View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

The Streaming k-Mismatch Problem: Tradeoffs Between Space and Total Time

Shay Golan
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
golansh1@cs.biu.ac.il

Tomasz Kociumaka
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Tsvi Kopelowitz
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kopelot@gmail.com

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

Abstract

We revisit the k-mismatch problem in the streaming model on a pattern of length m and a streaming text of length n, both over a size-σ alphabet. The current state-of-the-art algorithm for the streaming k-mismatch problem, by Clifford et al. [SODA 2019], uses $\tilde{O}(k)$ space and $\tilde{O}\big(\sqrt{k}\big)$ worst-case time per character. The space complexity is known to be (unconditionally) optimal, and the worst-case time per character matches a conditional lower bound. However, there is a gap between the total time cost of the algorithm, which is $\tilde{O}(n\sqrt{k})$, and the fastest known offline algorithm, which costs $\tilde{O}\big(n + \min\big(\frac{nk}{\sqrt{m}}, \sigma n\big)\big)$ time. Moreover, it is not known whether improvements over the $\tilde{O}(n\sqrt{k})$ total time are possible when using more than $O(k)$ space.

We address these gaps by designing a randomized streaming algorithm for the k-mismatch problem that, given an integer parameter $k \le s \le m$, uses $\tilde{O}(s)$ space and costs $\tilde{O}\big(n + \min\big(\frac{nk^2}{m}, \frac{nk}{\sqrt{s}}, \frac{\sigma nm}{s}\big)\big)$ total time. For $s = m$, the total runtime becomes $\tilde{O}\big(n + \min\big(\frac{nk}{\sqrt{m}}, \sigma n\big)\big)$, which matches the time cost of the fastest offline algorithm. Moreover, the worst-case time cost per character is still $\tilde{O}\big(\sqrt{k}\big)$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Streaming pattern matching, Hamming distance, k-mismatch

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.15

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.12881`.

Funding This work was supported in part by ISF grants no. 1278/16 and 1926/19, by a BSF grant no. 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).

1 Introduction

In the fundamental Hamming distance problem, given two same-length strings X and Y, the goal is to compute $\mathsf{Ham}(X, Y)$, which is the number of aligned mismatches between X and Y. In the pattern matching version of the Hamming distance problem, the input is a pattern P of length m and a text T of length n, both over a size-σ alphabet, and the goal is to compute the Hamming distance between P and every length-m substring of T. In this paper, we focus on a well studied generalization known as the *k-mismatch* problem [1, 2, 4, 7, 10, 11, 12, 15, 18, 19, 22], which is the "fixed-threshold" version of the

© Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat;
licensed under Creative Commons License CC-BY

pattern matching Hamming distance problem: for a given parameter k, for each length-m substring S of T, if $\mathsf{Ham}(P,S) \leq k$, then compute $\mathsf{Ham}(P,S)$, and otherwise, report that $\mathsf{Ham}(P,S) > k$. Currently, the state-of-the-art (offline) algorithms for the k-mismatch problem are: (1) the algorithm of Fischer and Paterson [11], whose runtime is $\tilde{O}(\sigma n)$, and (2) the algorithm of Gawrychowski and Uznański [15], whose runtime is $\tilde{O}\big(n + \frac{nk}{\sqrt{m}}\big)$; see also [4].

The online and streaming models. The growing size of strings to be processed, often exceeding the available memory limits, motivated the study of pattern matching in the streaming model, where the characters of T arrive in a stream one at a time, and every occurrence of P needs to be identified as soon as the last character of the occurrence arrives [3, 4, 6, 7, 8, 13, 16, 17, 21, 20, 23]. In the *streaming k-mismatch problem*, the goal is to compute the Hamming distance between P and the current length-m suffix of T after each new character arrives, unless the Hamming distance is larger than k (which the algorithm reports in this case). Algorithms in the streaming model are typically required to use space of size sublinear in m. A closely related model is the *online model*, where the space usage of the algorithm is no longer explicitly limited.

Porat and Porat [20] introduced the first streaming k-mismatch algorithm using $\tilde{O}(k^2)$ time per character and $\tilde{O}(k^3)$ space. Subsequent improvements [7, 16] culminated in an algorithm by Clifford et al. [8], which solves the streaming k-mismatch problem in $\tilde{O}\big(\sqrt{k}\big)$ time per character using $\tilde{O}(k)$ space. The total time cost of $\tilde{O}\big(n\sqrt{k}\big)$ matches the time cost of the offline algorithm of Amir et al. [2], and the worst-case per-character running time matches a recent lower bound (valid for $\sigma = \Omega\big(\sqrt{k}\big)$ even with unlimited space usage) by Gawrychowski and Uznański [14], conditioned on the combinatorial Boolean matrix multiplication conjecture. However, the $\tilde{O}\big(n + \frac{nk}{\sqrt{m}}\big)$ total time cost of the offline algorithm of Gawrychowski and Uznański [15] is smaller, and the $\tilde{O}(\sigma n)$ total time cost of the offline algorithm of Fischer and Paterson [11] is smaller for small σ.

In the online model, where $O(m)$ space usage is allowed, the fastest algorithms follow from a generic reduction by Clifford et al. [5], which shows that if the offline k-mismatch problem can be solved in $O(n \cdot t(m,k))$ time, then the online k-mismatch problem can be solved in $O(n \sum_{i=0}^{\lceil \log m \rceil} t(2^i,k))$ time. In particular, this yields online algorithms with a total runtime of $\tilde{O}\big(n\sqrt{k}\big)$ and $\tilde{O}(n\sigma)$. Nevertheless, this approach cannot benefit from the state-of-the-art the offline algorithm of Gawrychowski and Uznański [15] since the running time of this algorithm degrades as m decreases. Thus, a natural question arises:

▶ **Question 1.** Is there an online/streaming algorithm for the k-mismatch problem whose total time cost is $\tilde{O}\big(n + \min\big(\frac{nk}{\sqrt{m}}, \sigma n\big)\big)$?

Space usage. It is straightforward to show that any streaming algorithm for the k-mismatch problem must use $\Omega(k)$ space [8]. Thus, the space usage of the algorithm of Clifford et al. [8] is optimal. Remarkably, we are unaware of any other tradeoffs between (sublinear) space usage and runtime for the k-mismatch problem. This leads to the following natural question.

▶ **Question 2.** Is there a time-space tradeoff algorithm for the k-mismatch problem, using $s \geq \Omega(k)$ space?

Our results. We address both Question 1 and Question 2 by proving the following theorem.

▶ **Theorem 3.** *There exists a randomized streaming algorithm for the k-mismatch problem that, given an integer parameter $k \le s \le m$, costs $\tilde{O}\big(n + \min\big(\frac{nk^2}{m}, \frac{nk}{\sqrt{s}}, \frac{\sigma nm}{s}\big)\big)$ total time and uses $\tilde{O}(s)$ space. Moreover, the worst-case time cost per character is $\tilde{O}(\sqrt{k})$. The algorithm is correct with high probability*[1].

Theorem 3 answers Question 2 directly. However, for Question 1, Theorem 3 only addresses the online setting, where $s = m$ can be set: since $k < \sqrt{m}$ yields $n > \frac{nk}{\sqrt{m}} > \frac{nk^2}{m}$, the total time cost is $\tilde{O}\big(n + \min\big(\frac{nk^2}{m}, \frac{nk}{\sqrt{m}}, \frac{\sigma nm}{m}\big)\big) = \tilde{O}\big(n + \min\big(\frac{nk}{\sqrt{m}}, \sigma n\big)\big)$. However, Question 1 remains open for the streaming model.

Another natural research direction is to extend Theorem 3 so that the pattern P could also be processed in a streaming fashion using $\tilde{O}(s)$ space, $\tilde{O}(\sqrt{k})$ time per character, and $\tilde{O}\big(m + \min\big(k^2, \frac{mk}{\sqrt{s}}, \frac{\sigma m^2}{s}\big)\big)$ time in total. To the best of our knowledge, among the existing streaming k-mismatch algorithms, only that of Clifford et al. [8] is accompanied with an efficient streaming procedure for preprocessing the pattern.

