithms for k -Mismatch CPM in the internal, dynamic and fully compressed setting, by using Theorems 1 and 2 with known implementations of the primitive PILLAR operations in these settings.

2 k -Mismatch CPM

For a string S and an integer $x \leq |S|$, we denote $\text{rot}_x(S) = S[x..|S|] \cdot S[0..x]$. For $i \in [0..|S| - m]$, we denote $S^{(i)} = S[i..i + m]$. Also, we denote the set of standard (non-circular) k -mismatch occurrences of P in T by $\text{Occ}_k(P, T) = \{i \in [0..n - m] : T^{(i)} =_k P\}$.

Let $P = P_1 P_2$, where $|P_1| = \lfloor m/2 \rfloor$. Each circular k -mismatch occurrence of P in T implies a standard k -mismatch occurrence of at least one of P_1 and P_2 . Henceforth, we assume, without loss of generality, that it implies a k -mismatch occurrence of P_1 , as the remaining case is symmetric. Our goal is to consider all k -mismatch occurrences of P_1 in T as anchors for computing circular k -mismatch occurrences of P in T . We call P_1 *the sample*.

By the so-called standard trick, we will consider $\mathcal{O}(n/m)$ substrings of T of length $\mathcal{O}(m)$ and process each of them separately. We denote one such substring by T' . All positions (occurrences) of the sample can be efficiently computed in such a small *window* T' :

► **Theorem 6** ([15, Main Theorems 5 and 8]). *Given a text T' and a pattern P_1 , satisfying $|T'| = \mathcal{O}(|P_1|)$, we can compute a representation of the set $\text{Occ}_k(P_1, T')$ as $\mathcal{O}(k^2)$ pairwise disjoint arithmetic progressions in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ PILLAR operations.¹*

We want to find in T extensions of occurrences of P_1 in T' , which approximately match some rotation of P . Consider only rotations of P of the form YP_1X , where $P = P_1XY$. Define the set of circular k -mismatch occurrences i of P in T that imply a k -mismatch standard occurrence of P_1 in a substring $T' = T[a..b]$ as follows (such occurrences are *anchored* in (P_1, T')):

$$\text{Anchored}_k(P, T, P_1, T') = \{i : T^{(i)} =_k YP_1X, P = P_1XY, i + |Y| - a \in \text{Occ}_k(P_1, T')\}.$$

Our algorithms compute a superset of $\text{Anchored}_k(P, T, P_1, T')$ that may also contain other circular k -mismatch occurrences of P in T (some of the ones that contain a k -mismatch occurrence of P_2). Let $A = \text{Occ}_k(P_1, T')$, and recall that this refers to standard k -mismatch occurrences.

2.1 Few k -mismatch Occurrences of the Sample: $|A| = \mathcal{O}(k)$

We assume that $|A| = \mathcal{O}(k)$. Let us denote by $\text{PAIRMATCH}_k(T, P, i, j)$ the set of all circular k -mismatch occurrences of P in T such that position i in T is aligned with position j in P :

$$\text{PAIRMATCH}_k(T, P, i, j) = \{p \in [i - m + 1..i] : T^{(p)} =_k \text{rot}_x(P), i - p \equiv j - x \pmod{m}\}.$$

In particular $\text{PAIRMATCH}_k(T, P, i, 0)$ is the set of circular k -mismatch occurrences of P such that the first position of P is aligned with position i in T .

The following lemma was shown without explicitly mentioning the PILLAR model.

► **Lemma 7** ([14, see Lemma 10]). *$\text{PAIRMATCH}_k(T, P, i, j)$ can be computed in $\mathcal{O}(k)$ time in the PILLAR model for any given k, i, j , with the output represented as $\mathcal{O}(k)$ intervals.*

► **Lemma 8.** *Given $A = \text{Occ}_k(P_1, T')$ of size $\mathcal{O}(k)$, a superset of $\text{Anchored}_k(P, T, P_1, T')$, represented as $\mathcal{O}(k^2)$ intervals, can be computed in $\mathcal{O}(k^2)$ time in the PILLAR model. The computed superset contains only circular k -mismatch occurrences of P in T .*

Proof. Suppose that $T' = T[a..b]$. We then have

$$\text{Anchored}_k(P, T, P_1, T') \subseteq \bigcup_{i \in A} \text{PAIRMATCH}_k(T, P, i + a, 0) \subseteq \text{CircOcc}_k(P, T).$$

By the hypothesis $|A| = \mathcal{O}(k)$, the result, represented as a union of $\mathcal{O}(k^2)$ intervals, can be computed in $\mathcal{O}(k^2)$ time in the PILLAR model by means of Lemma 7. ◀

2.2 Many k -mismatch Occurrences of the Sample: $|A| > 864k$

We assume that $|A| = |\text{Occ}_k(P_1, T')| > 864k$. By [15], these k -mismatch occurrences imply approximate periodicity in both P_1 and the portion of T' spanned by k -mismatch occurrences of P_1 . We show that, except for $\mathcal{O}(k^2)$ intervals of circular k -mismatch occurrences of P that we can compute using $\mathcal{O}(k)$ calls to PAIRMATCH_k as in Section 2.1, our problem reduces to matching length- m substrings of an approximately periodic substring V in T and of an

¹ When referring to statements of [15], we use their numbering in the full (arxiv) version of the paper.

approximately periodic substring U in P^2 with aligned approximate periodicity. Afterwards, testing a match of two length- m substrings for U and V reduces to the test only on positions breaking periodicity, called *misperiods*, since the equality on other positions is guaranteed due to common approximate periodicities. The number of misperiods in U and V is only $\mathcal{O}(k)$, so the complexity of our algorithms, in terms of PILLAR operations, depends only on k .

By S^p we denote the concatenation of p copies of a string S . A string Q is called primitive if $Q = B^p$ for a string B and a positive integer p implies that $p = 1$. We introduce the following problem.

PERIODICSUBSTRINGMATCH(U, V)

Input: Positive integers m, k , and q , an integer $r \in [0..q)$, and strings U, V , and Q such that $m \leq |U|, |V| \leq 2m$, $q = |Q|$, and U, V are at Hamming distance $\mathcal{O}(k)$ from prefixes of $Q^\infty, (\text{rot}_r(Q))^\infty$, respectively.

Output: The set of positions p in U for which there exists a position x in V such that $U^{(p)} =_k V^{(x)}$ and $p - x \equiv r \pmod{q}$.

Our goal is to reduce k -Mismatch CPM in this case to PERIODICSUBSTRINGMATCH. To this end, we use the idea of repetitive regions from [15].

