Periodicity in Data Streams with Wildcards

Funda Ergün[1], Elena Grigorescu[2], Erfan Sadeqi Azer[1], and Samson Zhou[2]

[1] School of Informatics and Computing, Indiana University, Bloomington, IN.
Email: fergun@indiana.edu, esadeqia@indiana.edu.
[2] Department of Computer Science, Purdue University, West Lafayette, IN.
Email: elena-g@purdue.edu, samsonzhou@gmail.com.

Abstract. We investigate the problem of detecting periodic trends within a string S of length n, arriving in the streaming model, containing at most k wildcard characters, where $k = o(n)$. A wildcard character is a special character that can be assigned any other character. We say S has wildcard-period p if there exists an assignment to each of the wildcard characters so that in the resulting stream the length $n - p$ prefix equals the length $n - p$ suffix. We present a two-pass streaming algorithm that computes wildcard-periods of S using $\mathcal{O}\left(k^3 \operatorname{polylog} n\right)$ bits of space, while we also show that this problem cannot be solved in sublinear space in one pass. We then give a one-pass randomized streaming algorithm that computes all wildcard-periods p of S with $p < \frac{n}{2}$ and no wildcard characters appearing in the last p symbols of S, using $\mathcal{O}\left(k^3 \log^9 n\right)$ space.

1 Introduction

We study the problem of detecting repetitive structure in a data stream S containing a small number of *wildcard characters*. Given an alphabet Σ and a special *wildcard* character '$\perp$'[3], let $S \in (\Sigma \cup \{\perp\})^n$ be a stream that contains at most k wildcards. We can assign a value from Σ to each wildcard character in S resulting in many possible values of S. Then we informally say S has *wildcard-period* p if there exists an assignment to each of the wildcard characters in S so that the resulting string consists of the repetition of a block of p characters.

Example 1. The string $S = abcab\perp a\perp c\perp bc$ has wildcard-period 3, since assigning 'c' to the first wildcard character, 'b' to the second wildcard character, and 'a' to the third results in the string 'abcabcabcabc', which consists of repetitions of the substring 'abc' of length 3.

The identification of repetitive structure in data has applications to bioinformatics, natural language processing, and time series data mining. Specifically, finding the smallest period of a string is necessary preprocessing for many algorithms, such as the classic Knuth-Morriss-Pratt [KMP77] algorithm in pattern matching, or the basic local alignment search tool (BLAST) [AGM+90] in computational biology.

We consider our problem in the *streaming model*, where we process the input in sequential order and sublinear space. However in practice, some of the data may be erased or corrupted beyond repair, resulting in symbols that we cannot read, '$\perp$'. As a consequence, we attempt to perform pattern matching with optimistic assignments to these values. This motivation has resulted in a number of literature on string algorithms with wildcard characters [MR95, Ind98, CH02, Kal02, CC07, HR14, LNV14, GKP16].

One possible approach to our problem is to generalize the exact periodicity problem, for which [EJS10] give a two-pass streaming algorithm for finding the smallest *exact* period of a string of length n that uses $\mathcal{O}\left(\log^2 n\right)$-space and $\mathcal{O}\left(\log n\right)$ time per arriving symbol. Their results can be easily generalized to an algorithm for finding the wildcard-period of strings using $\mathcal{O}\left(\log^2 n\right)$-space, but at a cost of $\mathcal{O}\left(|\Sigma|^k\right)$ post-processing time, which is often undesirable. More recently, [EGSZ17] study the problem of k-periodicity, where a string is permitted to have up to k permanent changes. The authors give a two-pass streaming algorithm that uses $\mathcal{O}\left(k^4 \log^9 n\right)$ bits of space and runs in $\mathcal{O}\left(k^2 \operatorname{polylog} n\right)$ amortized time per arriving symbol. This algorithm can be modified to recover the wildcard-period. We show how to do this more efficiently in Theorem 7.

[3] Although wildcard characters are usually denoted with '?', we use $\perp$ to differentiate from compilation errors - the LaTeX equivalent of wildcard characters

1.1 Our Contributions

The challenge of determining periodicity in the presence of wildcard characters can first be approached by working toward an understanding of specific structural properties of strings with wildcard characters. We show in Lemma 2 that the number of possible assignments to the wildcard characters over all periods is "small". This allows us to compress our data into sublinear space. In this paper, given a string S with at most k wildcard characters, we show:

(1) a two-pass randomized streaming algorithm that computes all wildcard-periods of S using $\mathcal{O}\left(k^3 \operatorname{polylog} n\right)$ space, *regardless of period length*, running in $\mathcal{O}\left(k^2 \operatorname{polylog} n\right)$ amortized time per arriving symbol,
(2) a one-pass randomized streaming algorithm that computes all wildcard-periods p of S with $p < \frac{n}{2}$ and no wildcard characters appearing in the last p symbols of S, using $\mathcal{O}\left(k^3 \operatorname{polylog} n\right)$ space, running in $\mathcal{O}\left(k^2 \operatorname{polylog} n\right)$ amortized time per arriving symbol (see Appendix A),
(3) a lower bound that any one-pass streaming algorithm that computes all wildcard-periods of S requires $\Omega(n)$ space even when randomization is allowed,

We remark that our algorithm can be easily modified to return the smallest, largest, or any desired wildcard-period of S. Finally, we note in Appendix B several results in the related problem of determining distance to p-periodicity. We give an overview of our techniques in Section 2.