2 Algorithmic Overview and Organization

A string S of length $|S| = n$ is a sequence of characters $S[0]S[1]\cdots S[n-1]$ over an alphabet Σ. A *substring* of S is denoted by $S[i\,..\,j] = S[i]S[i+1]\cdots S[j]$ for $0 \le i \le j < n$. If $i = 0$, the substring is called a *prefix* of S, and if $j = n-1$, the substring is called a *suffix* of S. For two strings S and S' of the same length $|S| = n = |S'|$, we denote by $\mathsf{Ham}(S, S')$ the Hamming distance of S and S', that is, $\mathsf{Ham}(S, S') = |\{0 \le i \le n-1 : S[i] \ne S'[i]\}|$. An integer ρ is a d-period of a string S if $\mathsf{Ham}(S[0\,..\,n-\rho-1], S[\rho\,..\,n-1]) \le d$.

2.1 Overview

To prove Theorem 3, we consider two cases, depending on whether or not there exists an integer $\rho \le k$ that is a d-period of P for some $d = O(k)$. If such a ρ exists, then we say that P is *periodic*[2], and otherwise P is said to be *aperiodic*.

Tail partitioning. In both cases of whether P is periodic or not, our algorithms use the well-known *tail partitioning* technique [6, 7, 8, 9, 17], which decomposes P into two substrings: a suffix P_{tail} and the complementary prefix P_{head} of length $m - |P_{tail}|$. Accordingly, the algorithm has two components. The first component computes the Hamming distance of P_{head} and every length-$|P_{head}|$ substring of T with some delay: the reporting of $\mathsf{Ham}(P_{head}, T[i-|P|+1\,..\,i-|P_{tail}|])$ is required to be completed before the arrival of $T[i]$. The second component computes the Hamming distance of P_{tail} and carefully selected length-$|P_{tail}|$ substrings of T. The decision mechanism for selecting substrings for the second component is required to guarantee that whenever $\mathsf{Ham}(P_{head}, T[i-|P|+1\,..\,i-|P_{tail}|]) \le k$: if $\mathsf{Ham}(P_{tail}, T[i-|P_{tail}|+1\,..\,i]) \le k$ then the second component computes $\mathsf{Ham}(P_{tail}, T[i-|P_{tail}|+1\,..\,i])$; otherwise, the second component reports $\mathsf{Ham}(P_{tail}, T[i-|P_{tail}|+1\,..\,i]) > k$. The second component has no delay.

[1] An event $\mathcal{E}$ happens with high probability if $\Pr[\mathcal{E}] \ge 1 - n^{-c}$ for a constant parameter $c \ge 1$.
[2] The classic notion of periodicity is usually much simpler than the one we define here. However, since in this paper we do not use the classic notion of periodicity, we slightly abuse the terminology.

Notice that if either $\mathsf{Ham}(P_{head}, T[i-|P|\,..\,i-|P_{tail}|]) > k$, which is detected by the first component, or $\mathsf{Ham}(P_{tail}, T[i-|P_{tail}|+1\,..\,i]) > k$, which is detected by the second component, then it must be that $\mathsf{Ham}(P, T[i-|P|\,..\,i]) > k$. Otherwise, $\mathsf{Ham}(P, T[i-|P|\,..\,i])$ is computed by summing $\mathsf{Ham}(P_{head}, T[i-|P|\,..\,i-|P_{tail}|])$ and $\mathsf{Ham}(P_{tail}, T[i-|P_{tail}|+1\,..\,i])$. In either case, the information is available for the algorithm right after $T[i]$ arrives.

Thus, our algorithm has four main components, depending on whether P is periodic or not, and depending on the head or tail case of the tail partitioning technique.

The aperiodic case. The algorithms for the aperiodic case are a combination of straightforward modifications of previous work together with the naïve algorithm; the details are given in Section 7. Nevertheless, we provide an overview below. In this case, $|P_{tail}| = 2k$.

The algorithm for P_{head} in the aperiodic case is a slight modification of an algorithm designed by Golan et al. [16], which reduces the streaming k-mismatch problem to the problem of finding occurrences of multiple patterns in multiple text-streams.

The algorithm for P_{tail} in the aperiodic case is the naïve algorithm of comparing all aligned pairs of characters. While in general the naïve algorithm could cost $O(|P_{tail}|)$ time per character, in our setting the algorithm uses the output of the algorithm on P_{head} as a filter, and so the algorithm computes $\mathsf{Ham}(P_{tail}, T[i-|P_{tail}|+1\,..\,i])$ only if $\mathsf{Ham}(P_{head}, T[i-|P|+1\,..\,i-|P_{tail}|]) \leq k$. Since P is aperiodic, we are able to show that occurrences of P_{head} are distant enough so that the naïve algorithm for P_{tail} costs $\tilde{O}(1)$ worst-case time per character. In order for the filter to be effective, instead of guaranteeing that the algorithm for computing $\mathsf{Ham}(P_{head}, T[i-|P|+1\,..\,i-|P_{tail}|])$ is completed before $T[i]$ arrives, we refine the tail partitioning technique so that the computation of $\mathsf{Ham}(P_{head}, T[i-|P|+1\,..\,i-|P_{tail}|])$ completes before $T[i-\frac{1}{2}|P_{tail}|]$ arrives, and if the naïve algorithm should be used, the execution takes place through the arrivals of the subsequent $\frac{1}{2}|P_{tail}|$ characters $T[i-\frac{1}{2}|P_{tail}|+1\,..\,i]$. The effects of this refinement on the runtime is only by constant multiplicative factors.

The periodic case. We begin by first assuming that P and T have a common $O(k)$-period $\rho \leq k$, and that $n \leq \frac{3}{2}m$. In this case, we represent the strings as characteristic functions (one function for each character in Σ). Since both P and T are assumed to be periodic, each characteristic function, when treated as a string, is also periodic. Next, we use the notion of *backward differences*: for any function $f : \mathbb{Z} \to \mathbb{Z}$, the *backward difference* of f due to ρ is $\Delta_\rho[f](i) = f(i) - f(i-\rho)$. Clifford et al. [8] showed that the Hamming distance of two strings can be derived from a summation of convolutions of backward differences due to ρ of characteristic functions; see Section 3.

In the case of P_{head}, a delay of up to $2s$ characters is allowed. To solve this case, we define the problem of computing the convolutions of the backward differences in batches; the details for this case are given in Section 4. Our solution uses an offline algorithm for computing the convolutions described in Section 3.1. In the case of P_{tail}, we use a solution for the online version of computing the convolutions of the backward differences, which is adapted from Clifford et al. [5]; the details are given in Section 5. In both cases, since we assume that P and T are periodic, our algorithms leverage the fact that the backward differences of the characteristic functions have a small number of non-zero entries. This lets the algorithms compute the Hamming distance of P_{tail} and every substring of T which has length $|P_{tail}|$.

In Section 6, we remove the periodicity assumption on T by applying a technique by Clifford et al. [7] which identifies at most one periodic region of T that contains all the k-mismatch occurrences of P. Moreover, we drop the $n \leq \frac{3}{2}m$ assumption using a standard trick of partitioning T into overlapping fragments of length $\frac{3}{2}m$.

3 Hamming Distance and the Convolution Summation Problem

Recall that the *support* of a function f is $\text{supp}(f) := \{x \mid f(x) \neq 0\}$. Let $|f| = |\text{supp}(f)|$. Throughout, we only consider functions with finite support mapping $\mathbb{Z}$ to $\mathbb{Z}$. The *convolution* of two functions $f, g : \mathbb{Z} \to \mathbb{Z}$ is a function $f * g : \mathbb{Z} \to \mathbb{Z}$ such that

$$[f * g](i) = \sum_{j \in \mathbb{Z}} f(j) \cdot g(i - j).$$

For a string X and a character $c \in \Sigma$, the *characteristic function* of X and c is $X_c : \mathbb{Z} \to \{0, 1\}$ such that $X_c(i) = 1$ if and only if $X[i] = c$. For a string X, let X^R be X reversed. The *cross-correlation* of strings X and Y over Σ is a function $X \otimes Y : \mathbb{Z} \to \mathbb{Z}$ such that

$$X \otimes Y = \sum_{c \in \Sigma} X_c * Y_c^R.$$

▶ **Lemma 4** ([8, Fact 7.1]). *Let P, T be strings. For $|P| - 1 \leq i < |T|$, we have $[T \otimes P](i) = |P| - \mathsf{Ham}(P, T[i - |P| + 1 \,..\, i])$. For $i < 0$ and for $i \geq |P| + |T|$, we have $[T \otimes P](i) = 0$.*

By Lemma 4, in order to compute $\mathsf{Ham}(P, T[i - |P| + 1 \,..\, i])$, it suffices to compute $[T \otimes P](i)$.