► **Definition 9.** A k -repetitive region in a string S of length m is a substring R of S of length $|R| \geq 3m/8$ for which there exists a primitive string Q such that

$$|Q| \leq m/(128k) \text{ and } \delta_H(R, Q^\infty[0..|R|]) = \lceil 8k|R|/m \rceil.$$

The following lemma is a simplified version of a lemma from [15] with one repetitive region.

► **Lemma 10** ([15, see Lemma 3.11]). Given a pattern P of length m , a text T of length $n \leq \frac{3}{2}m$, and a positive integer threshold $k \leq m$, if the pattern P contains a k -repetitive region, then $|\text{Occ}_k(P, T)| \leq 864k$.

Intuitively, in our main lemma in this section (Lemma 12), we show that if P_1 has many k -mismatch occurrences in T' , then each circular k -mismatch occurrence of P in T that is an element of $\text{Anchored}_k(P, T, P_1, T')$ is: (a) either computed in an instance of PERIODICSUBSTRINGMATCH; or (b) implies a k -mismatch occurrence of one of at most two k -repetitive regions of $P_2P_1P_2$. The occurrences of the second type can be computed using $\mathcal{O}(k)$ calls to PAIRMATCH $_k$ by viewing k -repetitive regions as samples and applying Lemma 10 to bound the number of k -mismatch occurrences.

In the proof of Lemma 12 we use the following theorem from [15]; part 1 follows from [15, Theorem 3.1] (existence of Q) and [15, Lemmas 3.8, 3.11, 3.14, and 4.4] (computation of Q), whereas the remaining parts are due to [15, Main Theorem 5].

► **Theorem 11** ([15]). Assume that we are given a pattern P of length m , a text T of length $n \leq \frac{3}{2}m$, and a positive integer threshold $k \leq m$. If $|\text{Occ}_k(P, T)| > 864k$ and T starts and ends with k -mismatch occurrences of P , that is, $0, |T| - m \in \text{Occ}_k(P, T)$, then:

1. there is a substring Q of P satisfying $|Q| \leq m/(128k)$ and $\delta_H(P, Q^\infty[0..|P|]) < 2k$, which can be computed in $\mathcal{O}(k)$ time in the PILLAR model;
2. each element of $\text{Occ}_k(P, T)$ is a multiple of $|Q|$;
3. $\delta_H(T, Q^\infty[0..n]) \leq 6k$;
4. $\text{Occ}_k(P, T)$ can be decomposed into $\mathcal{O}(k^2)$ arithmetic progressions with difference $|Q|$.

► **Lemma 12** (Reduction to PERIODICSUBSTRINGMATCH). *If we have $|T'| \leq \lfloor \frac{3}{4}m \rfloor$ and $|\text{Occ}_k(P_1, T')| > 864k$, then there is a superset of $\text{Anchored}_k(P, T, P_1, T')$ containing circular k -mismatch occurrences of P in T that is a union of $\mathcal{O}(k^2)$ intervals and of the output of PERIODICSUBSTRINGMATCH with U and V being substrings of T and P^2 , respectively. The intervals and the input to the PERIODICSUBSTRINGMATCH call can be computed in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ PILLAR operations.*

Proof. We apply Theorem 11 to P_1 and the part T'' of T' spanned by the k -mismatch occurrences in $\text{Occ}_k(P_1, T')$, obtaining a short primitive string Q . Consider the occurrence of P_1 at position $|P_2|$ of $P_2P_1P_2$. Let us extend this substring to the right, trying to accumulate enough mismatches with a prefix of Q^∞ in order to reach the threshold specified in Definition 9, which is $\Theta(k)$. In order to compute the next mismatch, it suffices to perform an LCP query between the remaining suffix of $P_2P_1P_2$ and some rotation of Q^∞ . An analogous process was described in detail in [15, Lemma 4.4], and the fact that the PILLAR model supports the described LCP queries is shown in [15, Corollary 2.9]. If we manage to accumulate enough mismatches, we call the resulting k -repetitive region R_R . We perform the same process by extending this occurrence of P_1 to the left, possibly obtaining a k -repetitive region R_L . Then, we let V be the shortest substring $(P_2P_1P_2)[v..v']$ of $P_2P_1P_2$ that spans both R_L (or P_2P_1 if R_L does not exist) and R_R (or P_1P_2 if R_R does not exist). Let us observe that V is at distance at most $2 \cdot \lceil 8km/m \rceil = 16k$ from a prefix of $(\text{rot}_{r(P)}(Q))^\infty$, where $r(P) = v - |P_2| \pmod{|Q|}$, by the definition of k -repetitive regions and the fact that $|R_R|, |R_L| \leq m$. Moreover, obviously, $|V| \leq 2m$.

The rotations of P that contain P_1 are in one-to-one correspondence with the length- m substrings of $P_2P_1P_2$. Each such substring either contains (at least) one of R_L and R_R or is contained in V . We now show that we can efficiently compute circular k -mismatch occurrences of P that imply k -mismatch occurrences of either R_L or R_R (if they exist) using the tools that developed in Section 2.1. We focus on R_R as R_L can be handled symmetrically. Due to Lemma 10, R_R has $\mathcal{O}(k)$ k -mismatch occurrences in T' , and they can be found in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ PILLAR operations using Theorem 6. For each such occurrence at some position i of T , we perform a call PAIRMATCH_k as in Lemma 8.

We are left with computing elements of $\text{Anchored}_k(P, T, P_1, T')$ corresponding to k -mismatch occurrences of length- m substrings of V in T in the case where $|V| \geq m$. Let us take the considered occurrence $T[a..b]$ of T'' in T , which is at distance at most $6k$ from a prefix of Q^∞ , and extend it to the right until either of the following three conditions is satisfied: (a) we reach the end of T ; (b) we have appended $\lceil m/2 \rceil$ letters; or (c) the resulting substring has $18k$ additional mismatches with the same-length prefix of Q^∞ . We extend the obtained substring to the left until either of the following three conditions is satisfied: (a) we reach the beginning of T ; (b) we have prepended $\lceil m/2 \rceil$ letters; or (c) the prepending substring has $18k$ mismatches with the same-length suffix of $Q^{|T|}$. We set the obtained substring $T[u..u']$ of T to be U . We observe that $|U| \leq 2m$ (since $\lfloor 3m/4 \rfloor \geq |T'| > 864k \Rightarrow m > 1152$ and $|U| \leq |T'| + 2 \lceil m/2 \rceil \leq 7m/4 + 2$) and U is at distance at most $6k + 2 \cdot 18k = 42k$ from a prefix of $(\text{rot}_{r(T)}(Q))^\infty$, where $r(T) = u - a \pmod{|Q|}$. If $|U| < m$, we do not construct the instance of PERIODICSUBSTRINGMATCH.