1.2 Related Work

The study of periodicity in data streams was initiated in [EJS10], in which the authors give an algorithm that detlects the period of a string, using $\operatorname{polylog} n$ bits of space. Independently, [BG11] gives a similar result with improved running time. Also, [EAE06] studies mining periodic patterns in streams, and [CM11] studies periodicity via linear sketches, [IKM00] studies periodicity in time-series databases and online data. [EMS10] and [LN11] study the problem of distinguishing periodic strings from aperiodic ones in the property testing model of sublinear-time computation. Furthermore, [AEL10] studies approximate periodicity in the RAM model under the Hamming and swap distance metrics.

The pattern matching literature is a vast area (see [AG97] for a survey) with many variants. In the data stream model, [PP09] and [CFP+16] study exact and approximate variants in offline and online settings. We use the sketches from [CFP+16] though there are some other works [AGMP13, CEPR09, RS17, PL07] with different sketches for strings. [CJPS13] also show several lower bounds for online pattern matching problem.

Strings with wildcard characters have been extensively studied in the offline model, usually called "partial words". Blanchet-Sadri [Bla08] presents a number of combinatorial properties on partial words, including a large section devoted to periodicity. Notably, [BMRW12] gives algorithms for determining the periodicity for partial words. Manea *et al.* [MMT14] improves these results, presenting efficient time offline algorithms for determining periodicity on partial words, minimizing either total time or update time per symbol.

Golan *et al.* [GKP16] study the pattern matching problem with a small number of wildcards in the streaming model. Prior to this work, several works had studied other aspects of pattern matching under wildcards (See [CH02], [CC07], [HR14],and [LNV14]).

Many ideas used in these sublinear algorithms stem from related work in the classical offline model. The well-known KMP algorithm [KMP77] initially used periodic structures to search for patterns within a text. Galil *et al.* [GS83] later improved the space performance of this pattern matching algorithm. Recently, [Gaw13] also used the properties of periodic strings for pattern matching when the strings are compressed. These interesting properties have allowed several algorithms to satisfy some non-trivial requirements of respective models (see [GKP16], [CFP+15] for example).

1.3 Preliminaries

Given an input stream $S[1, \ldots, n]$ of length $|S| = n$ over some alphabet Σ, we denote the i^{th} character of S by $S[i]$, and the substring between locations i and j (inclusive) $S[i, j]$. We say that two strings $S, T \in \Sigma^n$

have a *mismatch* at index i if $S[i] \neq T[i]$. Then the Hamming distance is the number of such mismatches, denoted $\Delta(S,T) = \left|\{i \mid S[i] \neq T[i]\}\right|$. We denote the concatenation of S and T by $S \circ T$. We denote the greatest common divisor of two integers x and y by $\gcd(x,y)$.

Multiple standard and equivalent definitions of periodicity are often used interchangeably. We say S has period p if $S = B^{\ell}B'$ where B is a block of length p that appears $\ell \geq 1$ times in a row, and B' is a prefix of B. For instance, *abcdabcdab* has period 4 where $B = abcd$, and $B' = ab$. Equivalently, $S[x] = S[x+p]$ for all $1 \leq x \leq n-p$. Similarly, the following definition is also used for periodicity.

Definition 1. *We say string S has period p if the length $n-p$ prefix of S is identical to its length $n-p$ suffix, $S[1,n-p] = S[p+1,n]$.*

More generally, we say S has k-period p (i.e., S has period p with k mismatches) if $S[x] = S[x+p]$ for all but at most k (valid) indices x. Equivalently, the following definition is also used for k-periodicity.

Definition 2. *We say string S has k-period p if $\Delta(S[1,n-p], S[p+1,n]) \leq k$.*

The definition of k-periodicity lends itself to the following observation.

Observation 1 *If p is a k-period of S, then at most k substrings in the sequence of substrings $S[1,p], S[p+1,2p], S[2p+1,3p], \ldots$ can differ from the preceding substring in the sequence.*

Finally, we use the following definition of wildcard-periodicity:

Definition 3. *We say that a string S has wildcard-period p if there exists an assignment to the wildcard characters, so that $S[1,n-p] = S[p+1,n]$ (i.e., the resulting string has period p. See Example 1).*

Note that the determinism of the assignments of the characters is very important, as evidenced by Example 2.

Example 2. Consider the string $S = aaa\perp bbb$. To check whether S has wildcard-period 1, we must compare $S[1,n-1] = aaa\perp bb$ and $S[2,n] = aa\perp bbb$. At first glance, one might think assigning the character 'b' to the wildcard in the prefix $S[1,n-1]$ and an 'a' in the suffix $S[2,n]$ will make the prefix and the suffix identical. However, this is not a legal move; there is not a single character that the wildcard can be replaced with that makes the above prefix and the suffix the same. Thus, S does not have a wildcard-period of 1.

The following example emphasizes the difference between k-periodicity and wildcard-periodicity:

Example 3. For $k = 1$, the string $S = aaaaabbbbb$ has k-period $p = 1$. However, to obtain wildcard-period $p = 1$, at least five characters in S must be changed to wildcards (for example, all of the characters 'a' or 'b').

