Journal of Computer and System Sciences 115 (2021) 73–85

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Circular pattern matching with k mismatches

Panagiotis Charalampopoulos [a,b,1], Tomasz Kociumaka [c,b,2], Solon P. Pissis [d,e,*], Jakub Radoszewski [b,3,4], Wojciech Rytter [b], Juliusz Straszyński [b,3,4], Tomasz Waleń [b,4], Wiktor Zuba [b,4]

[a] Department of Informatics, King's College London, London, UK
[b] Institute of Informatics, University of Warsaw, Warsaw, Poland
[c] Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
[d] CWI, Amsterdam, the Netherlands
[e] Vrije Universiteit, Amsterdam, the Netherlands

A R T I C L E   I N F O

Article history:
Received 1 December 2019
Received in revised form 1 July 2020
Accepted 19 July 2020
Available online 29 July 2020

Keywords:
Circular pattern matching
k-mismatch problem
Approximate pattern matching

A B S T R A C T

We consider the circular pattern matching with k mismatches (k-CPM) problem in which one is to compute the minimal Hamming distance of every length-m substring of T and any cyclic rotation of P, if this distance is no more than k. It is a variation of the well-studied k-mismatch problem. A multitude of papers has been devoted to solving the k-CPM problem, but only average-case upper bounds are known. In this paper, we present the first non-trivial worst-case upper bounds for this problem. Specifically, we show an $\mathcal{O}(nk)$-time algorithm and an $\mathcal{O}(n+\frac{n}{m}k^4)$-time algorithm. The latter algorithm applies in an extended way a technique that was very recently developed for the k-mismatch problem Bringmann et al. (2019) [10].

A preliminary version of this work appeared at FCT 2019 [35]. In this version we improve the time complexity of the second algorithm from $\mathcal{O}(n+\frac{n}{m}k^5)$ to $\mathcal{O}(n+\frac{n}{m}k^4)$.

© 2020 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Pattern matching is a fundamental problem in computer science [1]. It consists in finding all substrings of a text T of length n that match a pattern P of length m. In many real-world applications, a measure of similarity is usually introduced allowing for *approximate* matches between the given pattern and substrings of the text. The most widely-used similarity measure is the Hamming distance between the pattern and all length-m substrings of the text.

* Corresponding author.
E-mail addresses: panagiotis.charalampopoulos@kcl.ac.uk (P. Charalampopoulos), kociumaka@mimuw.edu.pl (T. Kociumaka), solon.pissis@cwi.nl (S.P. Pissis), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), jks@mimuw.edu.pl (J. Straszyński), walen@mimuw.edu.pl (T. Waleń), w.zuba@mimuw.edu.pl (W. Zuba).

1 Supported by the ERC grant TOTAL agreement no. 677651, a Studentship from the Faculty of Natural and Mathematical Sciences at King's College London and an A.G. Leventis Foundation Educational Grant.
2 Supported by ISF grants no. 1278/16 and 1926/19, a BSF grant no. 2018364, and an ERC grant MPM (no. 683064) under the EU's Horizon 2020 Research and Innovation Programme.
3 Supported by the "Algorithms for text processing with errors and uncertainties" project carried out within the HOMING program of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund, project no. POIR.04.04.00-00-24BA/16.
4 Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

https://doi.org/10.1016/j.jcss.2020.07.003
0022-0000/© 2020 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Computing the Hamming distance between P and all length-m substrings of T has been investigated for the past 30 years. The first efficient solution requiring $\mathcal{O}(n\sqrt{m \log m})$ time was independently developed by Abrahamson [2] and Kosaraju [3] in 1987. The k-mismatch version of the problem asks for finding only the substrings of T that are close to P, specifically, at Hamming distance at most k. The first efficient solution to this problem running in $\mathcal{O}(nk)$ time was developed in 1986 by Landau and Vishkin [4]. It took almost 15 years for a breakthrough result by Amir et al. improving this to $\mathcal{O}(n\sqrt{k \log k})$ [5]. More recently, there has been a resurgence of interest in the k-mismatch problem. Clifford et al. gave an $\mathcal{O}((n/m)(k^2 \log k) + n\text{polylog}n)$-time algorithm [6], which was subsequently improved further by Gawrychowski and Uznański to $\mathcal{O}((n/m)(m + k\sqrt{m})\text{polylog}n)$ [7]. Very recently, Chan et al. [8] improved the polylog$n$ factors in the latter solution at the cost of (Monte-Carlo) randomization. Moreover, Gawrychowski and Uznański [7] showed that a significantly faster “combinatorial” algorithm for this problem is rather unlikely.

The k-mismatch problem has also been considered on compressed representations of the text [9–13], in the parallel model [14], in the streaming model [15,6,16], and in the setting of dynamic strings [13]. Furthermore, it has been considered in non-standard stringology models, such as the parameterized model [17] and the order-preserving model [18].

The matching relation (e.g., identity or Hamming distance at most k) in the standard pattern matching setting assumes that the leftmost and rightmost positions of the pattern are conceptually important. In many real-world applications, such as in bioinformatics [19–22] or in image processing [23–26], any cyclic shift (rotation) of P is a relevant pattern. In bioinformatics, the position where a sequence starts can be totally arbitrary due to, for instance, arbitrariness in the sequencing of a circular molecular structure or inconsistencies introduced into sequence databases due to different linearization standards [19]. In image processing, 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 [23]. Thus one is interested in computing the minimal distance of every length-m substring of T and any cyclic rotation of P, if this distance is no more than k. This is the circular pattern matching with k mismatches (k-CPM) problem.

A multitude of papers [27–32] have thus been devoted to solving the k-CPM problem but, to the best of our knowledge, only average-case upper bounds are known; i.e., in these works the assumption is that text T is uniformly random. The main result states that, after preprocessing pattern P, the average-case optimal search time of $\mathcal{O}(n\frac{k+\log m}{m})$ [33] can be achieved for certain values of the error ratio k/m (see [31,27] for more details on the preprocessing costs). Note that the exact (no mismatches allowed) version of the CPM problem can be solved as fast as exact pattern matching; namely, in $\mathcal{O}(n)$ time [34].

In this paper, we draw our motivation from **(i)** the importance of the k-CPM problem in real-world applications and **(ii)** the fact that no (non-trivial) worst-case upper bounds are known. Trivial here refers to running the fastest-known algorithm for the k-mismatch problem [7] separately for each of the m rotations of P. This yields an $\mathcal{O}(n(m + k\sqrt{m})\text{polylog}n)$-time algorithm for the k-CPM problem. This is clearly unsatisfactory: it is a simple exercise to design an $\mathcal{O}(nm)$-time or an $\mathcal{O}(nk^2)$-time algorithm. In an effort to tackle this unpleasant situation, we present two much more efficient algorithms: a simple $\mathcal{O}(nk)$-time algorithm and an $\mathcal{O}(n + \frac{n}{m}k^4)$-time algorithm. Our second algorithm applies in an extended way a technique that was developed very recently for k-mismatch pattern matching in grammar compressed strings by Bringmann et al. [10]. We also show that both of our algorithms can be implemented in $\mathcal{O}(m)$ space.

A preliminary version of this work was published as [35].

Our approach We first consider a simple version of the problem (called Anchor-Match) in which we are given a position in T (an *anchor*) which belongs to potential k-mismatch circular occurrences of P. A simple $\mathcal{O}(k)$-time algorithm is given (after linear-time preprocessing) to compute all relevant occurrences. By considering separately each position in T as an anchor we obtain an $\mathcal{O}(nk)$-time algorithm. The concept of an anchor is extended to the so-called *matching pairs*: when we know a pair of positions, one in P and the other in T, that are aligned. Then comes the idea of a *sample* $\mathcal{S}$, which is a fragment of P of length $\Theta(m/k)$ which supposedly exactly matches a corresponding fragment in T. We choose $\mathcal{O}(k)$ samples and work for each of them and for windows of T of size $2m$. As it is typical in many versions of pattern matching, our solution is split into periodic and non-periodic cases. If $\mathcal{S}$ is non-periodic the sample occurs only $\mathcal{O}(k)$ times in a window and each occurrence gives a matching pair (and consequently two possible anchors). Then we perform Anchor-Match for each such anchor. The hard part is the case when $\mathcal{S}$ is periodic. Here we compute all exact occurrences of $\mathcal{S}$ and obtain $\mathcal{O}(k)$ groups of occurrences, each one being an arithmetic progression. Now each group is processed using the approach “few matches or almost periodicity” of Bringmann et al. [10]. In the latter case periodicity is approximate allowing up to k mismatches. Finally, we are able to decrease the exponent of k by one in the complexity using a marking trick.