In the call to PERIODICSUBSTRINGMATCH, we set $Q := \text{rot}_{r(T)}(Q)$ and $r := (r(P) - r(T)) \pmod{|Q|}$. Now, since Q is primitive, it does not match any of its non-trivial rotations. Further, as we have at least $128k - 42k - 1$ (resp. $128k - 16k - 1$) exact occurrences of Q in any length- m fragment of U (resp. V), the periodicities must be synchronized in any circular k -mismatch occurrence. Hence, for any $p \in \text{Anchored}_k(P, T, P_1, T')$ that corresponds to $U^{(p)} =_k V^{(x)}$ we must have $p + r(T) \equiv x + r(P) \pmod{|Q|}$, and hence $p - x \equiv r(P) - r(T) \equiv r \pmod{|Q|}$.

It suffices to show that there is no position $i \in \text{Anchored}_k(P, T, P_1, T')$ such that $T^{(i)}$ is at distance at most k from a substring of V and $[i..i+m] \not\subseteq [u..u']$. Suppose that this is the case towards a contradiction, and assume without loss of generality that $i < u$; the other case is symmetric. We notice that this can only be the case if we stopped extending to the left because we accumulated enough mismatches. Further, let the implied k -mismatch occurrence of P_1 start at some position t of T , let x be an integer such that $\min_y \delta_H(V^{(y)}, T^{(i)}) = \delta_H(V^{(x)}, T^{(i)})$, and let F be the length- $(t-i)$ suffix of $Q^{|T|}$, i.e., $Q^{|T|}[[T] \cdot |Q| - (t-i) .. |T| \cdot |Q|]$. Then, via the triangle inequality, we have

$$\begin{aligned} \delta_H(V^{(x)}, T^{(i)}) &\geq \delta_H(V^{(x)}[0..t-i], T[i..t]) \\ &\geq \delta_H(T[i..t], F) - \delta_H(F, V^{(x)}[0..t-i]) \geq 18k - 16k > k, \end{aligned}$$

thus obtaining a contradiction. This completes the proof of this lemma. \blacktriangleleft

2.3 The Reporting Version of k -Mismatch CPM

In this section, we give a solution to PERIODICSUBSTRINGMATCH. Let us recall the notion of misperiods that was introduced in [14].

► **Definition 13.** A position a in S is a misperiod with respect to a substring Q of S if $S[a] \neq Q^\infty[a]$. We denote the set of misperiods by $\text{Misp}(S, Q)$.

In $\mathcal{O}(k)$ time in the PILLAR model we can compute the sets $I = \text{Misp}(U, Q)$ and $J = \text{Misp}(V, \text{rot}_r(Q))$. This is due to [15, Corollary 2.9], which allows us to answer queries of the form $\text{LCP}(S[i..j], Q^\infty[a..b])$ in $\mathcal{O}(1)$ time in the PILLAR model. For an integer z , let us denote $\mathbf{W}_z = [z..z+m)$ (a window of size m). We define $\text{Mispers}(i, j) = |\mathbf{W}_i \cap I| + |\mathbf{W}_j \cap J|$.

The following problem is a simpler version of PERIODICSUBSTRINGMATCH that was considered in [14]. (Actually, [14] considered a slightly more restricted problem which required that no two misperiods in $U^{(p)}$ and $V^{(x)}$ are aligned and computed a superset of its solution that corresponds exactly to the statement below.)

PERIODICPERIODICMATCH(U, V)

Input: Same as in PERIODICSUBSTRINGMATCH

Output: The set of positions p in U for which there exists a position x in V such that $U^{(p)} =_k V^{(x)}$, $p - x \equiv r \pmod{q}$, and $\text{Mispers}(p, x) \leq k$.

For an integer set A and an integer r , let $A \oplus r = \{a + r : a \in A\}$. An interval chain for an interval I and non-negative integers a and q is a set of the form $I \cup (I \oplus q) \cup (I \oplus 2q) \cup \dots \cup (I \oplus aq)$. Here q is called the *difference* of the interval chain.

► **Lemma 14** ([14, Lemma 15]). Given sets $I = \text{Misp}(U, Q)$ and $J = \text{Misp}(V, \text{rot}_r(Q))$, we can compute in $\mathcal{O}(k^2)$ time a solution to PERIODICPERIODICMATCH represented as $\mathcal{O}(k^2)$ interval chains, each with difference q .

Next we observe that if $U^{(p)} =_k V^{(x)}$, then either some two misperiods in $U^{(p)}$ and $V^{(x)}$ are aligned, or the total number of misperiods in these substrings is at most k .

► **Observation 15.** We have $\text{PERIODICSUBSTRINGMATCH}(U, V) =$

$$\text{PERIODICPERIODICMATCH}(U, V) \cup \bigcup_{i \in I, j \in J} \text{PAIRMATCH}_k(U, V, i, j).$$

By the observation, there are $\mathcal{O}(k^2)$ instances of PAIRMATCH_k , each taking $\mathcal{O}(k)$ time in the PILLAR model, and $\text{PERIODICPERIODICMATCH}$ can be solved in $\mathcal{O}(k^2)$ time. This results in total time complexity $\mathcal{O}(k^3)$ for $\text{PERIODICSUBSTRINGMATCH}$. Together with the previous reductions in Lemmas 8 and 12, this leads to Theorem 1. We only need to transform the output from a union of intervals and interval chains to a list of circular k -mismatch occurrences without duplicates. A full proof of Theorem 1 can be found in the full version.

2.4 A Faster Algorithm for the Decision Version of k -Mismatch CPM

Two aligned misperiods can correspond to zero or one mismatch, while two misaligned misperiods always yield two mismatches. Let us recall that $I = \text{Misp}(U, Q)$ and $J = \text{Misp}(V, \text{rot}_r(Q))$. We define the following *mismatch correcting* function:

$$\nabla(i, j) = \begin{cases} 0 & \text{if } (i, j) \notin I \times J, \text{ otherwise:} \\ 1 & \text{if } U[i] \neq V[j], \\ 2 & \text{if } U[i] = V[j]. \end{cases}$$

This function corrects *surplus mismatches*. Let $\text{Surplus}(i, j) = \sum_{t=0}^{m-1} \nabla(i+t, j+t)$.

DECISION $\text{PERIODICSUBSTRINGMATCH}(U, V)$

Input: Same as before, with the sets I, J stored explicitly.

Output: Any position $p \in [0..|U| - m]$ such that $\text{Mispers}(p, x) - \text{Surplus}(p, x) \leq k$ for some $x \in [0..|V| - m]$ such that $p - x \equiv r \pmod{q}$.