Therefore, k-periodicity is a good notion for capturing periodicity with respect to long-term, persistent changes, while wildcard-periodicity is a good notion for capturing periodicity against a number of symbols that are errors or erasures.

We shall require data structures and subroutines that allow comparing of strings with mismatches. The below useful fingerprinting algorithm utilizes Karp-Rabin fingerprints [KR87] to obtain general and important properties:

Theorem 2. *[KR87] Given two strings S and T of length n, there exists a polynomial encoding that uses $\mathcal{O}(\log n)$ bits of space, and outputs whether $S = T$ or $S \neq T$. Moreover, this encoding supports concatenation of strings and can be done in the streaming setting.*

From here, we use the term *fingerprint* to refer to this data structure. We will also need use an algorithm for pattern matching with mismatches, which we call the k-mismatch algorithm.

Theorem 3. *[CFP+16] Given a string S and an index x, there exists an algorithm which, with probability $1 - \frac{1}{n^2}$, outputs all indices i where $\Delta(S[1,x], S[i+1,i+x]) \leq k$ using $\mathcal{O}\left(k^2 \log^8 n\right)$ bits of space. Moreover, the algorithm runs in $\mathcal{O}\left(k^2 \operatorname{polylog} n\right)$ amortized time per arriving symbol.*

Concurrent with our work, Clifford *et al.* [CKP17] provide a nearly-optimal solution to the k-mismatch algorithm, which can potentially be used in the framework of [EGSZ17] to immediately improve over the existing k-periodicity algorithms.

2 Our Approach

To find all the wildcard-periods of S, during our first pass we determine a set $\mathcal{T}$ of *candidate* wildcard-periods, similar to the approach in [EGSZ17], that includes all the true wildcard-periods. We also determine a set $\mathcal{W}$ of positions of the wildcard characters. By a structural result (Lemma 2), we can then use the second pass to verify the candidates and identify the true wildcard-periods.

Pattern matching and periodicity seem to have a symbiotic relationship (for example, exact pattern matching and exact periodicity use each other as subroutines [KMP77, EJS10], as do k-mismatch pattern matching [CFP+16] and k-periodicity [EGSZ17]). It feels tempting and natural to try to apply the algorithm from [GKP16] for pattern matching with wildcards. Unfortunately, there does not seem to be an immediate way of doing this: the [GKP16] algorithm searches for a wildcard-free pattern in text containing up to k wildcards, while we would like to allow wildcards in the pattern *and* the text. We instead choose to use the k-mismatch algorithm from [CFP+16] in the first pass and obtain new structural results about possible assignments to the wildcard characters in the second pass.

In the first pass, we treat wildcards simply as an additional character. We let $\mathcal{T}$ be the set of indices (candidate periods) π that satisfy

$$\Delta\left(S[1,x], S[\pi+1,\pi+x]\right) \leq 2k,$$

for some appropriate value of x that we specify later. Note that each wildcard character can cause up to two mismatches; thus, all true wildcard-periods must satisfy the above inequality. We show that $\mathcal{T}$ can be easily compressed, even though it may contain a linear number of candidates. Specifically, we can succinctly represent $\mathcal{T}$ by adding a few additional "false candidates" into $\mathcal{T}$.

If the correct assignments of the wildcards were known a priori, then the problem would reduce to determining exact periodicity. Unfortunately, we do not know the correct assignments to the wildcard characters prior to the data stream, so most of the difficulty lies in the guessing of assignments, bounding the total number of assignments, and storing these assignments. Thus, the main difference between wildcard-periodicity and both exact periodicity and k-periodicity is the process of verifying candidates. Whereas exact and k-periodicity can be verified by comparing the number of mismatches between the prefix and suffix of length $n-p$, wildcard-periodicity is sensitive to the correct assignments of the wildcards. We address this challenge by noting $\mathcal{W}$, the positions of the wildcard characters in the first pass. Since we also have the list of candidate wildcard-periods following the first pass, we can guess the assignments of the wildcard characters in the second pass by looking at the characters in a few select locations, as in Example 4.

Example 4. The string $S = ababa\perp ab$ has wildcard-period $p = 2$. The assignment of the wildcard at position $i = 6$ must be the characters at positions $i \pm p$. Note that $S[i+p] = S[8] = b$ and $S[i-p] = S[4] = b$.

From Example 4, we observe the following:

Observation 4 *If S has wildcard-period p and a wildcard character is known to be at position i, then the assignment of the wildcard must be the character $S[i \pm ap]$, for some integer a, that is not a wildcard.*

We show how to use Observation 4 and the compressed version of $\mathcal{T}$ in the second pass to verify the candidates and output the true wildcard-periods of S.

We note that recent algorithmic improvements to the k-mismatch problem [CKP17] use $\mathcal{O}\left(k\log^2 n\right)$ space. Using this algorithm in place of Theorem 3 as a subroutine in our algorithms improves the space usage to $\mathcal{O}\left(k^3\log^3 n\right)$ bits in the two-pass algorithm.

Finally, we observe that [EJS10] shows computing the period of a string in one-pass requires $\Omega(n)$ space. Since the problem of periodicity for strings containing wildcards is a generalization of exact periodicity, the same lower bound applies.