2. Preliminaries

Let $S = S[0]S[1]\cdots S[n-1]$ be a *string* of length $|S| = n$ over an integer alphabet Σ. The elements of Σ are called *letters*. For two positions i and j on S, we denote by $S[i \mathinner{.\,.} j] = S[i]\cdots S[j]$ the *fragment* of S that starts at position i and ends at position j (the fragment is empty, denoted ε, if $j < i$). A *prefix* of S is a fragment that starts at position 0, i.e., of the form $S[0 \mathinner{.\,.} j]$, and a *suffix* is a fragment that ends at position $n-1$, i.e., of the form $S[i \mathinner{.\,.} n-1]$. For an integer p, we define the pth *power* of S, denoted by S^p, as the string obtained from concatenating p copies of S. The string obtained by concatenating infinitely many copies of S is denoted by S^∞. If S and S' are two strings of the same length, then by $S =_k S'$

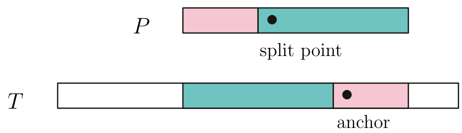

Fig. 1. The split point and the anchor for a k-occurrence of P in T.

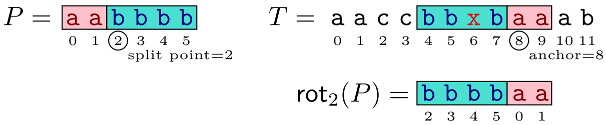

Fig. 2. A 1-occurrence of a pattern P = aabbbb in a text T = aaccbbxbaaab at position $p = 4$ with rotation $x = 2$; $M(4,2) = \{(4,2),(5,3),(6,4),(7,5),(8,0),(9,1)\}$.

we denote the fact that S and S' have at most k mismatches, that is, that the Hamming distance between S and S' does not exceed k.

We say that an integer $1 \le q \le |S|$, is a *period* of a string S if $S[i] = S[i+q]$ for $i = 0, \dots, |S| - q - 1$. We denote the smallest period of S by $\mathsf{per}(S)$ and say that a string S is *periodic* if $2\mathsf{per}(S) \le |S|$. Fine and Wilf's periodicity lemma [36] asserts that if a string of length n has periods p and q and $n \ge p + q - \gcd(p,q)$, then the string has a period $\gcd(p,q)$.

For a string S and integer $0 \le x < |S|$, by $\mathsf{rot}_x(S)$ we denote the string that is obtained from S by moving the prefix of S of length x to the end. More formally,

$$\mathsf{rot}_x(S) = VU, \text{ where } S = UV \text{ and } |U| = x.$$

We call the string $\mathsf{rot}_x(S)$ a *rotation* of S and often represent rotations of S using the underlying values x.

2.1. Anatomy of circular occurrences

In what follows, we denote by m the length of the pattern P and by n the length of the text T. We say that P has a *k-mismatch circular occurrence* (in short, a *k-occurrence*) in T at position p if $T[p \mathinner{.\,.} p+m-1] =_k \mathsf{rot}_x(P)$ for some rotation x. In this case, the position x in the pattern is called the *split point* and the position $p + (m - x) \bmod m$ in the text[5] is called the *anchor*. In other words, if $P = UV$ and its rotation VU occurs in T, then the first position of V in P is the split point of this occurrence, and the first position of U in T is the anchor of this occurrence (see Fig. 1).

The main problem in scope can now be stated as follows.

> k-CPM PROBLEM
>
> **Input:** A text T of length n, a pattern P of length m, and a positive integer k.
>
> **Output:** The positions of all k-occurrences of P in T.

For an integer z, let us denote $\mathbf{W}_z = [z \mathinner{.\,.} z+m-1]$ (a *window* of size m). Intuitively, this window corresponds to a length-m fragment of the text T. For a k-occurrence at position p of T with rotation x, we introduce a set of pairs of positions in the fragment of the text and the corresponding positions from the original (unrotated) pattern P:

$$M(p,x) = \{(i, (i - p + x) \bmod m) : i \in \mathbf{W}_p\}.$$

The pairs $(i,j) \in M(p,x)$ are called *matching pairs* of an occurrence p with rotation x. In particular, $(p + ((m-x) \bmod m), 0) \in M(p,x)$. An example is provided in Fig. 2.

3. An $\mathcal{O}(nk)$-time algorithm

We first introduce an auxiliary problem in which one wants to compute all k-occurrences of P in T with a given anchor $\mathbf{a}$. This problem describes the core computational task in our first solution.

> ANCHOR-MATCH PROBLEM
>
> **Input:** A text T of length n, a pattern P of length m, a positive integer k, and a position $\mathbf{a}$.
>
> **Output:** All k-occurrences of P in T with anchor $\mathbf{a}$, represented as a collection of $\mathcal{O}(k)$ intervals.

[5] The modulo operation is needed to handle the trivial rotation with $x = 0$.

$P^2 =$ a b a a b a b a a b a a b a b a a b a a b a b a a b a a b a b a

$T =$ b b a a b a a a a b a a a a b b a b a b b a b a b b a a b a a b

$V =$ 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

anchor=16

Fig. 3. An illustration of the setting in Lemma 2 with $P = (\texttt{abaababa})^2$, the text as in the figure, anchor $\mathbf{a} = 16$, and $k = 3$. The string V, used in the proof of the lemma, shows the positions of the at most k mismatches to the left and to the right of the anchor. The output consists of the three intervals $[1\,..\,3]$, $[7\,..\,8]$ and $[13\,..\,14]$ shown in orange. For example, 7 is a 3-occurrence since the fragment of V of length $|P|$ starting at this position (represented by a rectangle) contains at most 3 ones. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

For a binary string X, by $\|X\|$ we denote the arithmetic sum of characters in X. We define the following auxiliary problem.

Light-Fragments Problem

Input: Positive integers m, k and a string V of length n over alphabet $\{0, 1\}$ containing $\mathcal{O}(k)$ non-zero characters. The string V is specified by its positions with non-zero characters (sorted increasingly).

Output: The set $A = \{i : \|V[i\,..\,i+m-1]\| \le k\}$ represented as a collection of $\mathcal{O}(k)$ intervals.

Lemma 1. *The* Light-Fragments *problem can be solved in $\mathcal{O}(k)$ time.*

Proof. Let I be the set of positions of V with non-zero characters; $|I| = \mathcal{O}(k)$ by definition. We define a piecewise constant function $f : [0\,..\,|V| - m + 1] \to \mathbb{Z}$ such that $f(x) = |I \cap [0\,..\,x]|$. Let $g(x) = f(x+m-1) - f(x-1)$. Then $g(x) = |I \cap [x\,..\,x+m-1]|$. The function f has $\mathcal{O}(k)$ pieces, so g has $\mathcal{O}(k)$ pieces as well, and both can be computed in $\mathcal{O}(k)$ time since I is sorted. In the end, we report the pieces where g has value at most k. $\square$

Let us recall a standard algorithmic tool for preprocessing text T. We denote the length of the longest common prefix (resp. suffix) of two strings U and V by $\mathsf{lcp}(U, V)$ (resp. $\mathsf{lcs}(U, V)$). There is an $\mathcal{O}(n)$-sized data structure answering such queries over suffixes (resp. prefixes) of T in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time preprocessing. It consists of the suffix array of T and a data structure for answering range minimum queries; see [1]. Using the kangaroo method [4,14], these queries can handle mismatches; after an $\mathcal{O}(n)$-time preprocessing of T, longest common prefix (resp. suffix) queries with up to k mismatches can be answered in $\mathcal{O}(k)$ time.

Lemma 2. *After $\mathcal{O}(n)$-time preprocessing of T and P, the* Anchor-Match *problem can be solved in $\mathcal{O}(k)$ time for any given k and* $\mathbf{a}$.

Proof. In the preprocessing, we prepare a data structure for lcp queries in $P\#T$, where $\#$ is a special character that occurs neither in P nor in T.

Consider now a query for an anchor $\mathbf{a}$ over T. Let $L = T[a-m\,..\,a-1]$ and $R = T[a\,..\,a+m-1]$, with $\#$ at out-of-bounds positions of T. We define a binary string L' such that $L'[i] = 1$ if and only if $L[i] \ne P[i]$. We define R' analogously. Let L'' be the longest suffix of L' such that $\|L''\| \le k$ and R'' be the longest prefix of R' such that $\|R''\| \le k$.

Using the kangaroo method [4,14], the strings L'', R'' can be constructed in $\mathcal{O}(k)$ time. The Anchor-Match problem now reduces to the Light-Fragments problem for the string $V = L''R''$. $\square$

For an illustration of Lemma 2, inspect Fig. 3.