We consider an $(|U| - m + 1) \times (|V| - m + 1)$ grid \mathcal{G} . The δ -th diagonal in \mathbb{Z}^2 consists of points (i, j) that satisfy $i - j = \delta$. It is called an *essential* diagonal if $\delta = i - j$ for some $i \in I, j \in J$ and $\delta \equiv r \pmod{q}$. A point (i, j) on an essential diagonal is called *essential* if $i \in I$ and $j \in J$. Let us observe that only essential points influence the function Surplus , and the number of these points is $\mathcal{O}(k^2)$. This implies the following lemma using a simple 1D sweeping algorithm.

► **Lemma 16** (Compact representation of Mispers and Surplus). *In $\mathcal{O}(k^2 \log \log k)$ time we can:*

- (a) *Partition the grid \mathcal{G} by $\mathcal{O}(k)$ vertical and $\mathcal{O}(k)$ horizontal lines into $\mathcal{O}(k^2)$ disjoint rectangles such that for each point (i, j) in a single rectangle the value $\text{Mispers}(i, j)$ is the same. Each rectangle stores the value $\text{Mispers}(i, j)$ for an arbitrary point (i, j) that it contains.*
- (b) *Partition all essential diagonals in \mathcal{G} into $\mathcal{O}(k^2)$ pairwise disjoint diagonal segments, such that for each point (i, j) in a single segment the value $\text{Surplus}(i, j)$ is the same. Each segment stores the value $\text{Surplus}(i, j)$ for an arbitrary point (i, j) that it contains.*

Proof. Partitioning (a): We partition the first axis (second axis) into axis segments such that for all i in the same segment, the set $\mathbf{W}_i \cap I$ ($\mathbf{W}_j \cap J$, respectively) is the same. Then we create rectangles being Cartesian products of the segments.

Partitioning of an axis is performed with a 1D sweep; we describe it in the context of the first axis. For each $i \in I$, we create an event at position $i - m + 1$ where the misperiod i is inserted and an event at position $i + 1$ (if $i + 1 \leq |U| - m + 1$) where it is removed. We can now sort all events in $\mathcal{O}(k \log \log k)$ time using integer sorting [28] and process them in order,

storing the number of misperiods. For all i in a segment without events, the set $\mathbf{W}_i \cap I$ is the same. We obtain $\mathcal{O}(k)$ segments on each axis which yields $\mathcal{O}(k^2)$ rectangles. Part (a) works in $\mathcal{O}(k^2)$ time.

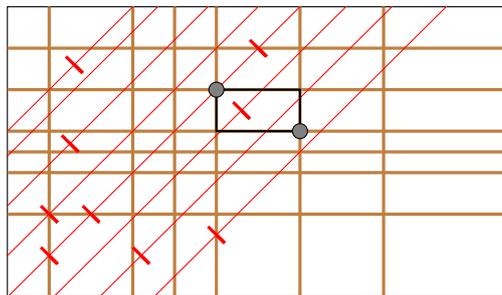
Partitioning (b): First we bucket sort essential points by (essential) diagonals using integer sorting. On each essential diagonal, we sort the essential points bottom-up and perform the same kind of 1D sweep as in (a), using ∇ to compute the weights of the events. The whole algorithm works in $\mathcal{O}(k^2 \log \log k)$ time. ◀

We assume that the grid \mathcal{G} is partitioned by selected horizontal and vertical lines into disjoint rectangles, called *cells*. These cells and some diagonal segments are weighted. Let us denote by $\text{cell}(i, j)$ and $\text{diag}(i, j)$ the weight of the cell and the diagonal segment containing (i, j) , respectively. In the following problem, we care only about points on diagonal segments.

DIAGONALSEGMENTS

Input: A grid partitioned by $\mathcal{O}(k)$ vertical and $\mathcal{O}(k)$ horizontal lines into $\mathcal{O}(k^2)$ weighted rectangles, called cells, and $\mathcal{O}(k^2)$ pairwise disjoint weighted diagonal line segments, all parallel to the line that passes through $(0, 0)$ and $(1, 1)$.

Output: Report a point $(x, y) \in \mathbb{Z}^2$ on some diagonal line segment with minimum $\text{val}(x, y) := \text{cell}(x, y) + \text{diag}(x, y)$.



■ **Figure 2** The weight of the distinguished cell is equal to $\text{Mispers}(i, j)$ for all points (i, j) in that cell. The diagonals are partitioned into segments. The weight of a single diagonal segment is equal to $-\text{Surplus}(i, j)$ for all points (i, j) that lie on that segment. In **DIAGONALSEGMENTS** we are to find any point (i, j) on some diagonal that minimizes the sum of the weight of its cell and of its diagonal segment, i.e., $\text{Mispers}(i, j) - \text{Surplus}(i, j)$. To this end, it suffices to consider endpoints of diagonal segments and crossings of diagonal segments with rectangles' boundaries.

An intuition of the solution to **DIAGONALSEGMENTS** is shown in Figure 2.

► **Lemma 17.** *The **DIAGONALSEGMENTS** problem can be solved in $\mathcal{O}(k^2 \log k / \log \log k)$ time.*

Proof. A sought point (x, y) that minimizes $\text{val}(x, y)$ either (1) is an endpoint of a diagonal segment or (2) lies on an intersection of a diagonal segment and an edge of a cell.

▷ **Claim 18.** Given $\mathcal{O}(p)$ horizontal lines and $\mathcal{O}(p)$ points, one can compute for each point the nearest line above it in $\mathcal{O}(p \log \log p)$ time.

Proof. We create a list of integers containing the vertical coordinates of all queried points and of all lines. Then we sort all of them in $\mathcal{O}(p \log \log p)$ time. The required answers can then be retrieved by a simple traversal of the sorted list. ◀

35:12 Approximate Circular Pattern Matching

In case (1) it suffices to identify, for all end points of all diagonal segments, the cells which they belong to. Let us assume that vertical and horizontal lines partition the grid into columns and rows, respectively. Then each cell can be uniquely identified by its column and row. With Claim 18 we can compute in $\mathcal{O}(k^2 \log \log k)$ time, for all queried points, the rows they belong to; the computation of columns is symmetric.

In case (2), let us consider intersections with horizontal edges; the intersections with vertical edges can be handled symmetrically. Assume x is the horizontal coordinate. We perform an affine transformation of the plane $(x, y) \mapsto (y - x, y)$ after which diagonal line segments become vertical, but horizontal line segments remain horizontal. The sought points can be computed using the following claim in $\mathcal{O}(k^2 \log k / \log \log k)$ time.

▷ **Claim 19.** Given s vertical and horizontal weighted line segments such that no two line segments of the same direction intersect, an intersection point of a vertical and a horizontal line segment with minimum total weight can be computed in $\mathcal{O}(s \log s / \log \log s)$ time.