Theorem 5 (Implied from Theorem 3 from [EJS10] and Theorem 16 from [EGSZ17]). *Given a string S with at most k wildcard characters, any one-pass streaming algorithm that computes the smallest wildcard-period requires $\Omega(n)$ space.*

3 Two-Pass Algorithm to Compute Wildcard-Periods

In this section, we provide a two-pass, $\mathcal{O}\left(k^3 \log^9 n\right)$-space algorithm to output all wildcard-periods of some string S containing at most k wildcard characters. At a high level, we first identify a list of candidates of the periods of S, detected via the k-mismatch algorithm of [CFP+16] as a black box. Although the number of candidates could be linear, it turns out the string has enough structure that the list of candidates can be succinctly expressed as the union of k arithmetic progressions.

However, this list of candidates is insufficient in identifying the possible assignments to the wildcard characters. To address this issue, we explore the structure of periods with wildcards in order to limit the possible assignments for each wildcard character. Thus, the first pass also records $\mathcal{W}$, the positions of all wildcard characters so that during the second pass, we go over S as well as the compressed data to verify the candidate periods.

We present two algorithms in parallel to find the periods, based on their lengths. The first algorithm identifies all periods p with $p \leq \frac{n}{2}$, while the second algorithm identifies all periods p with $p > \frac{n}{2}$.

3.1 Computing Small Wildcard-Periods

In this section, we describe a two-pass algorithm for finding wildcard-periods of length at most $n/2$. The first pass of the algorithm identifies a set $\mathcal{T}$ of candidate wildcard-periods in terms of indices of S, and maintains its succinct representation $\mathcal{T}^C$, which includes a number of additional indices. It also records $\mathcal{W}$, the positions of all wildcard characters. The second pass of the algorithm recovers each index of $\mathcal{T}$ from $\mathcal{T}^C$ and verifies whether or not the index is a wildcard-period. We can find the assignments of the wildcard characters in the second pass, by looking at the characters in a few locations that we determine via $\mathcal{W}$. We emphasize the following properties of $\mathcal{T}$ and $\mathcal{T}^C$:

(1) All wildcard-periods (possibly as well as additional candidate wildcard-periods that are false positives) are in $\mathcal{T}$.

(2) $\mathcal{T}^C$ can be stored in sublinear space and $\mathcal{T}$ can be fully recovered from $\mathcal{T}^C$.

(3) In the second pass, we can verify and eliminate in sublinear space candidates that are not true periods.

In the first pass, we treat the wildcard characters as a regular, additional alphabet symbol. We observe that if string S with such wildcards has wildcard-period p, there are at most $2k$ indices i such that $S[i] \neq S[i+p]$, caused by the wildcard characters (the converse is not necessarily true). It follows that any wildcard-period p must satisfy

$$\Delta\left(S[1,x], S[p+1,p+x]\right) \leq 2k$$

for all $x \leq n-p$, and specifically for $x = \frac{n}{2}$. Thus, we set $x = \frac{n}{2}$ and refer to any index p that satisfies $\Delta\left(S[1,x], S[p+1,p+x]\right) \leq 2k$ as a *candidate wildcard-period.* The set of all candidate wildcard-periods forms the set $\mathcal{T}$. Because $\Delta\left(S[1,x], S[p+1,p+x]\right) \leq 2k$ is a necessary but not sufficient condition for a wildcard-period p, Property 1 follows.

We give the first pass of the algorithm in full in Algorithm 1.

Algorithm 1 (To determine any wildcard-period p with $p \leq \frac{n}{2}$) First pass

Input: A stream S of n symbols $s_i \in \Sigma \cup \{\perp\}$ with at most k wildcard characters $\perp$.
Output: A succinct representation of all candidate wildcard periods and the positions of the wildcard characters.

initialize $\pi_j = -1$ for each $0 \leq j < 4k \log n + 2$.
initialize $\mathcal{T}^C = \emptyset$.
for each index i (found using the k-mismatch algorithm) such that

$$\Delta\left(S\left[1, \frac{n}{2}\right], S\left[i+1, \frac{n}{2}+i\right]\right) \leq 2k$$

do
    consider j for which i is in the interval $H_j = \left[\frac{jn}{4(2k \log n+1)} + 1, \frac{(j+1)n}{4(2k \log n+1)}\right)$:
    if there exists no candidate $t \in \mathcal{T}^C$ in the interval H_j **then**
        add i to $\mathcal{T}^C$.
    else
        let t be the smallest candidate in $\mathcal{T}^C \cap H_j$ and either $\pi_j = -1$ or $\pi_j > 0$.
        if $\pi_j = -1$ **then**
            set $\pi_j = i - t$.
        else
            set $\pi_j = \mathsf{gcd}(\pi_j, i - t)$.
record the positions $\mathcal{W}$ of all wildcard characters.

Here, we show why the remaining properties for $\mathcal{T}$ and $\mathcal{T}^C$ are satisfied. Our algorithm divides the candidates into $\mathcal{O}(k \log n)$ ranges $H_1, H_2, \ldots, H_{\mathcal{O}(k \log n)}$ and stores the candidates in each range $H_j = \left[\frac{jn}{4(2k \log n+1)} + 1, \frac{(j+1)n}{4(2k \log n+1)}\right)$ in compressed form as an arithmetic series.