The *backward difference* of a function $f : \mathbb{Z} \to \mathbb{Z}$ due to ρ is $\Delta_\rho[f](i) = f(i) - f(i - \rho)$.

▶ **Observation 5** ([8, Obs. 7.2]). *If a string X has a d-period ρ, then $\sum_{c \in \Sigma} |\Delta_\rho[X_c]| \leq 2(d + \rho)$.*

Our computation of $T \otimes P$ in a streaming fashion is based on the following lemma:

▶ **Lemma 6** (Based on [8, Fact 7.4 and Corollary 7.5]). *For every $i \in \mathbb{Z}$ and $\rho \in \mathbb{Z}_+$, we have $[T \otimes P](i) = \left[\sum_{c \in \Sigma} \Delta_\rho[T_c] * \Delta_\rho[P_c^R]\right](i) - [T \otimes P](i - 2\rho) + 2[T \otimes P](i - \rho)$.*

When computing $[T \otimes P](i)$, if the algorithm maintains a buffer of the last 2ρ values of $T \otimes P$, then the algorithm already has the values of $[T \otimes P](i - \rho)$ and $[T \otimes P](i - 2\rho)$. Thus, in order to construct $T \otimes P$, the focus is on constructing $\sum_{c \in \Sigma} \Delta_\rho[T_c] * \Delta_\rho[P_c^R]$.

3.1 Convolution Summation Problem

We express the task of constructing $\sum_{c \in \Sigma} \Delta_\rho[T_c] * \Delta_\rho[P_c^R]$ in terms of a more abstract *convolution summation* problem stated as follows. The input is two sequences of functions $\mathcal{F} = (f_1, f_2, \ldots, f_t)$ and $\mathcal{G} = (g_1, g_2, \ldots g_t)$ such that for every $1 \leq i \leq t$ we have $f_i : \mathbb{Z} \to \mathbb{Z}$ and $g_i : \mathbb{Z} \to \mathbb{Z}$, and the goal is to construct the function $\mathcal{F} \otimes \mathcal{G} = \sum_{j=1}^{t} (f_j * g_j)$.

Let $\mathcal{H}$ be a sequence of functions. We define the *support* of $\mathcal{H}$ as $\text{supp}(\mathcal{H}) = \bigcup_{h \in \mathcal{H}} \text{supp}(h)$. The total number of non-zero entries in all of the functions of $\mathcal{H}$ is denoted by $\|\mathcal{H}\| = \sum_{h \in \mathcal{H}} |h|$. The *diameter* of a function f is $\mathsf{diam}(f) = \max(\text{supp}(f)) - \min(\text{supp}(f)) + 1$ if $\text{supp}(f) \neq \emptyset$, and $\mathsf{diam}(f) = 0$ otherwise. We define the *diameter* of a sequence of functions $\mathcal{H}$ as $\mathsf{diam}(\mathcal{H}) = \max\{\mathsf{diam}(h) \mid h \in \mathcal{H}\}$.

In our setting, the input for the convolution summation problem is two sequences of *sparse functions*, which are functions that have a small support. Thus, we assume that the input functions are given in an efficient *sparse representation*, e.g., a linked list (of the non-zero functions) of linked lists (of non-zero entries).

Algorithm for the offline convolution summation problem. The following lemma, proved in the full version of the paper, provides an algorithm that efficiently computes $\mathcal{F} \otimes \mathcal{G}$ for two sequences of functions $\mathcal{F}$ and $\mathcal{G}$ which are given in a sparse representation. Notice that the output of the algorithm is also restricted to the non-zero values of $\mathcal{F} \otimes \mathcal{G}$ only.

▶ **Lemma 7** (Based on [4, Lemma 7.5]). *Let $\mathcal{F} = (f_1, \ldots, f_t)$ and $\mathcal{G} = (g_1, \ldots, g_t)$ be two sequences of functions, such that $\mathsf{diam}(\mathcal{F}), \mathsf{diam}(\mathcal{G}) \in [1\,..\,n]$, and also $\mathsf{diam}(\mathcal{F} \otimes \mathcal{G}) = O(n)$. Then there exists an (offline) algorithm that computes the non-zero entries of $\mathcal{F} \otimes \mathcal{G}$ using $O(n)$ space, whose time cost is*

$$\Psi(\mathcal{F},\mathcal{G}) = \tilde{O}\left(\|\mathcal{F}\|+\|\mathcal{G}\|+\sum_{j=1}^{t}\min(|f_j||g_j|,n)\right) = \tilde{O}\left(\min\left(tn, \|\mathcal{F}\|\cdot\|\mathcal{G}\|, (\|\mathcal{F}\|+\|\mathcal{G}\|)\sqrt{n}\right)\right).$$

4 Periodic Pattern and Text – with Delay

Our approach is based on the reduction to the convolution summation problem of Section 3. The text arrives online, so we consider a similar setting for convolution summation.

4.1 The Incremental Batched Convolution Summation Problem

In the incremental batched version of the convolution summation problem, the algorithm is given two sequences of t functions $\mathcal{F}$ and $\mathcal{G}$, where both $\mathrm{supp}(\mathcal{F}), \mathrm{supp}(\mathcal{G}) \subseteq [0\,..\,n-1]$. The sparse representation of $\mathcal{G}$ is available for preprocessing, whereas $\mathcal{F}$ is revealed online in batches of diameter s: the ith batch consists of all of the non-zero entries of the functions of $\mathcal{F}$ in the range $[(i-1)\cdot s\,..\,i\cdot s)$, also in a sparse representation. After each update, the goal is to compute the values of $\mathcal{F} \otimes \mathcal{G}$ in the same range as the input, $[(i-1)\cdot s\,..\,i\cdot s)$. In the rest of this section, we prove the following lemma.

▶ **Lemma 8.** *There exists a deterministic algorithm that solves the incremental batched convolution summation problem for $s = \Omega(\|\mathcal{F}\| + \|\mathcal{G}\|)$, using $O(s)$ space, $\tilde{O}((\|\mathcal{F}\| + \|\mathcal{G}\|)\sqrt{s})$ time per batch arrival and $\tilde{O}\left(n + \min\left(\|\mathcal{F}\|\cdot\|\mathcal{G}\|, \frac{n(\|\mathcal{F}\|+\|\mathcal{G}\|)}{\sqrt{s}}, \frac{tn^2}{s}\right)\right)$ total time.*

A natural approach for proving Lemma 8 is to utilize the algorithm of Lemma 7, whose runtime depends on the diameters of a pair of sequences of functions. Thus, in order to use this approach, we design a mechanism for reducing the diameters of $\mathcal{F}$ and $\mathcal{G}$ while still being able to properly compute the values of $\mathcal{F} \otimes \mathcal{G}$.

Reducing the diameter. For a function $h : \mathbb{Z} \to \mathbb{Z}$ and a domain $D \subseteq \mathbb{Z}$, let $h|_D : \mathbb{Z} \to \mathbb{Z}$ be a function where $h|_D(i) = h(i)$ for $i \in D$ and $h|_D(i) = 0$ for $i \notin D$. For a sequence of functions $\mathcal{H} = (h_1, h_2, \ldots, h_t)$, denote the sequence of functions restricted to domain D as $\mathcal{H}|_D = (h_1|_D, h_2|_D, \ldots, h_t|_D)$. For $a, b \in \mathbb{Z}$, let us define integer intervals $\Gamma_a = [s\cdot(a-1)\,..\,s\cdot(a+1))$ and $\Phi_b = [s\cdot(b-1)\,..\,s\cdot b)$[3].