Proof. We perform a left-to-right line sweep. The events in the sweep are the beginnings and endings of horizontal line segments and vertical line segments. The events can be sorted by their x -coordinates in $\mathcal{O}(s \log \log s)$ time with integer sorting. The horizontal line segments are stored using a dynamic predecessor data structure [46], and their weights in the same order are stored in a dynamic RMQ data structure of size $\mathcal{O}(s)$ that supports insertions, deletions, and range minimum queries in amortized $\mathcal{O}(\log s / \log \log s)$ time [11]. This way, when considering a vertical line segment, we can compute the minimum-weight horizontal line segment that intersects it in $\mathcal{O}(\log s / \log \log s)$ time. ◀

This concludes the solution to DIAGONALSEGMENTS. ◀

► **Theorem 2.** *The decision version of k -Mismatch CPM for strings over an integer alphabet can be solved in $\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$ time.*

Proof. Assume that P_1 is the sample. We use the so-called standard trick that covers T by a collection of its substrings T' of length $\lfloor \frac{3}{4}m \rfloor$ starting at positions divisible by $\lfloor \frac{1}{4}m \rfloor$. The following computations are performed for each T' .

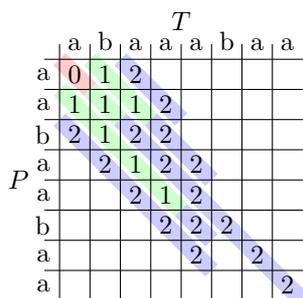
In this paragraph all complexities are stated in the PILLAR model. We use Theorem 6 to compute a representation of the set $A = \text{Occ}(P_1, T')$. If $|A| \leq 864k$, we can use Lemma 8 that outputs $\mathcal{O}(k^2)$ intervals of circular k -mismatch occurrences in $\mathcal{O}(k^2)$ time. Otherwise, we apply Lemma 12 which in $\mathcal{O}(k^2 \log \log k)$ time outputs $\mathcal{O}(k^2)$ intervals of circular k -mismatch occurrences and returns an instance of PERIODICSUBSTRINGMATCH. Next we use the geometric interpretation of PERIODICSUBSTRINGMATCH. The weight of a cell is the value $\text{Mispers}(i, j)$ common to all points (i, j) in this cell. Similarly, the weight of a diagonal segment equals $-\text{Surplus}(i, j)$, common to all points in this segment (it is the number of surplus misperiods which we have to subtract). These values are computed in $\mathcal{O}(k^2 \log \log k)$ time in Lemma 16. Now, the decision version of the PERIODICSUBSTRINGMATCH problem is reduced to the computation of the minimum value of $\text{Mispers}(i, j) - \text{Surplus}(i, j)$, which corresponds to the sum of weights of a cell and diagonal segment meeting at the same point (i, j) . The decision version of PERIODICSUBSTRINGMATCH can be reduced in $\mathcal{O}(k^2 \log \log k)$ time to one instance each of the DIAGONALSEGMENTS (if the sought point is on a diagonal segment) and PERIODICPERIODICMATCH (in the opposite case) problems. The thesis follows from Lemmas 14 and 17. ◀

► **Remark 20.** Using this geometric approach, the reporting version of k -Mismatch CPM can also be solved in $\mathcal{O}(n + (n/m) \cdot k^2 \log^{\mathcal{O}(1)} k + k \cdot \text{Output})$ time.

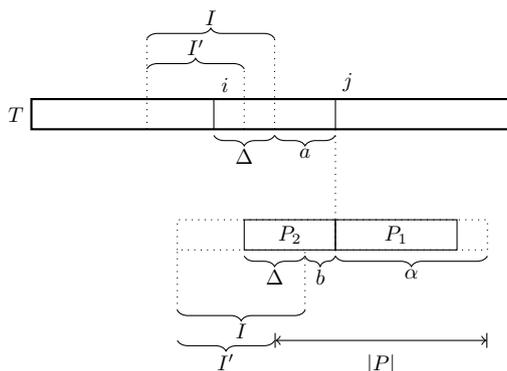
3 k -Edit CPM

In the proof of the following lemma, we rely on the Landau–Vishkin algorithm [38]; cf. Figure 3a for an illustration. The details can be found in the full version.

► **Lemma 21.** *Given a text T of length n , a pattern P of length m , and an integer $k > 0$, we can compute in $\mathcal{O}(k^2)$ time in the PILLAR model an $\mathcal{O}(k^2)$ -size representation of the edit distance between all pairs of prefixes of T and P that are at edit distance at most k , that is, the set $LV := \{(a, b, d) \in (0..n] \times (0..m] \times [0..k] : \delta_E(T[0..a], P[0..b]) = d \leq k\}$. Our representation of LV consists of $\mathcal{O}(k^2)$ sets of the form $\{(a + \Delta, b + \Delta, d) : \Delta \in [0..x]\}$.*



(a) Example for Lemma 21:
 $T = abaaabaa$, $P = aabaabaa$, $k = 2$.



(b) Notation used in the proof of Lemma 22.

■ **Figure 3** Illustrations for Lemmas 21 and 22.

► **Lemma 22.** *Given a text T of length n , a pattern P of length m , an integer $k > 0$, and a position j of T , after $\mathcal{O}(nk^2)$ preprocessing we can compute in $\mathcal{O}(k^2)$ time in the PILLAR model all positions $i \in \text{CircOcc}_k(P, T) \cap [0..j]$ such that $\delta_E(T[i..j], P_2) + \delta_E(T[j..r], P_1) \leq k$ for some position r of T and some strings P_1 and P_2 that satisfy $P = P_1P_2$, represented as $\mathcal{O}(k^2)$ intervals, possibly with duplicates.*

Proof. We start with the calculation of the compact representation of LV from Lemma 21 for the reversals of strings $T[0..j]$ and P , and k . Next, for each element $\{(a + \Delta, b + \Delta, d) : \Delta \in I\}$ of this compact representation, we will calculate an interval $I' \subseteq I$ such that positions from $\{j - a - \Delta : \Delta \in I'\}$ are in $\text{CircOcc}_k(P, T)$.

From the definition of LV we know that for any $\Delta \in I$, $i = j - a - \Delta$ and $P_2 = P[m - b - \Delta..m]$, we have $\delta_E(T[i..j], P_2) = d$. All we have to do is to verify if there exists P_1 fulfilling requirements of the lemma.