Since we use the k-mismatch algorithm in the first pass, we describe a structural property of the resulting list of candidates:

Theorem 6. *[EGSZ17] Let p_i be a candidate k-period for a string S, with $p_1 < p_2 < \ldots < p_m$ all contained within H_j. Given the fingerprints of $S[1, n - p_1]$ and $S[p_1 + 1, n]$, we can determine whether or not S has k-period p_i for any $1 \leq i \leq m$ by storing at most $\mathcal{O}(k^2 \log n)$ additional fingerprints. These fingerprints represent substrings of the form $S[p_1 + a\pi_j, p_1 + (a+1)\pi_j - 1]$, where $a > 0$ is an integer and $\pi_j = \mathsf{gcd}(p_2 - p_1, p_3 - p_2, \ldots, p_m - p_{m-1})$.*

The structural property can be visualized in Figure 1. Even though the list of candidates could be linear in

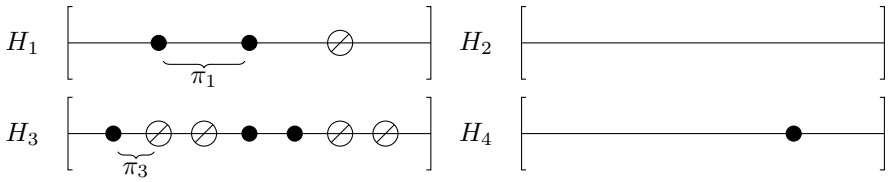

Fig. 1. The dots represent candidate wildcard-periods. For any interval that has more than two dots, it follows that all dots are equally spaced after the first. The black dots represent $\mathcal{T}$ while white dots are artificially inserted to form $\mathcal{T}$, dots that follow an arithmetic sequence.

size, Theorem 6 enforces a structure upon the list of candidates, so that an arithmetic sequence with first term p_1 and common difference d includes all of $p_1, p_2, \ldots, p_m$. Thus, we can succinctly represent a superset $\mathcal{T}^C$ that contains $\mathcal{T}$ and Property 2 follows.

We now show that any wildcard period p is included among the list of candidates stored by Algorithm 1 during the first pass, and can be recovered from the list.

Lemma 1. *If $p < \frac{n}{2}$ is a period and $p \in H_j$, then p can be recovered from $\mathcal{T}^C$ and π_j.*

Proof. Suppose $p \in H_j$ is a wildcard period. Then there exists an assignment to the wildcard characters such that $S[1, n-p] = S[p+1, n]$. It follows that for $i = p$,

$$\Delta\left(S\left[1, \frac{n}{2}\right], S\left[i+1, \frac{n}{2}+i\right]\right) \leq 2k,$$

so the index $i = p$ will be reported by the k-mismatch algorithm in the first pass.

If at that time during Pass 1 there is no other index in $\mathcal{T}^C \cap H_j$, then p will be inserted into $\mathcal{T}^C$, so p can clearly be recovered from $\mathcal{T}^C$. If there is another index q in $\mathcal{T}^C \cap H_j$, then π_j will be updated to be a divisor of $p - q$. Hence, $p - q$ is a multiple of π_j. Furthermore, any future update to π_j will result in a value that divides the current value of π_j, due to a greatest common divisor operation. Thus, $p - q$ will remain a multiple of the final value of π_j, and so the set $\mathcal{T}$ at the end of the first pass will contain p.

It remains to show that the list of candidate wildcard-periods can be verified in sublinear space in the second pass (Property 3). To do this, we need a combinatorial property for periodicity on strings with wildcard characters.

3.2 Verifying Candidates

Recall that after the first pass, the algorithm maintains $\mathcal{O}(k \log n)$ succinctly represented arithmetic progressions H_j, corresponding to the candidate wildcard periods. The algorithm also maintains $\mathcal{W}$, the list of positions of wildcard characters in S. In the second pass, the algorithm must check, for each $t \in H_j$, $0 \leq j < 2k \log n + 2$, whether $S[1, n-t] = S[t+1, n]$ for an appropriate setting of the wildcard characters. The challenge is computing the fingerprints of both $S[1, n-t]$ and $S[t+1, n]$ in sublinear space, especially if the number of candidates t is linear.

We first set a specific j and note that for the smallest candidate $t \in H_j$, there are at most $\mathcal{O}\left(k^2 \log n\right)$ unique substrings $S[t+1, t+\pi_j]$, $S[t+\pi_j+1, t+2\pi_j]$, $S[t+2\pi_j+1, t+3\pi_j]$, ... Since any other candidate $r \in H_j$ satisfies $r = t + a\pi_j$ for some integer $a > 0$, then $S[t+1, n]$ is the concatenation

$$S[t+1, t+\pi_j] \circ S[t+\pi_j+1, t+2\pi_j] \circ \cdots \circ S[t+(a-1)\pi_j+1, t+a\pi_j] \circ S[r+1, n].$$

Thus, by storing $\mathcal{O}\left(k^2 \log n\right)$ fingerprints and positions, we can recover the fingerprint of the substring $S[r+1, n]$ for each $r \in H_j$.

The second obstacle is handling wildcard characters in the computation of the fingerprints of $S[1, n-t]$ and $S[t+1, n]$. To address this challenge, our algorithm delays the calculation of the contribution of wildcard characters to the fingerprints until we know the assignment of the wildcard character with respect to a candidate period. We show that for a specific j, then there are at most $\mathcal{O}\left(k^2 \log n\right)$ possible assignments for the wildcard character $S[w] = S[w \pm t]$ with respect to all candidates $t \in H_j$, across all $w \in \mathcal{W}$, where $\mathcal{W}$ is the positions of all wildcard characters recorded by Algorithm 1. Therefore, we can compute the assignment for each wildcard character with respect to a candidate period in the second pass, and then compute the fingerprint of $S[1, n-t]$ and $S[t+1, n]$.