Proposition 3. *The k-CPM problem can be solved in $\mathcal{O}(nk)$ time and $\mathcal{O}(n)$ space.*

Proof. We invoke the algorithm of Lemma 2 for all $\mathbf{a} \in [0\,..\,n-1]$ and obtain $\mathcal{O}(nk)$ intervals of k-occurrences of P in T. Instead of storing all the intervals, we count how many intervals start and end at each position of T. We can then compute the union of the intervals by processing these counts from left to right. $\square$

4. Algorithmic tools

In this section, we introduce further algorithmic tools to get to our second solution.

4.1. Internal queries in a text

Let T be a string of length n called text. An Internal Pattern Matching (IPM) query, for two given fragments F and G of the text such that $|G| \le 2|F|$, computes the set of all occurrences of F in G. If there are more than two occurrences, they

$$V = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

$$U = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

$$0\ 1\ 2\ 3\ \boxed{4\ 5}\ 6\ 7\ \boxed{8\ 9}\ 10\ 11\ \boxed{12\ 13}\ 14\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25\ 26\ 27\ 28$$

Fig. 4. An instance of the ALIGNED-LIGHT-SUM problem with $m = 15$, $k = 2$, $q = 4$, $U = 010^{12}10^{14}$, and $V = 010^{14}$. The output is the interval chain $\mathsf{Chain}_4([4\,..\,5], 2)$ shown in orange.

form an arithmetic sequence with difference $\mathsf{per}(F)$. A data structure for IPM queries in T can be constructed in $\mathcal{O}(n)$ time and answers queries in $\mathcal{O}(1)$ time (see [37] and [38, Theorem 1.1.4]). It can be used to compute all occurrences of a given fragment F of length p in T, expressed as a union of $\mathcal{O}(n/p)$ pairwise disjoint arithmetic sequences with difference $\mathsf{per}(F)$, in $\mathcal{O}(n/p)$ time.

4.2. Simple geometry of arithmetic sequences of intervals

We next present algorithms that will be used in subsequent proofs for handling regular sets of intervals.

For an interval I and an integer r, let $I \oplus r = \{i + r : i \in I\}$. We define

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

This set is further called an *interval chain* (with difference q). Note that a chain can be represented in $\mathcal{O}(1)$ space using four integers: a, q, and the endpoints of I. An illustration of an interval chain representing the output for a problem defined in the next subsection can be found in Fig. 4.

For a given value of q, let us fit the integers from $[1\,..\,n]$ into the cells of a grid of width q so that the first row consists of numbers 1 through q, the second of numbers $q + 1$ to $2q$, etc. Let us call this grid $\mathcal{G}_q$. A chain Chain_q can be conveniently represented in the grid $\mathcal{G}_q$ using the following lemma from [39].

Lemma 4 *([39]). The set $\mathsf{Chain}_q(I, a)$ is a union of $\mathcal{O}(1)$ orthogonal rectangles in $\mathcal{G}_q$. The coordinates of the rectangles can be computed in $\mathcal{O}(1)$ time.*

The following lemma can be used to compute a union of interval chains.

Lemma 5. *Given c interval chains, all of which have difference q and are subsets of $[0\,..\,n]$, the union of these chains, expressed as a subset of $[0\,..\,n]$, can be computed in $\mathcal{O}(n + c)$ time.*

Proof. By Lemma 4, the problem reduces to computing the union of $\mathcal{O}(c)$ rectangles on a grid of total size n. Let t be a 2D array of the same shape as $\mathcal{G}_q$, initially set to zeroes. For a rectangle with opposite corners (x_1, y_1) and (x_2, y_2), with $x_1 \le x_2$ and $y_1 \le y_2$, we increment $t[x_1, y_1]$, decrement $t[x_2 + 1, y_1]$ and $t[x_1, y_2 + 1]$, and increment $t[x_2 + 1, y_2 + 1]$ (provided that the respective cells are within the array). This takes $\mathcal{O}(c)$ time. We then compute prefix sums of t, which are defined as

$$t'[x, y] = \sum_{i=1}^{x} \sum_{j=1}^{y} t[i, j].$$

Such values can be computed in time proportional to the size of the grid, i.e., in $\mathcal{O}(n)$ time. Finally, we note that (x, y) is contained in $t'[x, y]$ rectangles, concluding the proof. $\square$

Remark 6. The proof of Lemma 5 is essentially based on an idea that was used, for example, for reducing the decision version of range stabbing queries in 2D to weighted range counting queries in 2D (cf. [40]).

We will also use the following auxiliary lemma.

Lemma 7. *Let X and Z be intervals and q be a positive integer. The set*

$$Z' := \{z \in Z : \exists_{x \in X}\, z \equiv x \pmod q\},$$

represented as a disjoint union of at most three interval chains, each with difference q, can be computed in $\mathcal{O}(1)$ time.

Proof. If $|X| \ge q$, then $Z' = Z$ is an interval and thus an interval chain. If $|X| < q$, then Z' can be divided into disjoint intervals of length smaller than or equal to $|X|$. The intervals from the second until the penultimate one (if any such exist) have length $|X|$. Hence, they can be represented as a single chain, as the first element of each such interval is equal $\bmod\ q$ to the first element of X. The two remaining intervals can be treated as chains as well. $\square$

4.3. The aligned-light-sum problem

We define the following abstract problem that resembles the LIGHT-FRAGMENTS problem from Section 3.

> ALIGNED-LIGHT-SUM PROBLEM
>
> **Input:** Positive integers m, k, q and strings U, V over alphabet $\{0, 1\}$, each containing $\mathcal{O}(k)$ non-zero characters. The strings are specified by their positions with non-zero characters.
>
> **Output:** The set $A = \{i : \exists_j \ \|U[i \mathinner{.\,.} i+m-1]\| + \|V[j \mathinner{.\,.} j+m-1]\| \le k \ \wedge \ j \equiv i \pmod q\}$.

Lemma 8. *The* ALIGNED-LIGHT-SUM *problem can be solved in $\mathcal{O}(k^2)$ time with the output represented as a collection of $\mathcal{O}(k^2)$ interval chains, each with difference q.*

Proof. Let I and I' be the positions with non-zero characters in U and V, respectively. We partition $[0 \mathinner{.\,.} |U|-m]$ into intervals such that, for all indices j in a single interval, the set $\mathbf{W}_j \cap I$ is the same. For this, we use a sliding window approach. We generate events corresponding to x and $x-m+1$ for all $x \in I$ and sort them. When j crosses an event, the set $\mathbf{W}_j \cap I$ changes. Thus, we obtain a partition of $[0 \mathinner{.\,.} |U|-m]$ into intervals $Z_1, \dots, Z_{n_1}$. We obtain a similar partition of $[0 \mathinner{.\,.} |V|-m]$ into intervals $Z'_1, \dots, Z'_{n_2}$. We have $n_1, n_2 = \mathcal{O}(k)$.

Let us now fix Z_j and $Z'_{j'}$. First, we check if the condition on the total number of non-zero characters is satisfied for arbitrary $z \in Z_j$ and $z' \in Z'_{j'}$. If so, we compute the set $X = Z'_{j'} \bmod q = \{z' \bmod q : z' \in Z'_{j'}\}$. It is a single circular interval and can be computed in constant time. The required result is

$$\{z \in Z_j : z \bmod q \in X\}.$$

By Lemma 7, this set can be represented as a union of three chains, each with difference q, and, as such, it can be computed in $\mathcal{O}(1)$ time. The conclusion follows. □

5. An $\mathcal{O}(n+k^5)$-time algorithm for short texts

In this section, we proceed by assuming that $m \le n \le 2m$ and aim at an $\mathcal{O}(n+k^5)$-time algorithm. In the next sections, we remove this assumption and reduce the exponent of k to 4.

A *(deterministic) sample* is a short fragment $\mathcal{S}$ of the pattern P. An occurrence in the text without any mismatch is called *exact*. We introduce a problem of SAMPLE-MATCH that consists in finding all k-occurrences of P in T such that $\mathcal{S}$ matches exactly the corresponding length-$|\mathcal{S}|$ fragment of T.

We split the pattern P into $2k+3$ samples of length $\left\lfloor \frac{m}{2k+3} \right\rfloor$ or $\left\lceil \frac{m}{2k+3} \right\rceil$ each. In any k-occurrence of P in T, at least $k+2$ of the samples match exactly the corresponding fragments of T (up to k samples may match with a mismatch and at most one sample may contain the split point).

Remark 9. We require at least $k+2$ samples (instead of just one) to match exactly for two reasons: (1) in order to have more than a half of them match exactly, which will guarantee that the interval chains that are obtained from applications of the ALIGNED-LIGHT-SUM problem have the same difference and thus can be unioned using Lemma 5 (see the proof of Proposition 21); and (2) for the marking trick in the next section.