For each $k' \in [0..k]$, we compute $\text{LPref}_{k'}$ in $\mathcal{O}(nk)$ time [37]. Then, the maximal possible length of P_1 (within our edit distance budget) is $\alpha := \text{LPref}_{k-d}[j]$. Now, we need to define I' in such a way that it corresponds to pairs (P_1, P_2) with total length $|P|$, that is, we set $I' := \{\Delta \in I : b + \Delta + \alpha \geq m\}$. The notation used in this proof is illustrated in Figure 3b. The most time consuming part of this procedure is the calculation of the compact representation of LV , which takes $\mathcal{O}(k^2)$ time in the PILLAR model by Lemma 21. ◀

By applying the above lemma for P , T , and all positions j of T , and merging the obtained $\mathcal{O}(nk^2)$ intervals using bucket sort, we obtain Theorem 3.

Let us now move to the decision version of k -Edit CPM. For any integer $k \geq 0$, we define array $\text{LSuf}_k[0..n]$ such that $\text{LSuf}_k[i] = t$ if and only if $P[m-t..m]$ is the longest suffix of P at edit distance at most k from a suffix of $T[0..i]$. We make the following easy observation.

► **Observation 23.** *The pattern P has a circular k -edit occurrence in the text T if and only if $\text{LPref}_{k'}[j] + \text{LSuf}_{k-k'}[j] \geq m$ holds for some $j \in [0..n]$ and $k' \in [0..k]$.*

Due to Observation 23, Theorem 4, which is proved in Section 4, yields Theorem 5.

4 An Algorithm for All- k -LPAM: Proof of Theorem 4

We will show how to compute, given a position p in the text, values $\text{LPref}_{k'}[i]$ for all $k' \in [0..k]$ and $i \in [p.. \min\{p+k, n+1\}]$ in $\mathcal{O}(k^2 \log^3 k)$ time in the PILLAR model. This will yield the desired solution to All- k -LPAM by taking values of p which are multiples of k .

The *deletion distance* $\delta_D(U, V)$ of two strings U and V is the minimum number of letter insertions and deletions required to transform U to V ; in comparison to edit distance, substitutions are not allowed. For a string S , by $S_\$$ we denote the string $S[0]\$S[1]\$\cdots S[|S|-1]\$$. By the following known fact, we can easily transform the pattern and the text, doubling k , and henceforth consider the deletion distance instead of the edit distance.

► **Fact 24.** *For any two strings U and V over an alphabet Σ that does not contain $\$$, we have $2 \cdot \delta_E(U, V) = \delta_D(U_\$, V_\$)$.*

Note that an LCP query on suffixes of $U_\$$ and $V_\$$ trivially reduces in $\mathcal{O}(1)$ time to an LCP query on suffixes of U and V .

► **Definition 25.** *For two strings U and V and an integer range I , we define the alignment graph $G(U, V, I)$ as the weighted undirected graph over the set of vertices \mathbb{Z}^2 with the following edges for each $(a, b) \in \mathbb{Z}^2$:*

- $(a, b) \leftrightarrow (a+1, b)$ with weight 1, $(a, b) \leftrightarrow (a, b+1)$ with weight 1,
- $(a, b) \leftrightarrow (a+1, b+1)$ with weight 0 unless $a \in [0..|U|)$, $b \in [0..|V|)$, $U[a] \neq V[b]$, and $b-a \in I$.

For an alignment graph $G(U, V, I)$, there are $|I|$ diagonals on which diagonal edges may be missing. Intuitively, everything outside these diagonals is considered as a match.

► **Observation 26** ([16, see Lemma 8.5]). *For all fragments $U[a..a']$ and $V[b..b']$ of U and V , respectively, $\delta_D(U[a..a'], V[b..b'])$ is the length of the shortest $(a, b) \rightsquigarrow (a', b')$ path in $G(U, V, \mathbb{Z})$.*

Let $G := G(P, T, [p-k..p+2k])$. For each $t \in [0..m]$, we define a $(3k+2) \times (3k+2)$ distance matrix D_t such that $D_t[i, j]$ is the length of the shortest path between $(0, i+p-k-1)$ and $(t, t+j+p-k-1)$ in the alignment graph G . Let us recall that a matrix M is a *Monge matrix* if for every pair of rows $i < j$ and every pair of columns $\ell < r$, $M[i, \ell] + M[j, r] \leq M[i, r] + M[j, \ell]$. The planarity of G and the fact that all vertices of the considered shortest paths lie in $[0..t] \times \mathbb{Z}$, imply the following.

► **Observation 27.** *For every $t \in [0..m]$, the matrix D_t is a Monge matrix.*

Let us recall that a permutation matrix is a square matrix over $\{0, 1\}$ that contains exactly one 1 in each row and in each column. A permutation matrix P of size $s \times s$ corresponds to a permutation π of $[0..s)$ such that $P[i, j] = 1$ if and only if $\pi(i) = j$. For two permutations π_1, π_2 and their corresponding permutation matrices P_1, P_2 , by $\Delta(\pi_1, \pi_2) = \Delta(P_1, P_2)$ we

denote a shortest sequence of transpositions f_1, \dots, f_q of neighboring elements such that $f_q \circ \dots \circ f_1 \circ \pi_1 = \pi_2$. For an $s \times s$ matrix A , we denote by A^Σ an $(s+1) \times (s+1)$ matrix such that

$$A^\Sigma[i, j] = \sum_{i' \geq i} \sum_{j' < j} A[i', j'], \quad \text{for } i, j \in [0..s].$$

The lemma below follows readily from the tools developed in [16, Sections 8, 9] which, in turn, are based on the ideas of Tiskin [44, 45].

► **Lemma 28.**

- (a) Let $t \in [0..m]$, $i \in [p..p+k]$, $j \in [i-k..i+k] \cap [i-t..n-t]$, and $k' \in [0..k]$. Then, $\delta_D(T[i..j+t], P[0..t]) \leq k'$ if and only if $D_t[i-p+k+1, j-p+k+1] \leq k'$.
- (b) For each $t \in [0..m]$, there is a $(3k+1) \times (3k+1)$ permutation matrix P_t such that $D_t[i, j] = 2P_t^\Sigma[i, j] + i - j$ holds for all $i, j \in [0..3k+1]$. P_0 is an identity permutation matrix. A sequence $\Delta(P_0, P_1), \dots, \Delta(P_{m-1}, P_m)$ contains at most $3k(3k+1)/2$ transpositions of neighboring elements in total and all its non-empty elements can be computed in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ LCP queries on pairs of suffixes of P and T .

In the following lemma, whose proof is omitted here, we extend a known result on answering submatrix maximum queries on Monge matrices [35] (see also [25]) to a dynamic (the matrix changes by sub-row increments) and partially persistent (we need to be able to query all previously created matrices) setting. Sub-column queries are sufficient for our purposes; such queries were considered as a simpler case also in [35, 25]. We further consider minimum queries instead of maximum queries, which is a fairly straightforward change.