Lemma 2. *For a given j, $t \in H_j$ and $w \in \mathcal{W}$, let $\sigma_t(w)$ denote the assignment of $S[w]$. Then $|\{\sigma_t(w)\}| = \mathcal{O}\left(k^2 \log n\right)$.*

Proof. Let t be the smallest candidate in H_j and z be the largest candidate in H_j so that $z = t + a\pi_j$ for some integer $a > 0$. We partition $\mathcal{W}$ into $\mathcal{W}_1$, the set of indices greater than z, and $\mathcal{W}_2$, the set of indices no more than z. We consider the wildcard characters $w_i \in \mathcal{W}_1$, and note that the proof for $\mathcal{W}_2$ is symmetric. Consider the $\mathcal{O}(k)$ sequences

$$\begin{matrix} S[w_1 - t] & S[w_1 - t - \pi_j] & \cdots & S[w_1 - t - a\pi_j] \\ S[w_2 - t] & S[w_2 - t - \pi_j] & \cdots & S[w_2 - t - a\pi_j] \\ \vdots & \vdots & \ddots & \vdots \\ S[w_{|\mathcal{W}_1|} - t] & S[w_{|\mathcal{W}_1|} - t - \pi_j] & \cdots & S[w_{|\mathcal{W}_1|} - t - a\pi_j] \end{matrix}$$

Each term in a sequence that differs from the previous term corresponds to a mismatch between $S[w_i - t - \pi_j + 1, w_i - t]$, $S[w_i - t - 2\pi_j + 1, w_i - t - \pi_j]$, $S[w - t - 3\pi_j + 1, w - t - 2\pi_j], \ldots$. For each j, there are at most $\mathcal{O}\left(k^2 \log n\right)$ unique chains of substrings with length π_j beginning at index $t+1$. Hence, across all $\mathcal{O}(k)$ sequences $S[w_i - t]$, $S[w_i - t - \pi_j]$, $S[w_i - t - 2\pi_j], \ldots$, there are at most $\mathcal{O}\left(k^2 \log n\right)$ unique characters. Since the assignment of $S[w_i]$ with respect to any candidate $r \in H_j$ is $S[w_i - r] = S[w_i - t - b\pi_j]$ for some integer $b > 0$, then it follows that there are at most $\mathcal{O}\left(k^2 \log n\right)$ assignments of $S[w]$ across all $w \in \mathcal{W}_1$. As the symmetric proof holds for $\mathcal{W}_2$, then there are at most $\mathcal{O}\left(k^2 \log n\right)$ assignments of $S[w]$ across all $w \in \mathcal{W}$.

Thus, deciding the assignment of $S[w_i]$ with respect to a candidate $t \in H_j$ is simple:

For each j such that $0 \leq j < 4k \log n + 2$:

(1) Let t be the smallest candidate in H_j and z be the largest candidate in H_j so that $z = t + a\pi_j$ for some $a > 0$.
(2) For each $w \in \mathcal{W}$:
  (a) If $w > z$, succinctly record the values of $S[w - t]$, $S[w - t - \pi_j]$, …, $S[w - t - a\pi_j]$.
  (b) If $w \leq z$, succinctly record the values of $S[w + t]$, $S[w + t + \pi_j]$, …, $S[w + t + a\pi_j]$.
  Let $r \in H_j$ so that $r = t + b\pi_j$ for some $b > 0$.
(3) The assignment of $S[w]$ with respect to r is any $S[w \pm cr]$ that is not a wildcard character (where c is an integer).

We describe the second pass in Algorithm 2, recalling that at the end of the first pass, the algorithm records $\mathcal{O}(k \log n)$ arithmetic progressions, succinctly represented, as well as the positions of all wildcard characters.

Algorithm 2 (To determine any wildcard-period p with $p \leq \frac{n}{2}$) Second pass

Input: A stream S of symbols $s_i \in \Sigma$ with at most k wildcard characters, a succinct representation of all candidate wildcard periods and the position of the wildcard characters.
Output: All wildcard-periods $p \leq \frac{n}{2}$.

for each t such that $t \in \mathcal{T}^C$ **do**
  for each w such that $w \in \mathcal{W}$, implicitly determine the value of $S[w]$ with respect to t.
  let j be the integer for which t is in the interval $H_j = \left[\frac{jn}{4(2k \log n+1)} + 1, \frac{(j+1)n}{4(2k \log n+1)}\right)$
  if $\pi_j > 0$ **then** ▷ H_j has multiple values in $\mathcal{T}^C$
    record up to $128k^2 \log n + 1$ unique fingerprints of length π_j, starting from t.
  else ▷ H_j has one value in $\mathcal{T}^C$
    record up to $128k^2 \log n + 1$ unique fingerprints of length t, starting from t.
  check if $S[1, n - t] = S[t + 1, n]$ and return t if this is true.
for each t which is in interval $H_j = \left[\frac{jn}{4(2k \log n+1)} + 1, \frac{(j+1)n}{4(2k \log n+1)}\right)$ for some integer j **do**
  if there exists an index in $\mathcal{T}^C \cap H_j$ whose distance from t is a multiple of π_j **then**
    check if $S[1, n - t] = S[t + 1, n]$ and return t if this is true.