▶ **Observation 9.** *Each integer is in exactly one range Φ_b and in exactly two ranges Γ_a.*

Notice that the ith batch consists of $\mathcal{F}|_{\Phi_i}$. In response to the ith batch, the algorithm needs to compute $[\mathcal{F} \otimes \mathcal{G}]|_{\Phi_i}$. For this, we express $\mathcal{F}_i = \mathcal{F}|_{[0\,..\,si)} = \mathcal{F}|_{\Phi_1 \cup \Phi_2 \cup \cdots \cup \Phi_i}$ (which is the aggregate of all the batches received so far) and $\mathcal{G}$ in terms of two sequences of functions $\mathcal{F}_i^*$ and $\mathcal{G}^*$, so that $[\mathcal{F} \otimes \mathcal{G}]|_{\Phi_i} = [\mathcal{F}_i^* \otimes \mathcal{G}^*]|_{\Phi_i}$. The motivation for using $\mathcal{F}_i^*$ and $\mathcal{G}^*$ is to reduce the diameter, which comes at the price of increasing the number of functions.

Let $q = \lceil \frac{n}{s} \rceil$, and define

$$\mathcal{F}_i^* = \left(f_1|_{\Phi_{i-0}}, \ldots, f_t|_{\Phi_{i-0}}, f_1|_{\Phi_{i-1}}, \ldots, f_t|_{\Phi_{i-1}}, \ldots\ldots\ldots, f_1|_{\Phi_{i-(q-1)}}, \ldots, f_t|_{\Phi_{i-(q-1)}}\right),$$
$$\mathcal{G}^* = \left(g_1|_{\Gamma_0}\ , \ldots, g_t|_{\Gamma_0}\ , g_1|_{\Gamma_1}\ , \ldots, g_t|_{\Gamma_1}\ , \ldots\ldots\ldots, g_1|_{\Gamma_{q-1}}\ , \ldots, g_t|_{\Gamma_{q-1}}\ \right).$$

[3] Note that the Γ intervals are used for partitioning $\mathcal{G}$ while Φ intervals are used for partitioning $\mathcal{F}$.

▶ **Lemma 10.** *For any $i \in [1\,..\,q]$ we have $[\mathcal{F} \otimes \mathcal{G}]|_{\Phi_i} = [\mathcal{F}^*_i \otimes \mathcal{G}^*]|_{\Phi_i}$.*

Proof. For two sets $X, Y \subseteq \mathbb{Z}$, we denote $X - Y = \{x - y \mid x \in X \text{ and } y \in Y\}$. For any pair of functions $f, g : \mathbb{Z} \to \mathbb{Z}$ and any $j \in \Phi_i$, since the ranges Φ_b for $b \in \mathbb{Z}$ form a partition of $\mathbb{Z}$, we have that $(f * g)(j) = \sum_{a\in\mathbb{Z}}(f|_{\Phi_{i-a}} * g)(j)$. Moreover, by the definition of the convolution operator and since $\Phi_i - \Phi_{i-a} \subseteq \Gamma_a$, we have $\sum_{a\in\mathbb{Z}}(f|_{\Phi_{i-a}} * g)(j) = \sum_{a\in\mathbb{Z}}(f|_{\Phi_{i-a}} * g|_{\Gamma_a})(j)$. Hence, $[\mathcal{F}\otimes\mathcal{G}]|_{\Phi_i} = \sum_{a\in\mathbb{Z}}[\mathcal{F}|_{\Phi_{i-a}} \otimes \mathcal{G}|_{\Gamma_a}]|_{\Phi_i}$. However, since $\mathrm{supp}(\mathcal{F}) \subseteq [0\,..\,qs)$, we have that for every $b \le 0$ and every $f \in \mathcal{F}$, it must be that $f|_{\Phi_b} = 0$. Similarly, since $\mathrm{supp}(\mathcal{G}) \subseteq [0\,..\,qs)$, we have that for every $a < 0$ and every $g \in \mathcal{G}$, it must be that $g|_{\Gamma_a} = 0$. Thus, for $a \notin [0\,..\,q-1]$, we have that $\mathcal{F}|_{\Phi_{i-a}} \otimes \mathcal{G}|_{\Gamma_a} = 0$, and so $[\mathcal{F} \otimes \mathcal{G}]|_{\Phi_i} = \sum_{a=0}^{q-1}[\mathcal{F}|_{\Phi_{i-a}} \otimes \mathcal{G}|_{\Gamma_a}]|_{\Phi_i}$. Finally, since $\mathcal{F}^*_i$ is the concatenation of $\mathcal{F}|_{\Phi_{i-a}}$ for $a \in [0\,..\,q-1]$, and $\mathcal{G}^*$ is the concatenation of $\mathcal{G}|_{\Gamma_a}$ for $a \in [0\,..\,q-1]$, the lemma follows. ◀

The following is a consequence of the definitions of $\mathcal{F}^*$ and $\mathcal{G}^*$, and Observation 9. Recall that the diameter of a sequence of functions $\mathcal{H}$ is $\mathsf{diam}(\mathcal{H}) = \max\{\mathsf{diam}(h) \mid h \in \mathcal{H}\}$.

▶ **Observation 11.** *The sequences $\mathcal{F}^*_i$ and $\mathcal{G}^*$ consist of $t \cdot q$ functions each. Moreover, $\mathsf{diam}(\mathcal{F}^*_i) \le s$, $\mathsf{diam}(\mathcal{G}^*) \le 2s$, $\|\mathcal{F}^*_i\| \le \|\mathcal{F}\|$, and $\|\mathcal{G}^*\| \le 2 \cdot \|\mathcal{G}\|$.*

The algorithm. During the preprocessing phase, the algorithm transforms $\mathcal{G}$ into $\mathcal{G}^*$, and constructs $\mathcal{F}^*_0$, which is an empty linked list.

Upon receiving the ith batch, which is $\mathcal{F}|_{\Phi_i}$, the algorithm computes $\mathcal{F}^*_i$ as follows: First, the algorithm concatenates $\mathcal{F}|_{\Phi_i}$ with $\mathcal{F}^*_{i-1}$ and truncates the last t functions from the resulting sequence of $(q+1)t$ functions. It is easy to implement the concatenation (including updating indices in the sparse representation) with a time cost which is linear in the number of non-zero functions in $\mathcal{F}|_{\Phi_i}$ and $\mathcal{F}^*_{i-1}$, and is at most $O(\|\mathcal{F}\|)$. The implementation of the truncation is void since only the first $(i-1)\cdot t \le (q-1)\cdot t$ functions of $\mathcal{F}^*_{i-1}$ are non-zero[4].

Next, the algorithm applies the procedure of Lemma 7 in order to compute $\mathcal{F}^*_i \otimes \mathcal{G}^*$. Finally, the algorithm returns $[\mathcal{F}^*_i \otimes \mathcal{G}^*]|_{\Phi_i}$ which, by Lemma 10, is $[\mathcal{F} \otimes \mathcal{G}]|_{\Phi_i}$.

Complexities. The preprocessing phase is done in $O(\|\mathcal{G}\|)$ time using $O(s)$ space.

For the ith batch, the computation of $\mathcal{F}^*_i$ costs $O(\|\mathcal{F}\|)$ time, which is $O(q\|\mathcal{F}\|) = O(\frac{n}{s}\|\mathcal{F}\|) = O(n)$ time in total because $s = \Omega(\|\mathcal{F}\| + \|\mathcal{G}\|)$. Then, the computation of $\mathcal{F}^*_i \otimes \mathcal{G}^*$ with the procedure of Lemma 7 costs $O(\Psi(\mathcal{F}^*_i, \mathcal{G}^*)) = \tilde{O}((\|\mathcal{F}^*_i\| + \|\mathcal{G}^*\|)\sqrt{s}) = \tilde{O}((\|\mathcal{F}\| + \|\mathcal{G}\|)\sqrt{s})$ time. In the following, we derive several upper bounds on $\sum_{i=1}^{q}\Psi(\mathcal{F}^*_i, \mathcal{G}^*)$, which together upper bound the total time cost.