5.1. Matching non-periodic samples

We solve the SAMPLE-MATCH problem for a non-periodic sample $\mathcal{S}$ in $\mathcal{O}(k^2)$ time in two steps. First, we show an $\mathcal{O}(k)$-time solution of following PAIR-MATCH problem, asking to compute all k-occurrences of P in T which align $T[i]$ with $P[j]$.

> PAIR-MATCH PROBLEM
>
> **Input:** A text T of length n, a pattern P of length m, a positive integer k, and two integers $i \in [0 \mathinner{.\,.} n-1]$ and $j \in [0 \mathinner{.\,.} m-1]$.
>
> **Output:** The set $A(i, j)$ of all positions in T where we have a k-mismatch occurrence of $\mathrm{rot}_x(P)$ for some x such that (i, j) is a matching pair.

We then reduce the SAMPLE-MATCH problem to $\mathcal{O}(k)$ instances of the PAIR-MATCH problem, where j is the starting position of $\mathcal{S}$ in P and i is an occurrence of $\mathcal{S}$ in T; notice that there are $\mathcal{O}(k)$ such occurrences.

Lemma 10. *After $\mathcal{O}(n)$-time preprocessing of T and P, the* PAIR-MATCH *problem can be solved in $\mathcal{O}(k)$ time for any given k, i, j, with the output represented as a collection of $\mathcal{O}(k)$ intervals.*

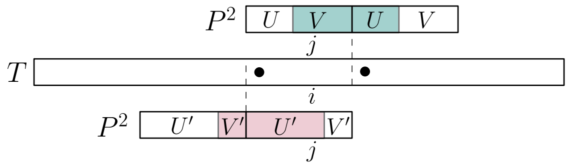

Fig. 5. The two possible anchors for the matching pair of positions (i, j) are shown as bullet points. A possible k-occurrence of P in T corresponding to the left (resp. right) anchor is shown below T (above T, resp.). Note that $P^2 = PP$.

Proof. Recall that the Anchor-Match problem returns all k-occurrences of P in T with a given anchor. The Pair-Match problem can be essentially reduced to the Anchor-Match problem, since for a given matching pair of characters in P and T, there are at most two ways of choosing the anchor depending on the relation between j and a split point: these are $i - j$ and $i + |P| - j$ (see Fig. 5). Clearly, we choose $i - j$ as an anchor only if $i - j \geq 0$ and $i + |P| - j$ only if $i + |P| - j < |T|$. We then have to take the intersection of the answer with $[i - m + 1 \,.\,.\, i]$ to ensure that the k-occurrence contains position i. $\square$

Lemma 11. *After $\mathcal{O}(n)$-time preprocessing, the* Sample-Match *problem for a non-periodic sample $\mathcal{S}$ can be solved in $\mathcal{O}(k^2)$ time, which the output represented as a union of $\mathcal{O}(k^2)$ intervals of occurrences.*

Proof. Since $\mathcal{S}$ is non-periodic, it has $\mathcal{O}(k)$ occurrences in T, which can be computed in $\mathcal{O}(k)$ time after an $\mathcal{O}(n)$-time preprocessing using IPM queries [37,38] in $P\#T$. Let j be the starting position of $\mathcal{S}$ in P and i be a starting position of an occurrence of $\mathcal{S}$ in T. For each of the $\mathcal{O}(k)$ such pairs (i, j), the computation reduces to the Pair-Match problem for i and j. The statement follows by Lemma 10. $\square$

5.2. Matching periodic samples

Let us assume that $\mathcal{S}$ is periodic with $q := \mathsf{per}(\mathcal{S}) \leq \frac{1}{2}|\mathcal{S}|$. A fragment of T containing an inclusion-maximal arithmetic sequence of occurrences of $\mathcal{S}$ in T with difference q is called here an *$\mathcal{S}$-run*. If $\mathcal{S}$ matches a fragment in the text, then the match belongs to an $\mathcal{S}$-run. For example, the underlined fragment of $T = \texttt{bb}\underline{\texttt{abababab}}\texttt{aa}$ is an $\mathcal{S}$-run for $\mathcal{S} = \texttt{abab}$.

Lemma 12. *If $\mathcal{S}$ is periodic, the number of $\mathcal{S}$-runs in the text is $\mathcal{O}(k)$ and they can all be computed in $\mathcal{O}(k)$ time after $\mathcal{O}(n)$-time preprocessing of T and P.*

Proof. We construct the data structure for IPM queries on $P\#T$. This allows us to compute the set of all occurrences of $\mathcal{S}$ in T as a collection of $\mathcal{O}(k)$ arithmetic sequences with difference $\mathsf{per}(\mathcal{S})$. We then check for every two consecutive sequences if they can be joined together. This takes $\mathcal{O}(k)$ time and results in $\mathcal{O}(k)$ $\mathcal{S}$-runs. $\square$

For two equal-length strings S and S', we denote the set of their *mismatches* by

$$\mathsf{Mis}(S, S') = \{i \in [0\,.\,.\,|S| - 1] \,:\, S[i] \neq S'[i]\}.$$

We say that position a in S is a *misperiod* with respect to a fragment $S[i\,.\,.\,j]$ if $S[a] \neq S[b]$ where b is the unique position such that $b \in [i\,.\,.\,j]$ and $(j - i + 1) \mid (b - a)$. We define the set $\mathsf{LeftMisper}_k(S, i, j)$ as the set of k maximal misperiods that are smaller than i and $\mathsf{RightMisper}_k(S, i, j)$ as the set of k minimal misperiods that are greater than j. Each of the sets can have less than k elements if the corresponding misperiods do not exist. We further define

$$\mathsf{Misper}_k(S, i, j) = \mathsf{LeftMisper}_k(S, i, j) \cup \mathsf{RightMisper}_k(S, i, j)$$

and $\mathsf{Misper}(S, i, j) = \bigcup_{k=0}^{\infty} \mathsf{Misper}_k(S, i, j)$.

The following lemma captures a combinatorial property behind the new technique of Bringmann et al. [10]. The intuition is shown in Fig. 6.

Lemma 13. *Assume that $S =_k S'$ and that $S[i\,.\,.\,j] = S'[i\,.\,.\,j]$. Let*

$$I = \mathsf{Misper}_{k+1}(S, i, j) \text{ and } I' = \mathsf{Misper}_{k+1}(S', i, j).$$

If $I \cap I' = \emptyset$, then $\mathsf{Mis}(S, S') = I \cup I'$, $I = \mathsf{Misper}(S, i, j)$, and $I' = \mathsf{Misper}(S', i, j)$.

Proof. Let $J = \mathsf{Misper}(S, i, j)$ and $J' = \mathsf{Misper}(S', i, j)$. We first observe that $I \cup I' \subseteq \mathsf{Mis}(S, S')$ since $I \cap I' = \emptyset$. Then, $S =_k S'$ implies that $|\mathsf{Mis}(S, S')| \leq k$ and hence $|I| \leq k$ and $|I'| \leq k$, which in turn implies that $I = J$ and $I' = J'$. The observation that $\mathsf{Mis}(S, S') \subseteq J \cup J'$ concludes the proof. $\square$

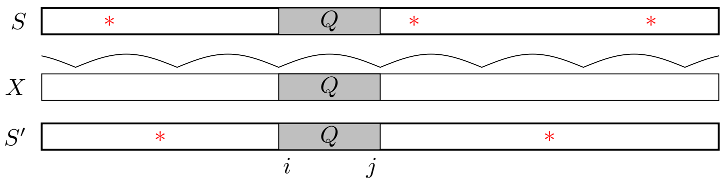

Fig. 6. Let S, S', and X be equal-length strings such that X is a factor of Q^∞ and $S[i\,..\,j] = S'[i\,..\,j] = X[i\,..\,j] = Q$. The asterisks in S denote the positions in $\mathsf{Mis}(S, X)$, or equivalently, the misperiods with respect to $S[i\,..\,j]$. Similarly for S'. One can observe that $\mathsf{Mis}(S, X) \cap \mathsf{Mis}(S', X) = \emptyset$ holds in this situation, and therefore $\mathsf{Mis}(S, X) \cup \mathsf{Mis}(S', X) = \mathsf{Mis}(S, S')$.

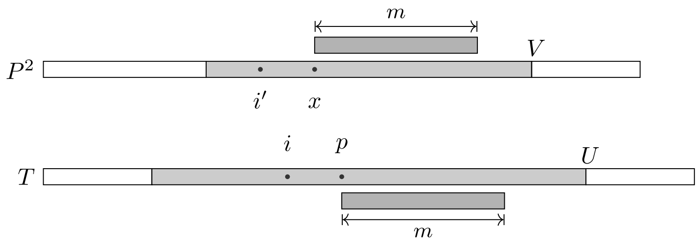

Fig. 7. In the case of a periodic sample, we have two $\mathcal{O}(k)$-periodic fragments U and V. We find all p in U such that for some position x in V, the fragments of length m starting at positions p and x are at Hamming distance at most k. If q is a period, then $i - p \equiv i' - x \pmod q$ due to synchronization of periodicities.