► **Lemma 29.** Let $M_0 = M$ be an $s \times s$ Monge matrix such that each entry of M_0 can be retrieved in $\mathcal{O}(1)$ time. We consider a sequence of operations:

- A sub-row increment takes the most recent matrix M_i and creates a matrix M_{i+1} resulting after this operation.
- A sub-column minimum query extracts the minimum in a given sub-column of a specified previously created matrix M_i .

The data structure for M_0 can be initialized in $\mathcal{O}(s \log^2 s)$ time. If each created matrix is a Monge matrix, then each incrementation can be performed in $\mathcal{O}(\log^3 s)$ time and each query can be answered in $\mathcal{O}(\log^2 s)$ time.

Proof of Theorem 4. Let us recall that we make all computations for $\mathcal{O}(n/k)$ values of p . We will store D_t for $t \in [0..m]$ using the data structure of Lemma 29. The initialization of D_0 takes $\mathcal{O}(k \log^2 k)$ time; we have $D_0[a, b] = |a - b|$. Each transposition in the maintained matrix P_t corresponds to a sub-column increment in D_t . Note that, for each $t \in [0..m]$, D_t is a Monge matrix due to Observation 27. Any intermediate matrix D is also Monge as it satisfies $D[i, j] = 2P^\Sigma[i, j] + i - j$ for a (maintained) permutation matrix P . Thus, for each t such that $\Delta(P_t, P_{t+1}) \neq \emptyset$, we can update the maintained Monge matrix as necessary using Lemma 29. By Lemma 28(b), the number of updates will be $\mathcal{O}(k^2)$ and the updates can be computed in $\mathcal{O}(k^2 \log \log k)$ time after $\mathcal{O}(n)$ -time preprocessing for LCP queries. The updates are performed in $\mathcal{O}(k^2 \log^3 k)$ total time.

We are to compute, for all $k' \in [0..k]$ and $i \in [p..p+k]$, the length of the longest prefix of P that matches a prefix of $T[i..n]$ with deletion distance at most k' . By Lemma 28(a) it suffices to find the maximum $t \in [0..m]$ such that $\min\{D_t[i-p+k+1, j-p+k+1] : j \in [i-k..i+k] \cap [i-t..n-t]\} \leq k'$. To this end, we apply binary search with $\mathcal{O}(\log k)$ steps on the set of all t that satisfy $\Delta(P_t, P_{t+1}) \neq \emptyset$, using $\mathcal{O}(\log^2 k)$ -time queries of Lemma 29. Thus, all binary searches take $\mathcal{O}(k^2 \log^3 k)$ time in total. The total time is $\mathcal{O}(n + (n/k) \cdot k^2 \log^3 k) = \mathcal{O}(nk \log^3 k)$. ◀

5 Conditional Hardness of Approximate CPM

We consider the following problem where the number of allowed mismatches is unbounded.

MISMATCH-CPM
Input: A text T of length n and a pattern P of length m .
Output: An array $\text{CPM}[0..n-m]$ with $\text{CPM}[i] = \min\{k \geq 0 : P \approx_k^{\delta_H} T[i..i+m]\}$.

In jumbled indexing, we are to answer pattern matching queries in the jumbled (abelian) sense. More precisely, given a Parikh vector of a pattern that specifies the quantity of each letter, we are to check if there is a substring of the text with this Parikh vector. In the case of a binary text, the problem of constructing a jumbled index is known to be equivalent (up to a log n -factor in the case where a concrete substring needs to be identified; see [20]) to constructing the following data structure.

Given a text X of length n over alphabet $\{0, 1\}$, for each $t \in [1..n]$ compute the values:

$$\min_t := \min \left\{ \sum_{j=i}^{i+t-1} X[j] : i \in [0..n-t] \right\}, \max_t := \max \left\{ \sum_{j=i}^{i+t-1} X[j] : i \in [0..n-t] \right\}.$$

For a few years since its introduction [19] the problem of constructing a binary jumbled index in $\mathcal{O}(n^{2-\epsilon})$ time for $\epsilon > 0$ was open. Chan and Lewenstein [13] settled this question affirmatively by proposing an $\mathcal{O}(n^{1.859})$ -time randomized construction; very recently it was improved to $\mathcal{O}(n^{1.5} \log^{\mathcal{O}(1)} n)$ time [18]. We make the following reduction.

► **Theorem 30.** *If Mismatch-CPM on binary strings can be solved in $S(n)$ time, then the BJI can be constructed in $\mathcal{O}(n + S(3n))$ time.*

Proof. We show how to compute \max_t for all $t \in [1..n]$. For computing \min_t , we can negate all the letters of X . An illustration of our reduction is provided in Figure 4.

$X = 0100101001001$	$n = 13$	$j = 5$	$t = 6$	$i = n - t = 7$
$\min_1 = 0$	$\max_1 = 1$	$\text{CPM}[i] = t - \max_t + j - \max_t = \text{CPM}[7] = 5$		
$\min_2 = 0$	$\max_2 = 1$	$P = 01001010010010000000000000$		
$\min_3 = 1$	$\max_3 = 2$	$T = \overset{0}{1}111111\overset{7}{1}11111000000000000000000000000000$		
$\min_4 = 1$	$\max_4 = 2$	$\text{rot}_4(P) = 101001001000000000000000100$		
$\min_5 = 1$	$\max_5 = 2$			
$\min_6 = 2$	$\max_6 = 3$			
...				

■ **Figure 4** $\text{CPM}[7] = 5$ corresponds to a Hamming distance of 5 between $T[7..7+|P|)$ and some rotation of P , namely of $\text{rot}_4(P)$.

It suffices to consider an instance of Mismatch-CPM with $P = X0^n$, $T = 1^n0^{2n}$. A prefix of a rotation of P of length at most n is also a substring of a string 0^nX0^n . For each $i \in [0..n)$, to compute $\text{CPM}[i]$ we need to choose a substring of 0^nX0^n of length $t = n - i$ with the most number of 1s as the prefix of some rotation of P . Indeed, if U is this prefix and P' is this rotation, that is, $P' = UV$, the remaining 0s in U and 1s in V will correspond to

mismatches between P' and $T[i..i+|P|)$. The maximum number of 1s among the substrings of $0^n X 0^n$ is the same as the maximum for X since it is never worse to choose a length- t prefix (suffix) of X than a shorter prefix (suffix) and a part from 0^n .

Thus, we have $\text{CPM}[i] = t - \max_t + j - \max_t$, where j is equal to the number of 1s in X . Hence, $\max_t = (t + j - \text{CPM}[n - t])/2$. These values can be recovered from the output of Mismatch-CPM in linear time. ◀

The following theorem shows how to compute the edit distance of two strings with the use of an algorithm for the decision version of k -Edit CPM, and thus also that a strongly subquadratic such algorithm would refute SETH [34]. The proof is omitted in this version.