Total time $\tilde{O}\left(\frac{n(\|\mathcal{F}\|+\|\mathcal{G}\|)}{\sqrt{s}}\right)$. Recall that $q = O(\frac{n}{s})$, and, by Observation 11, for any $i \in [1\,..\,q]$ we have $\|\mathcal{F}^*_i\| \le \|\mathcal{F}\|$ and $\|\mathcal{G}^*\| \le 2\|\mathcal{G}\|$. Thus,

$$\sum_{i=1}^{q}\Psi(\mathcal{F}^*_i, \mathcal{G}^*) = \sum_{i=1}^{q}\tilde{O}\left((\|\mathcal{F}^*_i\| + \|\mathcal{G}^*\|)\sqrt{s}\right) = \tilde{O}\left(\sum_{i=1}^{q}(\|\mathcal{F}\| + \|\mathcal{G}\|)\sqrt{s}\right) =$$
$$\tilde{O}\left(q\cdot(\|\mathcal{F}\| + \|\mathcal{G}\|)\sqrt{s}\right) = \tilde{O}(\tfrac{n}{s}\cdot(\|\mathcal{F}\| + \|\mathcal{G}\|)\sqrt{s}) = \tilde{O}\left(\tfrac{n(\|\mathcal{F}\|+\|\mathcal{G}\|)}{\sqrt{s}}\right).$$

[4] The reason for mentioning the truncation, even though it is void, is in order to guarantee that $\mathcal{F}^*_i$ matches the mathematical definition introduced above.

Total time $\tilde{O}\left(\frac{tn^2}{s}\right)$. Recall that for any $i \in [1\,..\,q]$ the sequences $\mathcal{G}^*$ and $\mathcal{F}_i^*$ consist of $q \cdot t$ functions of diameter $O(s)$, and $q = O(\frac{n}{s})$. Thus,

$$\sum_{i=1}^{q} \Psi(\mathcal{F}_i^*, \mathcal{G}^*) = \sum_{i=1}^{q} \tilde{O}\left(q \cdot t \cdot s\right) = \tilde{O}\left(q^2 \cdot t \cdot s\right) = \tilde{O}\left(\frac{n^2}{s^2} \cdot t \cdot s\right) = \tilde{O}\left(\frac{tn^2}{s}\right)$$

Total time $\tilde{O}(n + \|\mathcal{F}\| \cdot \|\mathcal{G}\|)$. Let $\mathcal{X} = \{x_1, x_2, \ldots, x_\tau\}$ and $\mathcal{Y} = \{y_1, y_2, \ldots, y_\tau\}$ be two sequences of τ functions from $\mathbb{Z}$ to $\mathbb{Z}$, and let $\nu = \max(\mathsf{diam}(\mathcal{X}), \mathsf{diam}(\mathcal{Y}))$. Recall that

$$\Psi(\mathcal{X}, \mathcal{Y}) = \tilde{O}\left(\|\mathcal{X}\| + \|\mathcal{Y}\| + \sum_{j=1}^{\tau} \min(|x_j||y_j|, \nu)\right) = \tilde{O}\left(\|\mathcal{X}\| + \|\mathcal{Y}\| + \sum_{j=1}^{\tau} |x_j||y_j|\right).$$

In our case, the run time is $\sum_{i=1}^{q} \Psi(\mathcal{F}_i^*, \mathcal{G}^*)$. Recall that the functions in $\mathcal{F}_i^*$ are $f_j|_{\Phi_{i-a}}$, and the functions in $\mathcal{G}^*$ are $g_j|_{\Gamma_a}$, both for $a \in [0\,..\,q)$ and $j \in [1\,..\,t]$. Thus,

$$\begin{aligned}
\sum_{i=1}^{q} \Psi(\mathcal{F}_i^*, \mathcal{G}^*) &= \sum_{i=1}^{q} \tilde{O}\left(\|\mathcal{F}_i^*\| + \|\mathcal{G}^*\| + \sum_{a=0}^{q-1}\sum_{j=1}^{t} \left|f_j|_{\Phi_{i-a}}\right|\left|g_j|_{\Gamma_a}\right|\right) \\
&= \sum_{i=1}^{q} \tilde{O}\left(\|\mathcal{F}\| + \|\mathcal{G}\|\right) + \tilde{O}\left(\sum_{j=1}^{t}\sum_{a=0}^{q-1} \left|g_j|_{\Gamma_a}\right| \sum_{i=1}^{q} \left|f_j|_{\Phi_{i-a}}\right|\right) && \triangleright \text{ by Obs. 11} \\
&= \tilde{O}(\tfrac{n}{s}s) + \tilde{O}\left(\sum_{j=1}^{t}\sum_{a=0}^{q-1} \left|g_j|_{\Gamma_a}\right|\left|f_j\right|\right) && \triangleright \text{ by Obs. 9} \\
&= \tilde{O}(n) + \tilde{O}\left(\sum_{j=1}^{t} \left|g_j\right|\left|f_j\right|\right) && \triangleright \text{ by Obs. 9} \\
&= \tilde{O}(n) + \tilde{O}\left(\|\mathcal{G}\| \sum_{j=1}^{t} \left|f_j\right|\right) = \tilde{O}(n) + \tilde{O}\left(\|\mathcal{G}\| \cdot \|\mathcal{F}\|\right) = \tilde{O}(n + \|\mathcal{F}\| \cdot \|\mathcal{G}\|)
\end{aligned}$$

This completes the proof of Lemma 8. ◀

4.2 Reduction from Hamming Distance to Incremental Batch Sparse Convolution Summation Problem

Now we show how to compute the Hamming distance between P and substrings of T with delay of $2s$, based on the algorithm of Lemma 8. Our result is stated in the following lemma.

▶ **Lemma 12.** *Suppose that there exists $\rho \le k$ which is a d-period of both P and T for some $d = O(k)$. Then, there exists a deterministic streaming algorithm for the k-mismatch problem with $n = \frac{3}{2}m$ that, given an integer parameter $k \le s \le m$, uses $\tilde{O}(s)$ space and costs $\tilde{O}\left(m + \min\left(k^2, \frac{mk}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right)$ total time. Moreover, the worst-case time cost per character is $\tilde{O}(\sqrt{k})$. The algorithm has delay of $2s$ characters.*

Proof. The algorithm receives the characters of T one by one and splits them into blocks of length s so that the indices in the bth block form Φ_b, as defined in Section 4.1. The last $O(s)$ characters of T and the last $O(s)$ values of $T \otimes P$ are buffered in $O(s)$ space.

For $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$, define sequences of functions $\mathcal{G} = (\Delta_\rho[P^R_{c_1}], \Delta_\rho[P^R_{c_2}], \ldots, \Delta_\rho[P^R_{c_\sigma}])$ and $\mathcal{F} = (\Delta_\rho[T_{c_1}], \Delta_\rho[T_{c_2}], \ldots, \Delta_\rho[T_{c_\sigma}])$. The algorithm initializes an instance of the procedure of Lemma 8 with $\mathcal{G}$ during the arrival of the first block. During the arrival of the bth block, the algorithm creates the batch $\mathcal{F}|_{\Phi_b}$ as follows: after the arrival of $T[i]$, if $T[i] \neq T[i-\rho]$, then the algorithm sets $\Delta_\rho[T_{T[i]}](i)$ to be 1 and $\Delta_\rho[T_{T[i-\rho]}](i)$ to be -1. During the arrival of the $(b+1)$th block, the batch $\mathcal{F}|_{\Phi_b}$ is processed using Lemma 8 in order to compute $[\mathcal{F} \otimes \mathcal{G}]|_{\Phi_b}$. Finally, for all $i \in \Phi_b$, the algorithm uses the buffer of $T \otimes P$ together with the values of $[\mathcal{F} \otimes G]|_{\Phi_b}$ in order to report the values $[T \otimes P](i) = [\mathcal{F} \otimes \mathcal{G}](i) - [T \otimes P](i-2\rho) + 2[T \otimes P](i-\rho)$ (see Lemma 6).

Complexities. For each incoming character of T, updating the text buffer and the functions $\Delta_\rho[T_c]$ for all $c \in \Sigma$ costs $O(1)$ time since changes are needed in $\Delta_\rho[T_c]$ for at most two characters c. Since $\rho \leq k$ is a d-period of both P and T, Observation 5 yields $\|\mathcal{G}\| \leq 2(k+d) = O(k)$ and $\|\mathcal{F}\| \leq 2(k+d) = O(k)$. Thus, the initialization of the procedure of Lemma 8 costs $O(\|\mathcal{G}\|) = O(k)$ time, and executing the procedure of Lemma 8 on the bth batch $\mathcal{F}|_{\Phi_b}$ costs $\tilde{O}((\|\mathcal{F}\| + \|\mathcal{G}\|)\sqrt{s}) = \tilde{O}(k\sqrt{s})$ time. Hence, applying the algorithm of Lemma 8 during the arrival of *any* block costs $\tilde{O}(k + k\sqrt{s}) = \tilde{O}(k\sqrt{s})$ time, which, by a standard de-amortization, is $\tilde{O}\big(\frac{1}{s} \cdot k\sqrt{s}\big) = \tilde{O}\big(\frac{k}{\sqrt{s}}\big) = \tilde{O}(\sqrt{k})$ time per character.