A string S is *k-periodic w.r.t. a fragment* $S[i\,..\,i+q-1]$ if $|\mathsf{Misper}(S, i, i+q-1)| \le k$. In this case, q is called the *k-period*. In particular, in the conclusion of the above lemma, S is $|I|$-periodic w.r.t. $S[i\,..\,j]$ and S' is $|I'|$-periodic w.r.t. $S'[i\,..\,j]$. This notion forms the basis of the following auxiliary problem, where we search for k-occurrences in which the rotation of the pattern and the fragment of the text are k-periodic for the same period q.

Let U and V be two strings and J and J' be sets containing positions in U and V, respectively. We say that length-m fragments $U[p\,..\,p+m-1]$ and $V[x\,..\,x+m-1]$ are *(J, J')-disjoint* if the sets $(\mathbf{W}_p \cap J) \ominus p$ and $(\mathbf{W}_x \cap J') \ominus x$ are disjoint.

Example 14. If $J = \{2, 4, 11, 15, 16, 17\}$, $J' = \{5, 6, 15, 18, 19\}$, and $m = 12$, then $U[3\,..\,14]$ and $V[6\,..\,17]$ are (J, J')-disjoint for the following strings, with characters at positions in J and J' replaced by bullets:

$$U = \texttt{ab}\bullet\boxed{\texttt{a}\bullet\texttt{b abc ab}\bullet\texttt{ abc}}\bullet\bullet\bullet$$
$$V = \texttt{abc ab}\bullet\boxed{\bullet\texttt{bc abc abc }\bullet\texttt{bc}}\bullet\bullet\texttt{c}$$

Let us introduce an auxiliary problem that is obtained in the case that is shown in the conclusion of the above lemma (i.e., when misperiods in the rotation of P and the corresponding fragment of T are not aligned); see also Fig. 7.

Periodic-Periodic-Match Problem

Input: Positive integers k and m, strings U and V such that $m \le |U|, |V| \le 2m$, integers i, i', q such that $U[i\,..\,i+q-1]$ matches $V[i'\,..\,i'+q-1]$, and two sets of size $\mathcal{O}(k)$:

$$J = \mathsf{Misper}(U, i, i+q-1), \quad J' = \mathsf{Misper}(V, i', i'+q-1).$$

(The strings U and V are not stored explicitly.)

Output: The set of positions p in U for which there exists a (J, J')-disjoint k-occurrence $U[p\,..\,p+m-1]$ of $V[x\,..\,x+m-1]$ for x such that

$$i - p \equiv i' - x \pmod q.$$

Intuitively, the modulo condition on the output of the Periodic-Periodic-Match problem corresponds to the fact that the approximate periodicity is aligned.

In the Periodic-Periodic-Match problem, we search for k-occurrences in which none of the misperiods in J and J' are aligned. In this case, each of the misperiods accounts for one mismatch in the k-occurrence. In the lemma below, we reduce the Periodic-Periodic-Match problem to the Aligned-Light-Sum problem, in which we only require that the total number of misperiods in an occurrence is at most k. This way, all the (J, J')-disjoint k-occurrences can be found. Also additional occurrences where two misperiods are aligned can be reported, but they are still valid k-occurrences (actually, k'-occurrences for some $k' < k$).

Data: A periodic fragment $\mathcal{S}$ of pattern P, an $\mathcal{S}$-run R in the text T, integers $q = \mathsf{per}(\mathcal{S})$ and k.
Result: A compact representation of k-occurrences of P in T including all k-occurrences where $\mathcal{S}$ matches exactly a fragment of R in T.
Let $R = T[s\,..\,s+|R|-1]$;
$J := \mathsf{Misper}_{k+1}(T, s, s+q-1)$; { $\mathcal{O}(k)$ time }
$J' := \mathsf{Misper}_{k+1}(P^2, m+p_{\mathcal{S}}, m+p_{\mathcal{S}}+q-1)$; { $\mathcal{O}(k)$ time }
$U := \mathsf{frag}_J(T)$; $V := \mathsf{frag}_{J'}(P^2)$;
$Y :=$ Periodic-Periodic-Match(U, V); { $\mathcal{O}(k^2)$ time }
$Y := Y \oplus \min(J)$;
$J' := J' \bmod m$;
$X :=$ Pairs-Match(T, J, P, J'); { $\mathcal{O}(k^3)$ time }
return $X \cup Y$;

Algorithm 1: Run-Sample-Matching.

Lemma 15. *We can compute in $\mathcal{O}(k^2)$ time a set of k-occurrences of P in T represented as $\mathcal{O}(k^2)$ interval chains, each with difference q, that is a superset of the solution to the* Periodic-Periodic-Match *problem.*

Proof. In the Periodic-Periodic-Match problem, the modulo condition forces the exact occurrences of the approximate period to match. Hence, it guarantees that all the positions except the misperiod positions match. Now, the Aligned-Light-Sum problem highlights these positions inside the string.

Claim 16. *The* Periodic-Periodic-Match *problem can be reduced in $\mathcal{O}(k)$ time to the* Aligned-Light-Sum *problem so that we obtain a superset of the desired result. The potential extra positions do not satisfy only the (J, J')-disjointness condition.*

Proof. Let the parameters m, k, and q remain unchanged. We create binary strings U' and V' of length $|U|$ and $|V|$, respectively, with positions with non-zero characters in the sets J and J', respectively. Then we prepend U' with $z = (i' - i) \bmod q$ zeros. Let A be the solution to the Aligned-Light-Sum problem for U' and V'. Then $(A \ominus z) \cap \mathbb{Z}_{\geq 0}$ is a superset of the solution to Periodic-Periodic-Match; the elements of the set that correspond to matches where non-zero elements of the strings U', V' were aligned do not satisfy the disjointness condition. □

Now, the thesis follows from Lemma 8. □

Let us further define

$$\text{Pairs-Match}(T, I, P, J) = \bigcup_{i \in I, j \in J} \text{Pair-Match}(T, i, P, j).$$

Let A be a set of positions in a string S and m be a positive integer. We then denote $A \bmod m = \{a \bmod m : a \in A\}$ and by $\mathsf{frag}_A(S)$ we denote the fragment $S[\min(A)\,..\,\max(A)]$. We provide pseudocode for an algorithm that computes all k-occurrences of P such that $\mathcal{S}$ matches an exact occurrence in T contained in a given $\mathcal{S}$-run (see Algorithm 1); inspect also Fig. 8. Let $p_{\mathcal{S}}$ denote the starting position of $\mathcal{S}$ in P, and let $m_{\mathcal{S}} = |\mathcal{S}|$.

Lemma 17. *After $\mathcal{O}(n)$-time preprocessing of T and P, algorithm Run-Sample-Matching works in $\mathcal{O}(k^3)$ time and returns a compact representation that consists of $\mathcal{O}(k^3)$ intervals and $\mathcal{O}(k^2)$ interval chains, each with difference q. Moreover, if there is at least one interval chain, then some rotation of the pattern P is k-periodic with a k-period $\mathsf{per}(\mathcal{S})$.*

Proof. See Algorithm 1. The sets J and J' can be computed in $\mathcal{O}(k)$ time:

Claim 18. *If S is a string of length n, then the sets $\mathsf{RightMisper}_k(S, i, j)$ and $\mathsf{LeftMisper}_k(S, i, j)$ can be computed in $\mathcal{O}(k)$ time after $\mathcal{O}(n)$-time preprocessing.*

Proof. For $\mathsf{RightMisper}_k(S, i, j)$, we use the kangaroo method [4,14] to compute the longest common prefix with at most k mismatches of $S[j+1\,..\,n-1]$ and U^∞ for $U = S[i\,..\,j]$. The value $\mathsf{lcp}(X^\infty, Y)$ for a fragment X and a suffix Y of a string S, occurring at positions a and b, respectively, can be computed in constant time as follows. If $\mathsf{lcp}(S[a\,..\,n-1], S[b\,..\,n-1]) < |X|$ then we are done. Otherwise the answer is given by $|X| + \mathsf{lcp}(S[b\,..\,n-1], S[b+|X|\,..\,n-1])$. The computations for $\mathsf{LeftMisper}_k(S, i, j)$ are symmetric. □

The $\mathcal{O}(k^3)$ and $\mathcal{O}(k^2)$ time complexities of computing X and Y follow from Lemmas 10 and 15, respectively (after $\mathcal{O}(n)$-time preprocessing). The sets X and Y consist of $\mathcal{O}(k^3)$ intervals and $\mathcal{O}(k^2)$ interval chains, each with difference q.