► **Theorem 31.** *If k -Edit CPM on quaternary strings can be solved in $\mathcal{O}(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$, then the edit distance of two binary strings each of length at most n can be computed in $\mathcal{O}(n^{2-\varepsilon} \log n)$ time.*

References

- 1 Karl R. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987. doi:10.1137/0216067.
- 2 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.
- 3 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.
- 4 Lorraine A. K. Ayad, Solon P. Pissis, and Ahmad Retha. libFLASM: A software library for fixed-length approximate string matching. *BMC Bioinformatics*, 17:454:1–454:12, 2016. doi:10.1186/s12859-016-1320-2.
- 5 Md. Aashikur Rahman Azim, Costas S. Iliopoulos, M. Sohel Rahman, and M. Samiruzzaman. A filter-based approach for approximate circular pattern matching. In *Bioinformatics Research and Applications - 11th International Symposium, ISBRA 2015*, volume 9096 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2015. doi:10.1007/978-3-319-19048-8_3.
- 6 Carl Barton, Costas S. Iliopoulos, Ritu Kundu, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Accurate and efficient methods to improve multiple circular sequence alignment. In *Experimental Algorithms, SEA 2015*, volume 9125 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2015. doi:10.1007/978-3-319-20086-6_19.
- 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 Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Average-case optimal approximate circular string matching. In *Language and Automata Theory and Applications - 9th International Conference, LATA 2015*, volume 8977 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2015. doi:10.1007/978-3-319-15579-1_6.
- 9 Carl Barton, Costas S. Iliopoulos, Solon P. Pissis, and William F. Smyth. Fast and simple computations using prefix tables under Hamming and edit distance. In *Combinatorial Algorithms - 25th International Workshop, IWOCA 2014*, volume 8986 of *Lecture Notes in Computer Science*, pages 49–61. Springer, 2014. doi:10.1007/978-3-319-19315-1_5.
- 10 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.
- 11 Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. Path minima queries in dynamic weighted trees. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011*, volume 6844 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2011. doi:10.1007/978-3-642-22300-6_25.

- 12 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 643–656. ACM, 2020. doi:10.1145/3357713.3384266.
- 13 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.
- 14 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszłyński, 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.
- 15 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. doi:10.1109/FOCS46700.2020.00095.
- 16 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance. In *Accepted to 63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, 2022. arXiv:2204.03087.
- 17 Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Bit-parallel algorithms for exact circular string matching. *The Computer Journal*, 57(5):731–743, 2014. doi:10.1093/comjnl/bxt023.
- 18 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC 2022: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.
- 19 Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Searching for jumbled patterns in strings. In *Proceedings of the Prague Stringology Conference 2009*, pages 105–117, 2009. URL: <http://www.stringology.org/event/2009/p10.html>.
- 20 Ferdinando Cicalese, Travis Gagie, Emanuele Giaquinta, Eduardo Sany Laber, Zsuzsanna Lipták, Romeo Rizzi, and Alexandru I. Tomescu. Indexes for jumbled pattern matching in strings, trees and graphs. In *String Processing and Information Retrieval - 20th International Symposium, SPIRE 2013*, volume 8214 of *Lecture Notes in Computer Science*, pages 56–63. Springer, 2013. doi:10.1007/978-3-319-02432-5_10.
- 21 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k -mismatch problem revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.
- 22 Michael J. Fischer and Michael S. Paterson. String-matching and other products. Technical report, Massachusetts Institute of Technology, USA, 1974.
- 23 Kimmo Fredriksson and Szymon Grabowski. Average-optimal string matching. *Journal of Discrete Algorithms*, 7(4):579–594, 2009. doi:10.1016/j.jda.2008.09.001.
- 24 Kimmo Fredriksson and Gonzalo Navarro. Average-optimal single and multiple approximate string matching. *ACM Journal of Experimental Algorithmics*, 9, 2004. doi:10.1145/1005813.1041513.
- 25 Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Submatrix maximum queries in Monge and partial Monge matrices are equivalent to predecessor search. *ACM Transactions on Algorithms*, 16(2):16:1–16:24, 2020. doi:10.1145/3381416.
- 26 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. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.62.

- 27 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.
- 28 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.
- 29 Tommi Hirvola and Jorma Tarhio. Bit-parallel approximate matching of circular strings with k mismatches. *ACM Journal of Experimental Algorithmics*, 22, 2017. doi:10.1145/3129536.
- 30 ThienLuan Ho, Seungroh Oh, and Hyunjin Kim. New algorithms for fixed-length approximate string matching and approximate circular string matching under the Hamming distance. *The Journal of Supercomputing*, 74(5):1815–1834, 2018. doi:10.1007/s11227-017-2192-6.
- 31 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, and Sharma V. Thankachan. Space-efficient construction algorithm for the circular suffix tree. In *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013*, volume 7922 of *Lecture Notes in Computer Science*, pages 142–152. Springer, 2013. doi:10.1007/978-3-642-38905-4_15.
- 32 Wing-Kai Hon, Chen-Hua Lu, Rahul Shah, and Sharma V. Thankachan. Succinct indexes for circular patterns. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011*, volume 7074 of *Lecture Notes in Computer Science*, pages 673–682. Springer, 2011. doi:10.1007/978-3-642-25591-5_69.
- 33 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.
- 34 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 35 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 338–355. SIAM, 2012. doi:10.1137/1.9781611973099.31.
- 36 S. Rao Kosaraju. Efficient string matching. Manuscript, 1987.
- 37 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/S0097539794264810.
- 38 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.
- 39 M. Lothaire. *Applied Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005. doi:10.1017/CB09781107341005.
- 40 Vicente Palazón-González and Andrés Marzal. On the dynamic time warping of cyclic sequences for shape retrieval. *Image and Vision Computing*, 30(12):978–990, 2012. doi:10.1016/j.imavis.2012.08.012.
- 41 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.
- 42 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.
- 43 Robert Susik, Szymon Grabowski, and Sebastian Deorowicz. Fast and simple circular pattern matching. In *Man-Machine Interactions 3, Proceedings of the 3rd International Conference on Man-Machine Interactions, ICMMI 2013*, volume 242 of *Advances in Intelligent Systems and Computing*, pages 537–544. Springer, 2013. doi:10.1007/978-3-319-02309-0_59.
- 44 Alexander Tiskin. Semi-local string comparison: algorithmic techniques and applications, 2007. doi:10.48550/ARXIV.0707.3619.
- 45 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008. doi:10.1007/s11786-007-0033-3.
- 46 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.