Note that $\operatorname{supp}(\mathcal{G}) \subseteq [0\,..\,m+\rho] = [0\,..\,O(m)]$ and $\operatorname{supp}(\mathcal{F}) \subseteq [0\,..\,\frac{3}{2}m + \rho] = [0\,..\,O(m)]$. Moreover, both $\mathcal{F}$ and $\mathcal{G}$ are sequences of σ functions. Hence, the total time cost of applying the algorithm of Lemma 8, updating the buffers, and computing the functions $\Delta_\rho[T_c]$ is $\tilde{O}\left(m + \min\left(\|\mathcal{F}\| \cdot \|\mathcal{G}\|, \frac{m(\|\mathcal{F}\|+\|\mathcal{G}\|)}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right) = \tilde{O}\left(m + \min\left(k^2, \frac{mk}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right)$.

Delay. For any index $i \in \Phi_b$, after the arrival of $T[i]$, at most s character arrivals take place until the call to the procedure of Lemma 8 involving $\mathcal{F}|_{\Phi_b}$. Moreover, due to the de-amortization, the computation of all the results $[T \otimes P]|_{\Phi_b}$ takes place during the arrivals of another s characters. Hence, the delay of the algorithm is at most $2s$ character arrivals. ◀

5 Periodic Pattern and Text – without Delay

In this section, we show how to compute the distances between P and substrings of T without any delay, assuming that $\rho \leq k$ is a d-period of both P and T for some $d = O(k)$. Our approach is to use the tail partition technique, described in Section 2. To do so, we set $|P_{tail}| = 2s$ and use the algorithm of Lemma 12 on P_{head} (notice that ρ is a d-period of both P_{head} and P_{tail}). The remaining task is to describe how to compute $\mathsf{Ham}(P_{tail}, T[i-2s+1\,..\,i])$.

In the online version of the convolution summation problem, the algorithm is given two sequences of functions $\mathcal{F}$ and $\mathcal{G}$, where $\operatorname{supp}(\mathcal{F}) \subseteq [0\,..\,n)]$ and $\operatorname{supp}(\mathcal{G}) \subseteq [0\,..\,m)$. The sparse representation of $\mathcal{G}$ is available for preprocessing, whereas $\mathcal{F}$ is revealed online index by index: At the ith update, the algorithm receives $\mathcal{F}|_{\{i\}}$, i.e., all the non-zero entries $f(i)$ for $f \in \mathcal{F}$, and the task is to compute $[\mathcal{F} \otimes \mathcal{G}](i)$. The procedure we use for this problem is based on the algorithm of Clifford et al. [5]. Nevertheless, since we state the procedure in terms of the convolution summation problem, whereas [5] states the algorithm in terms of pattern matching problems, we describe the details in the full version.

▶ **Lemma 13** (Based on [5, Theorem 1]). *Let $\mathcal{F}$ and $\mathcal{G}$ be two sequences of t functions each such that $\operatorname{supp}(\mathcal{F}) \subseteq [0\,..\,n)$, $\operatorname{supp}(\mathcal{G}) \subseteq [0\,..\,m)$, and $\delta = \max(\max_i \|\mathcal{F}|_{\{i\}}\|, \max_i \|\mathcal{G}|_{\{i\}}\|)$ is the maximum number of non-zero entries at a single index. There exists an online algorithm that upon receiving $\mathcal{F}|_{\{i\}}$ for subsequent indices i computes $[\mathcal{F} \otimes \mathcal{G}](i)$ using $O(\delta m)$ space in $\tilde{O}\big(\sqrt{\delta(\|\mathcal{F}\| + \|\mathcal{G}\|)}\big)$ time per index. Moreover, the total running time of the algorithm is $\tilde{O}(\min(n\delta + (\|\mathcal{F}\| + \|\mathcal{G}\|)\sqrt{n}, nt))$.*

Using the algorithm of Lemma 13 and the reduction from computing Hamming distance to the convolution summation problem defined in Section 3, we achieve the following lemma.

▶ **Lemma 14.** *Suppose that there exists $\rho \leq k$ which is a d-period of both P and T for some $d = O(k)$. Then, there exists a deterministic online algorithm for the k-mismatch problem that uses $\tilde{O}(m)$ space and costs $\tilde{O}\left(n + \min\left(k\sqrt{n}, n\sigma\right)\right)$ total time. Moreover, the worst-case time cost per character is $\tilde{O}(\sqrt{k})$.*

Proof. The algorithm maintains a buffer of the last 2ρ values of $T \otimes P$ and a buffer of the last ρ characters of T. Define sequences of functions $\mathcal{G} = (\Delta_\rho[P^R_{c_1}], \Delta_\rho[P^R_{c_2}], \ldots, \Delta_\rho[P^R_{c_\sigma}])$ and $\mathcal{F} = (\Delta_\rho[T_{c_1}], \Delta_\rho[T_{c_2}], \ldots, \Delta_\rho[T_{c_\sigma}])$, where $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$. The algorithm initializes an instance of the procedure of Lemma 13. After the arrival of $T[i]$, if $T[i] \neq T[i-\rho]$, then the algorithm sets $\Delta_\rho[T_{T[i]}](i)$ to be 1 and $\Delta_\rho[T_{T[i-\rho]}](i)$ to be -1. The algorithm transfers these values to the procedure of Lemma 13, which responds with the value of $[\mathcal{F} \otimes \mathcal{G}](i)$. Then, the algorithm reports $[T \otimes P](i)$, which equals $[\mathcal{F} \otimes \mathcal{G}](i) - [T \otimes P](i - 2\rho) + 2[T \otimes P](i - \rho)$ by Lemma 6. In this expression, the first term is returned by the procedure of Lemma 13 and the other two terms are retrieved from the buffer.

The update of the text buffer and $\Delta_\rho[T_c]$ for at most two characters c after the arrival of any text character costs $O(1)$ time per character and $O(n)$ time in total. Note that by Observation 5, since $\rho \leq k$ is a d-period of both P and T, we have that $\|\mathcal{G}\| \leq 2(d + k) = O(k)$ and $\|\mathcal{F}\| \leq 2(d + k) = O(k)$. Moreover, $\delta = \max(\max_i \|\mathcal{F}|_{\{i\}}\|, \max_i \|\mathcal{G}|_{\{i\}}\|) \leq 2$. Consequently, the algorithm of Lemma 13 uses $\tilde{O}(m)$ space and costs $\tilde{O}(\sqrt{k})$ time per character and $\tilde{O}(\min(n + k\sqrt{n}, n\sigma))$ time in total. All the other parts of the algorithm cost $O(1)$ time per character and $O(n)$ time in total. ◀

Thus, by combining Lemma 12 and Lemma 14, we obtain an algorithm that, given ρ which is is a d-period of both P and T, computes the Hamming distances up to k for every substring of the text without delay.

▶ **Lemma 15.** *Suppose that there exists $\rho \leq k$ which is a d-period of both P and T for some $d = O(k)$. Then, there exists a deterministic streaming algorithm for the k-mismatch problem with $n = \frac{3}{2}m$ that, given an integer parameter $k \leq s \leq m$, uses $\tilde{O}(s)$ space, and costs $\tilde{O}\left(m + \min\left(k^2, \frac{mk}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right)$ total time and $\tilde{O}(\sqrt{k})$ time per character in the worst case.*

Proof. Let P_{tail} be the suffix of P of length $2s$, and let P_{head} be the complementary prefix of P. Note that ρ is a d-period of both P_{head} and P_{tail}. We run the algorithm of Lemma 12 with P_{head} as a pattern. The results of the algorithm of Lemma 12 are added into a buffer of length $2s$. Thus, right after the arrival of $T[i]$, it is guaranteed that $\mathsf{Ham}(P_{head}, T[i - |P| + 1 \,..\, i - |P_{tail}|])$ is already computed and available for the algorithm. In addition, the algorithm of Lemma 14 is executed with P_{tail} as pattern, and this algorithm computes $\mathsf{Ham}(P_{tail}, T[i - |P_{tail}| + 1 \,..\, i])$ right after the arrival of $T[i]$, if this value is at most k. After the arrival of $T[i]$, the algorithm reports $\mathsf{Ham}(P, T[i - |P| + 1 \,..\, i]) = \mathsf{Ham}(P_{head}, T[i - |P| + 1 \,..\, i - |P_{tail}|]) + \mathsf{Ham}(P_{tail}, T[i - |P_{tail}| + 1 \,..\, i])$.