As for the "moreover" statement, by Lemma 13, if any occurrence q is reported in the Periodic-Periodic-Match problem, then it implies the existence of x such that $V[x\,..\,x+m-1]$ is k-periodic with a k-period q. However, $V[x\,..\,x+m-1]$ is a rotation of the pattern P. This concludes the proof. □

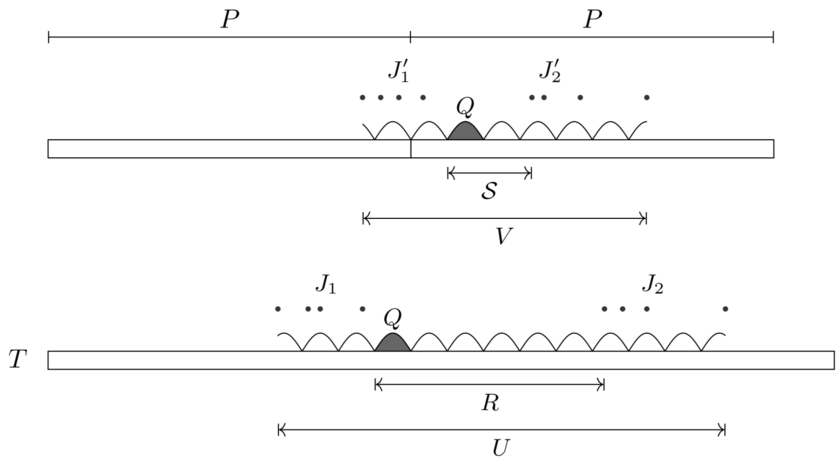

Fig. 8. More detailed setting in Algorithm 1; $J = J_1 \cup J_2$, $J' = J_1' \cup J_2'$.

The correctness of the algorithm follows from Lemma 13, as shown in the lemma below.

Lemma 19. *Assume $n \le 2m$. Let $\mathcal{S}$ be a periodic sample in P with smallest period q and R be an $\mathcal{S}$-run in T. Let X and Y be defined as in the pseudocode of Run-Sample-Matching. Then $X \cup Y$ is a set of k-occurrences of P in T which is a superset of the solution to* Sample-Match *for $\mathcal{S}$ in R.*

Proof. Both Pair-Match and Periodic-Periodic-Match problems return positions of k-occurrences of P in T. It suffices to show that $p \in X \cup Y$ if $\mathrm{rot}_x(P)$ has a k-mismatch occurrence in T at position p such that the designated fragment $\mathcal{S}$ matches a fragment of R exactly. We assume that the split point x in P is to the right of $\mathcal{S}$, i.e., that $x \ge p_{\mathcal{S}} + m_{\mathcal{S}}$. The opposite case—that $x < p_{\mathcal{S}}$—can be handled analogously.

Let $J = \mathsf{Misper}_{k+1}(T, s, s+q-1)$ and $J' = \mathsf{Misper}_{k+1}(P^2, m+p_{\mathcal{S}}, m+p_{\mathcal{S}}+q-1)$. We define L_1 and L_2 as the subsets of J and J', respectively, that are relevant for this k-occurrence, i.e.,

$$L_1 = J \cap \mathbf{W}_p, \quad L_2 = J' \cap \mathbf{W}_x.$$

Further let $L_2' = L_2 \bmod m$. If any $i \in L_1$ and $j \in L_2'$ are a matching pair for this k-occurrence, then it will be found in the Pairs-Match problem, i.e., $p \in X$. Let us henceforth consider the opposite case.

Let $S = T[p \,..\, p+m-1]$, $S' = \mathrm{rot}_x(P)$, and $i = m - x + p_{\mathcal{S}}$ be the starting position of $Q = \mathcal{S}[0 \,..\, q-1]$ in both strings. Further let $I = L_1 \ominus p$ and $I' = L_2 \ominus x$. We have that $I \cap I' = \emptyset$ by our assumption that misperiods do not align. We make the following claim.

Claim 20. $\mathsf{Mis}(S, S') = I \cup I'$.

Proof. Note that $I = \mathsf{Misper}_{k+1}(S, i, i+q-1)$ and $I' = \mathsf{Misper}_{k+1}(S', i, i+q-1)$. The former equality follows from the fact that $\mathsf{Misper}_{k+1}(T, s, s+q-1) = \mathsf{Misper}_{k+1}(T, t, t+q-1)$ for any $t \in [s \,..\, s+|R|-q]$. We can thus directly apply Lemma 13 to strings S and S'. $\square$

In particular, $|I| + |I'| \le k$. Moreover, $\min(J) < p$ and $p+m-1 < \max(J)$ as well as $\min(J') < x$ and $x+m-1 < \max(J')$, since otherwise we would have $|I| \ge k+1$ or $|I'| \ge k+1$. In conclusion, this k-occurrence will be found in the Periodic-Periodic-Match problem, i.e., $p \in Y$. $\square$

5.3. Algorithm summary

Proposition 21. *If $m \le n \le 2m$, the k-CPM problem can be solved in $\mathcal{O}(n + k^5)$ time and $\mathcal{O}(n)$ space.*

Proof. We split the pattern into $2k+3$ fragments and choose a sample $\mathcal{S}$ among them in every possible way.

If the sample $\mathcal{S}$ is not periodic, we use the algorithm of Lemma 11 for Sample-Match in $\mathcal{O}(k^2)$ time (after $\mathcal{O}(n)$-time preprocessing). It returns a representation of k-occurrences as a union of $\mathcal{O}(k^2)$ intervals.

If the sample $\mathcal{S}$ is periodic, we need to find all $\mathcal{S}$-runs in T. By Lemma 12, there are $\mathcal{O}(k)$ of them and they can all be computed in $\mathcal{O}(k)$ time (after $\mathcal{O}(n)$-time preprocessing). For every such $\mathcal{S}$-run R, we apply the Run-Sample-Matching algorithm. Its correctness follows from Lemma 19. By Lemma 17, it takes $\mathcal{O}(k^3)$ time and returns $\mathcal{O}(k^3)$ intervals and $\mathcal{O}(k^2)$

interval chains, each with difference $\mathsf{per}(\mathcal{S})$, of k-occurrences of P in T (after $\mathcal{O}(n)$-time preprocessing). Over all $\mathcal{S}$-runs, this takes $\mathcal{O}(k^4)$ time after the preprocessing and returns $\mathcal{O}(k^4)$ intervals and $\mathcal{O}(k^3)$ interval chains.

By Lemma 17, if any interval chains are reported in Run-Sample-Matching, then some rotation of the pattern is k-periodic with a k-period $\mathsf{per}(\mathcal{S})$. Then, at least $k+2$ of the $2k+3$ pattern fragments do not contain misperiods and hence they must have a period $q=\mathsf{per}(\mathcal{S})$. This is actually their smallest period, for if one of these fragments $\mathcal{S}'$ had a period $q'<q$, then $|\mathcal{S}'|\geq|\mathcal{S}|-1$ and, by Fine and Wilf's periodicity lemma [36], $\mathcal{S}'$ would have a period $q''=\gcd(q,q')<q$, which would imply that Q would also have a period q'', and hence $\mathcal{S}$ as well. Thus, throughout the course of the algorithm, Run-Sample-Matching can only return interval chains of period $\mathsf{per}(\mathcal{S})$ by the pigeonhole principle.

In total, Sample-Match takes $\mathcal{O}(k^4)$ time for a given sample (after preprocessing), $\mathcal{O}(n+k^5)$ time in total, and returns $\mathcal{O}(k^5)$ intervals and $\mathcal{O}(k^4)$ interval chains of k-occurrences, each with the same difference q. Let us note that an interval is a special case of an interval chain with an arbitrary difference, say, 1. We then apply Lemma 5 to compute the union of all chains of occurrences and the union of all intervals in $\mathcal{O}(n+k^5)$ total time. In the end, we return the union of the two unions.