For each arithmetic progression, there are $\mathcal{O}\left(k^2 \log n\right)$ total possibilities for all of the wildcard characters. Thus, the algorithm maintains the $\mathcal{O}\left(k^3 \log^2 n\right)$ characters corresponding to the value of all wildcard characters across all candidate positions.

We now show the ability to construct the fingerprints of $S[1, n - p]$ for any candidate period p.

Lemma 3. *Let p_i be a candidate k-period for a string S, with $p_1 < p_2 < \ldots < p_m$ all contained within H_j. Given the fingerprints of $S[1, n - p_1]$ and $S[p_1 + 1, n]$, we can determine whether or not S has wildcard-period p_i for any $1 \leq i \leq m$ by storing at most $\mathcal{O}\left(k^2 \log n\right)$ additional fingerprints.*

Proof. Consider a decomposition of S into substrings u_j of length p_i, so that $S = u_1 \circ u_2 \circ u_3 \circ \ldots$. Even though the algorithm does not record a fingerprint for each u_j, each index j for which $u_j \neq u_{j+1}$ corresponds

to at least one mismatch. Since the first pass searched for positions that contained at most k mismatches, then it follows from Observation 1 that there are $\mathcal{O}(k)$ indices j for which $u_j \neq u_{j+1}$. Thus, recording the fingerprints and locations of these indices j suffices to build fingerprints for S, ignoring the wildcard characters. Then we can verify whether or not p_i is a wildcard-period of S if the assignment of the wildcard characters with respect to p_i is also known.

By Theorem 6, the greatest common divisor π_j of the difference between each p_i in H_j is a $\mathcal{O}\left(k^2 \log n\right)$-period. That is, S can be decomposed $S = v \circ v_1 \circ v_2 \circ v_3 \circ \dots$ so that v has length p_1, and each subsequent substring v_i has length π_j. Then there exist at most $\mathcal{O}\left(k^2 \log n\right)$ indices i for which $v_i \neq v_{i+1}$, by Observation 1. Ignoring wildcard characters, storing the fingerprints and positions of these indices i allows the recovery of the fingerprint of $S[1, n-p_i]$ from the fingerprint of $S[1, n-p_{i-1}]$, since $p_i - p_{i-1}$ is a multiple of π_j. By Lemma 2, we know the values of the wildcard characters with respect to p_i. Therefore, we can confirm whether or not p_i is a wildcard-period.

We now show correctness of the algorithm.

Lemma 4. *For any period $p \leq \frac{n}{2}$, the algorithm outputs p.*

Proof. Since the intervals $\{H_j\}$ cover $\left[1, \frac{n}{2}\right]$, then $p \in H_j$ for some j. It follows from Lemma 1 that after the first pass, p can be recovered from $\mathcal{T}$ and π_j. Thus, the second pass tests whether or not p is a wildcard-period. By Lemma 3, the algorithm outputs p, as desired.

3.3 Computing Large Wildcard-Periods

As in Algorithm 1, we would like to identify candidate periods during the first pass of the algorithm, while treating the wildcard characters as an additional symbol in the alphabet. Unfortunately, if a wildcard-period p is greater than $\frac{n}{2}$, then it no longer satisfies

$$\Delta\left(S\left[1, \frac{n}{2}\right], S\left[p+1, p+\frac{n}{2}\right]\right) \leq 2k,$$

since $p + \frac{n}{2} > n$, and $S\left[p + \frac{n}{2}\right]$ is undefined. However, by treating the wildcard characters as an additional symbol, recall that $\Delta(S[1,x], S[p+1, p+x]) \leq 2k$ for all $x \leq n - p$. Then we would like to use as large an x as possible while still satisfying $x \leq n - p$ when choosing candidate wildcard periods p. To this effect, the observation in [EJS10] states that we can try exponentially decreasing values of x. Specifically, we run $\log n$ instances of the algorithm in succession, with $x = \frac{n}{2}, \frac{n}{4}, \dots$. Note that one of these values of x is the largest value as possible while still satisfying $x \leq n - p$. As a result, the corresponding algorithm instance outputs p, while the other instances do not output anything. We detail the first pass in full in Algorithm 3 in Appendix C.

This partition of $[1, n]$ into the disjoint intervals $\left[1, \frac{n}{2}\right]$, $\left[\frac{n}{2}+1, \frac{n}{2}+\frac{n}{4}\right]$, ... guarantees that any k-period p is contained in one of these intervals. Moreover, the intervals $\{H_j^{(r)}\}$ partition

$$\left[\frac{n}{2} + \frac{n}{4} + \dots + \frac{n}{2^{r-1}}, \frac{n}{2} + \dots + \frac{n}{2^r}\right],$$

and so p can be recovered from $\mathcal{T}_r^C$ and $\{\pi_j^{(r)}\}$. We present the second pass in Algorithm 4 in Appendix C.

Since correctness follows from the same arguments as the case where $p \leq \frac{n}{2}$, it remains to analyze the space complexity of our algorithm.