The time per character of both the algorithm of Lemma 12 and the algorithm of Lemma 14 is $\tilde{O}(\sqrt{k})$. The total time cost of the algorithm of Lemma 12 is $\tilde{O}\left(m + \min\left(k^2, \frac{mk}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right)$. Furthermore, the total time cost of the algorithm of Lemma 14 is $\tilde{O}\left(\min\left(m + k\sqrt{m}, m\sigma\right)\right) = \tilde{O}\left(m + \min\left(k^2, \frac{km}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right)$ due to $k \leq s \leq m$ and $m + k\sqrt{m} = O(m + k^2)$. Hence, the lemma follows. ◀

6 Periodic Pattern and Arbitrary Text – without Delay

In this section, we generalize Lemma 15 to the case where ρ is not necessarily a d-period of T, but ρ is still a d-period of P and $n = \frac{3}{2}m$. We use ideas building upon Clifford et al. [7, Lemma 6.2] to show that there exists a substring of T, denoted T^*, such that ρ is a $(2d + 4k + \rho)$-period of T^*, and T^* contains all of the k-mismatch occurrences of P in T. Our construction, specified below, exploits the following property of approximate periods.

▶ **Observation 16.** *Let X and Y be two equal length strings. If ρ is a d-period of X and $\mathsf{Ham}(X, Y) \le x$ then ρ is a $(d + 2x)$-period of Y.*

Let T_L be the longest suffix of $T[0\,..\,\frac{1}{2}m - 1]$ such that ρ is a $(d + 2k)$-period of T_L, and let T_R be the longest prefix of $T[\frac{1}{2}m\,..\,n - 1]$ such that ρ is a $(d + 2k)$-period of T_R. Finally, let T^* be the concatenation $T^* = T_L \cdot T_R$.

▶ **Lemma 17.** *All the k-mismatch occurrences of P in T are contained within T^*. Moreover, ρ is a $(2d + 4k + \rho)$-period of T^*.*

Proof. The second claim follows directly from the fact that $T^* = T_L \cdot T_R$ is a concatenation of two strings with $(d + 2k)$-period ρ (the extra ρ mismatches might occur at the boundary between T_L and T_R). Henceforth, we focus on the first claim.

We assume that P has at least one k-mismatch occurrence in T; otherwise, the claim holds trivially. Let $T[\ell\,..\,r]$ be the smallest fragment of T containing all the k-mismatch occurrences of P in T (so that the leftmost and the rightmost occurrences starts at positions ℓ and $r - m + 1$, respectively). Our goal is to prove that $T[\ell\,..\,r]$ is contained within T^*.

By Observation 16, since $T[\ell\,..\,\ell + m - 1]$ is a k-mismatch occurrence of P and ρ is a d-period of P, it must be that ρ is a $(d + 2k)$-period of $T[\ell\,..\,\ell + m - 1]$. In particular, since $\ell + m \ge \frac{1}{2}m$, we have that ρ is a $(d + 2k)$-period of $T[\ell\,..\,\frac{1}{2}m - 1]$. Hence, by its maximality, T_L must start at position ℓ or to the left of ℓ. Similarly, by Observation 16, since $T[r - m + 1\,..\,r]$ is a k-mismatch occurrence of P and ρ is an d-period of P, it must be that ρ is a $(d + 2k)$-period of $T[r - m + 1\,..\,r]$. In particular, since $r - m + 1 \le n - m \le \frac{1}{2}m$, we have that ρ is a $(d + 2k)$-period of $T[\frac{1}{2}m\,..\,r]$. Hence, by its maximality, T_R must end at position r or to the right of r. ◀

The algorithm works in two high-level phases. In the first phase, the algorithm receives $T[0\,..\,\frac{1}{2}m - 1]$, and the goal is to compute T_L. In the second phase, the algorithm receives $T[\frac{1}{2}m\,..\,n - 1]$ and transfers $T^* = T_L \cdot T_R$ to the subroutine of Lemma 12. The transfer starts with a delay of $|T_L|$ characters and a standard de-amortization speedup is applied to reduce the delay to 0 by the time $2|T_L| \le m$ characters are transferred, which is before the subroutine of Lemma 12 may start producing output. The algorithm terminates as soon as it reaches the end of T_R, i.e., when it encounters more than $d + 2k$ mismatches in $T[\frac{1}{2}m\,..\,n - 1]$.

The following *periodic representation* (similar to one by Clifford et al. [8]) is used for storing substrings of T.

▶ **Fact 18.** *For every positive integer ρ, there exists an algorithm that maintains a representation of $T[\ell\,..\,r]$ and supports the following operations in $O(1)$ time each:*

(a) *Change the representation to represent $T[\ell + 1\,..\,r]$ and return $T[\ell]$.*

(b) *Given $T[r + 1]$ and $T[r + 1 - \rho]$, change the representation to represent $T[\ell\,..\,r + 1]$.*

(c) *Given $\ell' \ge \ell$ such that $T[i] = T[i + \rho]$ for $\ell \le i < \ell'$, change the representation to represent $T[\ell'\,..\,r]$.*

If ρ is a d-period of $T[\ell\,..\,r]$ then the space usage is $O(d + \rho)$.

Proof. $T[\ell\,..\,r]$ is represented by a string S of length ρ such that $T[i] = S[i \bmod \rho]$ for $\ell \le i \le \min(\ell+\rho-1, r)$, and a list $\mathsf{L} = \{(i, T[i]) : \ell+\rho \le i \le r, T[i-\rho] \neq T[i]\}$. Notice that if ρ is a d-period of $T[\ell\,..\,r]$, then this representation uses $O(d+\rho)$ space.

To implement operation (a), the algorithm first retrieves $T[\ell] = S[\ell \bmod \rho]$. The algorithm then checks if the leading element of L is $(\ell+\rho, T[\ell+\rho])$. If so, the algorithm removes this pair from L and sets $S[\ell \bmod \rho] = T[\ell+\rho]$. To implement operation (b), the algorithm compares $T[r+1]$ with $T[r+1-\rho]$. If these values are different, then $(r+1, T[r+1])$ is appended to L. The implementation of operation (c) is trivial (neither S nor L is changed). ◀

▶ **Lemma 19.** *Suppose that there exists $\rho \le k$ which is a d-period of P for some $d = O(k)$. Then, there exists a deterministic streaming algorithm for the k-mismatch problem with $n = \frac{3}{2}m$ that, given an integer parameter $k \le s \le m$, uses $\tilde{O}(s)$ space, and costs $\tilde{O}\left(m + \min\left(k^2, \frac{mk}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right)$ total time and $\tilde{O}(\sqrt{k})$ time per character in the worst case.*

Proof. Based on the pattern P and the period ρ, the algorithm initializes an instance ALG of the algorithm of Lemma 12. Then, the algorithm processes T in two phases, while maintaining a buffer of ρ text characters and the representation of Fact 18 of a suffix T' of the already processed prefix of T.

First Phase. During the first phase, when the algorithm receives $T[0\,..\,\frac{1}{2}m-1]$, the suffix T' is defined as the longest suffix for which ρ is a $(d+2k)$-period. Suppose T' is $T[\ell\,..\,i-1]$ after processing $T[0\,..\,i-1]$. The algorithm first appends $T[i]$ to T', extending it to $T[\ell\,..\,i]$ using Fact 18(b). If ρ is still a $(d+2k)$-period of T', i.e., the list L has at most $d+2k$ elements, then the algorithm proceeds to the next character. Otherwise, T' is first trimmed to $T[\ell'\,..\,i]$, where $(\ell'+\rho, T[\ell'+\rho])$ is the first element of L, using Fact 18(c), and then to $T[\ell'+1\,..\,i]$ using Fact 18(a). The latter operation decrements the size of L to $d+2k$.

At the end of the first phase, T' is by definition equal to T_L. The running time of the algorithm in the first phase is $O(1)$ per character, and the space complexity is $O(d+k) = O(k)$.