In order to bound the space required by our algorithm by $\mathcal{O}(n)$, we do not store each interval chain explicitly throughout the execution of the algorithm. Instead, for each interval chain, we increment/decrement a constant number of cells in a ($\mathcal{G}_q$-shaped for interval chains or $\mathcal{G}_1$-shaped for intervals) 2D array of size $\mathcal{O}(n)$ as in the proof of Lemma 5, and compute the union of all such interval chains in the end. □

6. An $\mathcal{O}(n+k^4)$-time algorithm for short texts

Let us observe that for each non-periodic fragment $\mathcal{S}$, we have to solve $\mathcal{O}(k)$ instances of Pair-Match, while for each periodic fragment $\mathcal{S}$ and each $\mathcal{S}$-run, we obtain two sets J and J', each of cardinality $\mathcal{O}(k)$, where each pair of elements in $J\times J'$ requires us to solve an instance of Pair-Match. Further, recall that each instance of Pair-Match reduces to two calls to our $\mathcal{O}(k)$-time algorithm for Anchor-Match. We thus solve $\mathcal{O}(k^4)$ Anchor-Match instances in total, yielding a total time complexity of $\mathcal{O}(k^5)$. As can be seen in the proof of Proposition 21 and the pseudocode, this is the only bottleneck of our algorithm, with everything else requiring $\mathcal{O}(n+k^4)$ time. We will decrease the number of calls to Anchor-Match by using a marking trick.

We first present a simple application of the marking trick. Suppose that we are in the standard (non-circular) k-mismatch problem, where we are to find all k-mismatch occurrences of a pattern P of length m in a text T of length n, and $n\leq 2m$. Further, suppose that P is square-free, or, in other words, that it is nowhere periodic. Let us consider the following algorithm: We split the pattern into $k+1$ fragments of length roughly m/k each. Then, at each k-occurrence of P in T, at least one of the $k+1$ fragments must match exactly. We then find the $\mathcal{O}(k)$ such exact matches of each fragment in T and each of them nominates a position for a possible k-occurrence of P. We thus have $\mathcal{O}(k^2)$ candidate positions in total to verify.

Now consider the following refinement of this algorithm: We split the pattern into $2k$ fragments (instead of $k+1$), each of length roughly $m/(2k)$. Then, at each k-occurrence of P in T, at least k of the fragments must match exactly; we exploit this fact as follows: Each exact occurrence of a fragment in T gives a mark to the corresponding position for a k-occurrence of P. There are thus $\mathcal{O}(k^2)$ marks given in total. However, we only need to verify positions with at least k marks and these are now $\mathcal{O}(k)$ in total. An illustration is provided in Fig. 9.

Let us get back to the k-CPM problem. We have the following fact.

Fact 22. *The algorithm underlying Proposition 21 returns each k-occurrence either through a call to* Periodic-Periodic-Match *or through at least $k+2$ calls to* Anchor-Match.

Proof. Let us fix a k-occurrence p of P in T. Since at least $k+2$ out of the $2k+3$ samples must match exactly in this k-occurrence, the algorithm must return p through at least $k+2$ calls to Sample-Match. For each of these calls, the k-occurrence p is returned through a call to Periodic-Periodic-Match or through (at least one) call to Anchor-Match. Thus, if p is not returned by any of the calls to Periodic-Periodic-Match, it must be returned by at least $k+2$ calls to Anchor-Match. □

We run the algorithm yielding Proposition 21 with a single difference: Instead of processing each instance of Pair-Match separately, we apply the marking trick in order to decrease the exponent of k by one. This is achieved by a reduction in the number of calls to the algorithm that solves Anchor-Match. For each of the $\mathcal{O}(k^4)$ instances of Pair-Match, we mark the two possible anchors for a k-occurrence and note that only anchors with at least $k+2$ marks need to be verified; these are $\mathcal{O}(k^3)$ in total. Finally, for each such anchor, we apply our solution to the Anchor-Match problem, which requires $\mathcal{O}(k)$ time, hence obtaining an $\mathcal{O}(n+k^4)$-time algorithm. The correctness of this approach follows from Fact 22. We arrive at the following result.

Proposition 23. *If $m\leq n\leq 2m$, the k-CPM problem can be solved in $\mathcal{O}(n+k^4)$ time and $\mathcal{O}(n)$ space.*

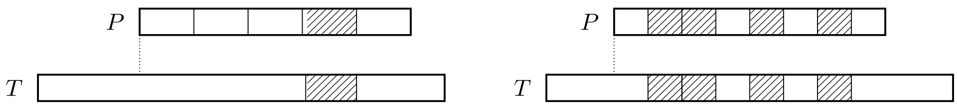

Fig. 9. We consider $k = 4$. To the left: a candidate starting position for P given by an exact match of one of the 5 fragments. To the right: a candidate starting position for P given by exact matches of 4 of the 8 fragments.

7. Final result

Both Propositions 3 and 23 use $\mathcal{O}(n)$ space. Moreover, Proposition 23 assumes that $n \le 2m$. In order to solve the general version of the k-CPM problem, where n is arbitrarily large, efficiently and using $\mathcal{O}(m)$ space, we use the so-called *standard trick*: we split the text into $\mathcal{O}(n/m)$ fragments, each of length $2m$ (perhaps apart from the last one), starting at positions equal to 0 mod m.

We need, however, to ensure that the data structures for answering lcp, lcs, and other internal queries over each such fragment of the text can be constructed in $\mathcal{O}(m)$ time when the input alphabet Σ is large. As a preprocessing step, we hash the letters of the pattern using perfect hashing. For each key, we assign a unique identifier from $\{1, \dots, m\}$. This takes $\mathcal{O}(m)$ time (with high probability) and space [41]. When reading a fragment F of length (at most) $2m$ of the text, we look up its letters using the hash table. If a letter is in the hash table, we replace it in F by its rank value; otherwise, we replace it by rank $m + 1$. We can now construct the data structures in $\mathcal{O}(m)$ time, and thus our algorithms can be implemented in $\mathcal{O}(m)$ space.

If $\Sigma = \{1, \dots, n^{\mathcal{O}(1)}\}$, the same bounds can be achieved deterministically. Specifically, we consider two cases. If $m > \sqrt{n}$, we sort the letters of every text fragment and of the pattern in $\mathcal{O}(m)$ time per fragment because n is polynomial in m and $|\Sigma|$ is polynomial in n. Then, we can merge the two sorted lists and replace the letters in the pattern and the text fragments by their ranks. Otherwise (if $m \le \sqrt{n}$), we construct a deterministic dictionary for the letters of the pattern in $\mathcal{O}(m \log^2 \log m)$ time [42]. The dictionary uses $\mathcal{O}(m)$ space and answers queries in constant time; we use it instead of perfect hashing in the previous solution.

We combine Propositions 3 and 23 with the above discussion to get our final result.

Theorem 24. *Circular Pattern Matching with k Mismatches can be solved in* $\mathcal{O}(\min(nk,\ n + \frac{n}{m} k^4))$ *time and* $\mathcal{O}(m)$ *space.*

Our algorithms output all positions in the text where some rotation of the pattern occurs with k mismatches. It is not difficult to extend the algorithms to output, for each of these positions, a corresponding witness rotation of the pattern.

CRediT authorship contribution statement

Panagiotis Charalampopoulos: Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Tomasz Kociumaka:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Solon P. Pissis:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Jakub Radoszewski:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Wojciech Rytter:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Juliusz Straszyński:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Tomasz Waleń:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Wiktor Zuba:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, 2007.
[2] K.R. Abrahamson, Generalized string matching, SIAM J. Comput. 16 (6) (1987) 1039–1051, https://doi.org/10.1137/0216067.
[3] S.R. Kosaraju, Efficient string matching, Manuscript (1987).
[4] G.M. Landau, U. Vishkin, Efficient string matching with k mismatches, Theor. Comput. Sci. 43 (1986) 239–249, https://doi.org/10.1016/0304-3975(86)90178-7.
[5] A. Amir, M. Lewenstein, E. Porat, Faster algorithms for string matching with k mismatches, J. Algorithms 50 (2) (2004) 257–275, https://doi.org/10.1016/S0196-6774(03)00097-X.
[6] R. Clifford, A. Fontaine, E. Porat, B. Sach, T. Starikovskaya, The k-mismatch problem revisited, in: R. Krautamer (Ed.), 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, SIAM, 2016, pp. 2039–2052.
[7] P. Gawrychowski, P. Uznański, Towards unified approximate pattern matching for Hamming and L_1 distance, in: I. Chatzigiannakis, C. Kaklamanis, D. Marx, D. Sannella (Eds.), Automata, Languages, and Programming, ICALP 2018, in: LIPIcs, vol. 107, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 62.

[8] T.M. Chan, S. Golan, T. Kociumaka, T. Kopelowitz, E. Porat, Approximating text-to-pattern Hamming distances, in: K. Makarychev, Y. Makarychev, M. Tulsiani, G. Kamath, J. Chuzhoy (Eds.), Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, ACM, 2020, pp. 643–656.

[9] P. Bille, R. Fagerberg, I.L. Gørtz, Improved approximate string matching and regular expression matching on Ziv-Lempel compressed texts, ACM Trans. Algorithms 6 (1) (2009) 3, https://doi.org/10.1145/1644015.1644018.

[10] K. Bringmann, P. Wellnitz, M. Künnemann, Few matches or almost periodicity: faster pattern matching with mismatches in compressed texts, in: T.M. Chan (Ed.), 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, SIAM, 2019, pp. 1126–1145.

[11] P. Gawrychowski, D. Straszak, Beating $\mathcal{O}(nm)$ in approximate LZW-compressed pattern matching, in: L. Cai, S. Cheng, T.W. Lam (Eds.), Algorithms and Computation, ISAAC 2013, in: Lecture Notes in Computer Science, vol. 8283, Springer, 2013, pp. 78–88.

[12] A. Tiskin, Threshold approximate matching in grammar-compressed strings, in: J. Holub, J. Zdárek (Eds.), Prague Stringology Conference 2014, PSC 2014, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2014, pp. 124–138, http://www.stringology.org/event/2014/p12.html.

[13] P. Charalampopoulos, T. Kociumaka, P. Wellnitz, Faster approximate pattern matching: a unified approach, CoRR, arXiv:2004.08350 [abs].

[14] Z. Galil, R. Giancarlo, Parallel string matching with k mismatches, Theor. Comput. Sci. 51 (1987) 341–348, https://doi.org/10.1016/0304-3975(87)90042-9.

[15] B. Porat, E. Porat, Exact and approximate pattern matching in the streaming model, in: 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, IEEE Computer Society, 2009, pp. 315–323.

[16] R. Clifford, T. Kociumaka, E. Porat, The streaming k-mismatch problem, in: T.M. Chan (Ed.), 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, SIAM, 2019, pp. 1106–1125.

[17] C. Hazay, M. Lewenstein, D. Sokol, Approximate parameterized matching, ACM Trans. Algorithms 3 (3) (2007) 29, https://doi.org/10.1145/1273340.1273345.

[18] P. Gawrychowski, P. Uznański, Order-preserving pattern matching with k mismatches, Theor. Comput. Sci. 638 (2016) 136–144, https://doi.org/10.1016/j.tcs.2015.08.022.

[19] L.A.K. Ayad, S.P. Pissis, MARS: improving multiple circular sequence alignment using refined sequences, BMC Genomics 18 (1) (2017) 86, https://doi.org/10.1186/s12864-016-3477-5.

[20] R. Grossi, C.S. Iliopoulos, R. Mercas, N. Pisanti, S.P. Pissis, A. Retha, F. Vayani, Circular sequence comparison: algorithms and applications, Algorithms Mol. Biol. 11 (2016) 12, https://doi.org/10.1186/s13015-016-0076-6.

[21] C.S. Iliopoulos, S.P. Pissis, M.S. Rahman, Searching and indexing circular patterns, in: M. Elloumi (Ed.), Algorithms for Next-Generation Sequencing Data, Techniques, Approaches, and Applications, Springer, 2017, pp. 77–90.

[22] C. Barton, C.S. Iliopoulos, R. Kundu, S.P. Pissis, A. Retha, F. Vayani, Accurate and efficient methods to improve multiple circular sequence alignment, in: E. Bampis (Ed.), Experimental Algorithms, SEA 2015, in: Lecture Notes in Computer Science, vol. 9125, Springer, 2015, pp. 247–258.

[23] L.A.K. Ayad, C. Barton, S.P. Pissis, A faster and more accurate heuristic for cyclic edit distance computation, Pattern Recognit. Lett. 88 (2017) 81–87, https://doi.org/10.1016/j.patrec.2017.01.018.

[24] V. Palazón-González, A. Marzal, Speeding up the cyclic edit distance using LAESA with early abandon, Pattern Recognit. Lett. 62 (2015) 1–7, https://doi.org/10.1016/j.patrec.2015.04.013.

[25] V. Palazón-González, A. Marzal, J.M. Vilar, On hidden Markov models and cyclic strings for shape recognition, Pattern Recognit. 47 (7) (2014) 2490–2504, https://doi.org/10.1016/j.patcog.2014.01.018.

[26] V. Palazón-González, A. Marzal, On the dynamic time warping of cyclic sequences for shape retrieval, Image Vis. Comput. 30 (12) (2012) 978–990, https://doi.org/10.1016/j.imavis.2012.08.012.

[27] K. Fredriksson, G. Navarro, Average-optimal single and multiple approximate string matching, ACM J. Exp. Algorithmics 9 (1.4) (2004) 1–47, https://doi.org/10.1145/1005813.1041513.

[28] C. Barton, C.S. Iliopoulos, S.P. Pissis, Fast algorithms for approximate circular string matching, Algorithms Mol. Biol. 9 (2014) 9, https://doi.org/10.1186/1748-7188-9-9.

[29] M.A.R. Azim, C.S. Iliopoulos, M.S. Rahman, M. Samiruzzaman, A fast and lightweight filter-based algorithm for circular pattern matching, in: P. Baldi, W. Wang (Eds.), 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 2014, ACM, 2014, pp. 621–622.

[30] M.A.R. Azim, C.S. Iliopoulos, M.S. Rahman, M. Samiruzzaman, A filter-based approach for approximate circular pattern matching, in: R.W. Harrison, Y. Li, I.I. Mandoiu (Eds.), Bioinformatics Research and Applications, ISBRA 2015, in: Lecture Notes in Computer Science, vol. 9096, Springer, 2015, pp. 24–35.

[31] C. Barton, C.S. Iliopoulos, S.P. Pissis, Average-case optimal approximate circular string matching, in: A. Dediu, E. Formenti, C. Martín-Vide, B. Truthe (Eds.), Language and Automata Theory and Applications, LATA 2015, in: Lecture Notes in Computer Science, vol. 8977, Springer, 2015, pp. 85–96.

[32] T. Hirvola, J. Tarhio, Bit-parallel approximate matching of circular strings with k mismatches, ACM J. Exp. Algorithmics 22 (2017) 1.5, https://doi.org/10.1145/3129536.

[33] W.I. Chang, T.G. Marr, Approximate string matching and local similarity, in: M. Crochemore, D. Gusfield (Eds.), Combinatorial Pattern Matching, CPM 1994, in: Lecture Notes in Computer Science, vol. 807, Springer, 1994, pp. 259–273.

[34] M. Lothaire, Applied Combinatorics on Words, Cambridge University Press, 2005, http://www.cambridge.org/gb/knowledge/isbn/item1172552/?site_locale=en_GB.

[35] P. Charalampopoulos, T. Kociumaka, S.P. Pissis, J. Radoszewski, W. Rytter, J. Straszyński, T. Waleń, W. Zuba, Circular pattern matching with k mismatches, in: L.A. Gąsieniec, J. Jansson, C. Levcopoulos (Eds.), Fundamentals of Computation Theory - 22nd International Symposium, FCT 2019, in: Lecture Notes in Computer Science, vol. 11651, Springer, 2019, pp. 213–228.

[36] N.J. Fine, H.S. Wilf, Uniqueness theorems for periodic functions, Proc. Am. Math. Soc. 16 (1) (1965) 109–114, https://doi.org/10.2307/2034009.

[37] T. Kociumaka, J. Radoszewski, W. Rytter, T. Waleń, Internal pattern matching queries in a text and applications, in: P. Indyk (Ed.), 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, SIAM, 2015, pp. 532–551.

[38] T. Kociumaka, Efficient data structures for internal queries in texts, Ph.D. thesis, University of Warsaw, Oct. 2018, https://www.mimuw.edu.pl/~kociumaka/files/phd.pdf.

[39] T. Kociumaka, J. Radoszewski, W. Rytter, J. Straszyński, T. Waleń, W. Zuba, Efficient representation and counting of antipower factors in words, in: C. Martín-Vide, A. Okhotin, D. Shapira (Eds.), Language and Automata Theory and Applications, LATA 2019, in: Lecture Notes in Computer Science, vol. 11417, Springer, 2019, pp. 421–433, full version at, https://arxiv.org/abs/1812.08101.

[40] M. Pătraşcu, Unifying the landscape of cell-probe lower bounds, SIAM J. Comput. 40 (3) (2011) 827–847, https://doi.org/10.1137/09075336X.

[41] M.L. Fredman, J. Komlós, E. Szemerédi, Storing a sparse table with $\mathcal{O}(1)$ worst case access time, J. ACM 31 (3) (1984) 538–544, https://doi.org/10.1145/828.1884.

[42] M. Ružić, Constructing efficient dictionaries in close to sorting time, in: L. Aceto, I. Damgård, L.A. Goldberg, M.M. Halldórsson, A. Ingólfsdóttir, I. Walukiewicz (Eds.), Automata, Languages and Programming, Part I, ICALP 2008, in: Lecture Notes in Computer Science, vol. 5125, Springer, 2008, pp. 84–95.