Second Phase. At the second phase, the algorithm receives $T[\frac{m}{2}\,..\,n-1]$, while counting the number of mismatches with respect to ρ. As soon as this number exceeds $d+2k$, which happens immediately after receiving the entire string T_R, the algorithm stops. As long as T' is non-empty, each input character is appended to T' using Fact 18(b), and the two leading characters of T' are popped using Fact 18(a) and transferred to ALG. Once T' becomes empty (which is after ALG receives $2|T_L| \le m$ characters), the input characters are transferred directly to ALG. This process guarantees that the input to ALG is T^* and that ALG is executed with no delay by the time the first m characters of T^* are passed. By Lemma 17, all k-mismatch occurrences of P in T are contained in T^*, so all these occurrences are reported in a timely manner.

Since ρ is a $(2d+4k+\rho)$-period of T^* (by Lemma 17) and T' is contained in T, then ρ is also a $(2d+4k+\rho)$-period of T' at all times. Consequently, the space complexity is $O(k)$ on top of the space usage of ALG, which is $\tilde{O}(s)$. Thus, in total, the algorithm uses $\tilde{O}(s)$ space.

The per-character running time is dominated by the time cost of ALG, which is $\tilde{O}(\sqrt{k})$. The total running time of the algorithm is also dominated by the total running time of ALG, which, by Lemma 12, is $\tilde{O}\left(m + \min\left(k^2, \frac{mk}{\sqrt{s}}, \frac{\sigma m^2}{s}\right)\right)$. ◀

The following corollary is obtained from Lemma 19 by the standard trick of splitting the text into $O(\frac{n}{m})$ substrings of length $\frac{3}{2}m$ with overlaps of length m.

▶ **Corollary 20.** *Suppose that there exists $\rho \leq k$ which is a d-period of P. Then, there exists a deterministic streaming algorithm for the k-mismatch problem that, given an integer parameter $k \leq s \leq m$, uses $\tilde{O}(s)$ space and costs $\tilde{O}\left(n + \min\left(\frac{nk^2}{m}, \frac{nk}{\sqrt{s}}, \frac{\sigma nm}{s}\right)\right)$ total time. Moreover, the worst-case time cost per character is $\tilde{O}(\sqrt{k})$.*

7 Aperiodic Pattern and Arbitrary Text

The following lemma, proved in the full version, appears in [16] with small modifications.

▶ **Lemma 21** (Based on [16, Theorem 5])**.** *Suppose that the smallest $4k$-period of the pattern P is $\Omega(k)$. Then, there exists a randomized streaming algorithm for the k-mismatch problem that uses $\tilde{O}(k)$ space and costs $\tilde{O}(1)$ time per character. The algorithm has delay of k characters and is correct with high probability.*

In order to improve the algorithm to report results without any delay, we use ideas similar to those introduced in Section 5. The following fact appeared in [7].

▶ **Fact 22** (Based on [7, Fact 3.1])**.** *If ρ is the smallest d-period of a pattern P, then the $\frac{1}{2}d$-mismatch occurrences of P in any text T start at least ρ positions apart.*

▶ **Lemma 23.** *Suppose that the smallest $6k$-period of the pattern P is $\Omega(k)$. Then, there exists a randomized streaming algorithm for the k-mismatch problem that uses $\tilde{O}(k)$ space and costs $\tilde{O}(1)$ time per character. The algorithm is correct with high probability.*

Proof. Let P_{tail} be the suffix of P of length $2k$ and let P_{head} be the complementary prefix of P. Since the smallest $6k$-period of P is $\Omega(k)$, $|P_{tail}| = 2k$, and $6k - 2k = 4k$, the smallest $4k$-period of P_{head} is also $\Omega(k)$. Thus, we execute the procedure of Lemma 21 with P_{head}. Then, whenever the procedure reports $\mathsf{Ham}(P_{head}, T[i - |P| + 1 \,..\, i - 2k])$ to be at most k, the algorithm starts a process that computes $\mathsf{Ham}(P_{tail}, T[i - 2k + 1 \,..\, i])$. The procedure of Lemma 21 reports $\mathsf{Ham}(P_{head}, T[i-|P|+1 \,..\, i-2k])$ before $T[i-k+1]$ arrives. Hence, there are still at least k character arrivals until $\mathsf{Ham}(P, T[i-|P|+1 \,..\, i])$ has to be reported. During these character arrivals, the computation of $\mathsf{Ham}(P_{tail}, T[i - 2k + 1 \,..\, i])$ is done simply by comparing pairs of characters. The total time of this computation is $O(k)$, and by standard de-amortization, this is $O(1)$ time per character during the arrival of the k characters.

Since the smallest $4k$-period of P is $\Omega(k)$, Fact 22 implies that any two k-mismatch occurrences of P in T are at distance $\Omega(k)$. Therefore, the maximum number of processes computing distances to P_{tail} at any time is $O(1)$. Thus, the time cost per character of the algorithm is dominated by the procedure of Lemma 21. ◀

8 Proof of Main Theorem

We conclude the paper with a proof of Theorem 3, which is our main result. In the preprocessing, the shortest $6k$-period ρ of the pattern P is determined. If $\rho \leq k$, then the text is processed using Corollary 20. This procedure uses $\tilde{O}(s)$ space and costs $\tilde{O}(\sqrt{k})$ time per character and $\tilde{O}\left(n+\min\left(\frac{nk^2}{m}, \frac{nk}{\sqrt{s}}, \frac{\sigma nm}{s}\right)\right)$ time in total. Otherwise, the text is processed based on the Lemma 23. The space complexity in this case is $\tilde{O}(k) = \tilde{O}(s)$, whereas the running time is $\tilde{O}(1) = \tilde{O}(\sqrt{k})$ per character and $\tilde{O}(n)$ in total.

References

1. Karl R. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987. doi:10.1137/0216067.
2. Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.
3. Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Transactions on Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.
4. Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *52nd Annual ACM Symposium on Theory of Computing, STOC 2020*, 2020. arXiv:2001.00211.
5. Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. *Information and Computation*, 209(4):731–736, 2011. doi:10.1016/j.ic.2010.12.007.
6. Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. Dictionary matching in a stream. In Nikhil Bansal and Irene Finocchi, editors, *23rd Annual European Symposium on Algorithms, ESA 2015*, volume 9294 of *LNCS*, pages 361–372. Springer, 2015. doi:10.1007/978-3-662-48350-3_31.
7. Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.
8. Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In Timothy M. Chan, editor, *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1106–1125. SIAM, 2019. doi:10.1137/1.9781611975482.68.
9. Raphaël Clifford and Benjamin Sach. Pseudo-realtime pattern matching: Closing the gap. In Amihood Amir and Laxmi Parida, editors, *21st Annual Symposium on Combinatorial Pattern Matching, CPM 2010*, volume 6129 of *LNCS*, pages 101–111. Springer, 2010. doi:10.1007/978-3-642-13509-5_10.
10. Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM Journal on Computing*, 31(6):1761–1782, 2002. doi:10.1137/S0097539700370527.
11. Michael J. Fischer and Michael S. Paterson. String matching and other products. In Richard M. Karp, editor, *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 113–125, Providence, RI, 1974. AMS.
12. Zvi Galil and Raffaele Giancarlo. Parallel string matching with k mismatches. *Theoretical Computer Science*, 51:341–348, 1987. doi:10.1016/0304-3975(87)90042-9.
13. Paweł Gawrychowski and Tatiana Starikovskaya. Streaming dictionary matching with mismatches. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM*, volume 128 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.21.
14. Paweł Gawrychowski and Przemysław Uznański. Personal communication, November 2018.
15. Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *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.
16. Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 65:1–65:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

17 Shay Golan and Ely Porat. Real-time streaming multi-pattern search for constant alphabet. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017*, volume 87 of *LIPIcs*, pages 41:1–41:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.41.

18 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7.

19 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.

20 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009*, pages 315–323. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.11.

21 Jakub Radoszewski and Tatiana Starikovskaya. Streaming k-mismatch with error correcting and applications. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference, DCC 2017*, pages 290–299. IEEE, 2017. doi:10.1109/DCC.2017.14.

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

23 Tatiana Starikovskaya, Michal Svagerka, and Przemysław Uznański. L_p pattern matching in a stream, 2019. arXiv:1907.04405.