Theorem 7. *There exists a two-pass randomized algorithm using $\mathcal{O}\left(k^3 \log^9 n\right)$ bits of space that finds the wildcard-period and runs in $\mathcal{O}\left(k^2 \operatorname{polylog} n\right)$ amortized time per arriving symbol.*

Proof. In the first pass, for each $\mathcal{T}_m$, we maintain a k-mismatch algorithm which requires $\mathcal{O}\left(k^2 \log^8 n\right)$ bits of space, as in Theorem 3. Since $1 \leq m \leq \log n$, we use $\mathcal{O}\left(k^2 \log^9 n\right)$ bits of space in total in the first pass.

In the second pass, we maintain $\mathcal{O}\left(k^2 \log n\right)$ fingerprints for any set of indices in $\mathcal{T}_m$, and there are $\mathcal{O}\left(k \log n\right)$ indices in $\mathcal{T}_m$ for each $1 \leq m \leq \log n$, for a total of $\mathcal{O}\left(k^3 \log^3 n\right)$ bits of space. In addition, we store the $\mathcal{O}\left(k^2 \log n\right)$ assignments for all the wildcard positions in each interval $H_j^{(r)}$, where $1 \leq r \leq \log n$ and $0 \leq j < 2k \log n + 2$. Thus, $\mathcal{O}\left(k^3 \log^9 n\right)$ bits of space suffice for both passes.

The running time of the algorithm is dominated by the time spent for $\log n$ parallel copies of k-mismatch algorithm in the first pass, i.e., Algorithm 3. From Theorem 3, the k-mismatch algorithm runs in $\mathcal{O}\left(k^2 \,\mathsf{polylog}\, n\right)$ amortized time per arriving symbol. The rest of the algorithm consists of simple tasks like computing gcd and can be performed very quickly. In the second pass, in total at most $\mathcal{O}\left(k^3 \,\mathsf{polylog}\, n\right)$ assignments are determined and stored. Thus, the second pass runs in $\mathcal{O}\left(1\right)$ amortized time per arriving symbol.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. The work was supported by the National Science Foundation under NSF Awards #1649515 and #1619081.

References

AEL10. Amihood Amir, Estrella Eisenberg, and Avivit Levy. Approximate periodicity. *Algorithms and Computation*, pages 25–36, 2010. 1.2

AG97. Alberto Apostolico and Zvi Galil, editors. *Pattern Matching Algorithms*. Oxford University Press, Oxford, UK, 1997. 1.2

AGM+90. Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990. 1

AGMP13. Alexandr Andoni, Assaf Goldberger, Andrew McGregor, and Ely Porat. Homomorphic fingerprints under misalignments: sketching edit and shift distances. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 931–940, 2013. 1.2

BG11. Dany Breslauer and Zvi Galil. Real-time streaming string-matching. In *Combinatorial Pattern Matching*, pages 162–172. Springer, 2011. 1.2

Bla08. Francine Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words*. Discrete mathematics and its applications. CRC Press, 2008. 1.2

BMRW12. Francine Blanchet-Sadri, Robert Mercas, Abraham Rashin, and Elara Willett. Periodicity algorithms and a conjecture on overlaps in partial words. *Theor. Comput. Sci.*, 443:35–45, 2012. 1.2

CC07. Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007. 1, 1.2

CEPR09. Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 778–784, 2009. 1.2

CFP+15. Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *Algorithms - ESA 23rd Annual European Symposium, Proceedings*, pages 361–372, 2015. 1.2

CFP+16. Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2039–2052, 2016. 1.2, 3, 2, 3, 8

CH02. Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 592–601, 2002. 1, 1.2

CJPS13. Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. Space lower bounds for online pattern matching. *Theoretical Computer Science*, 483:68–74, 2013. 1.2

CKP17. Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. *CoRR*, abs/1708.05223, 2017. 1.3, 2

CM11. Michael S. Crouch and Andrew McGregor. Periodicity and cyclic shifts via linear sketches. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX, and 15th International Workshop, RANDOM. Proceedings*, pages 158–170, 2011. 1.2

EAE06. Mohamed G. Elfeky, Walid G. Aref, and Ahmed K. Elmagarmid. STAGGER: periodicity mining of data streams using expanding sliding windows. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM)*, pages 188–199, 2006. 1.2

EGSZ17. Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming periodicity with mismatches. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 42:1–42:21, 2017. 1, 1.3, 2, 5, 6, A.2

EJS10. Funda Ergün, Hossein Jowhari, and Mert Saglam. Periodicity in streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010. Proceedings*, pages 545–559, 2010. 1, 1.2, 2, 2, 5, 3.3, B, B

EMS10. Funda Ergün, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Periodicity testing with sublinear samples and space. *ACM Trans. Algorithms*, 6(2):43:1–43:14, 2010. 1.2

Gaw13. Pawel Gawrychowski. Optimal pattern matching in lzw compressed strings. *ACM Transactions on Algorithms (TALG)*, 9(3):25, 2013. 1.2

GKP16. Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming pattern matching with d wildcards. In *24th Annual European Symposium on Algorithms*, pages 44:1–44:16, 2016. 1, 1.2, 2

GS83. Zvi Galil and Joel Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983. 1.2

HR14. Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. In *Combinatorial Pattern Matching - 25th Annual Symposium, CPM Proceedings*, pages 140–149, 2014. 1, 1.2

IKM00. Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB, Proceedings of 26th International Conference on Very Large Data Bases*, pages 363–372, 2000. 1.2

Ind98. Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *39th Annual Symposium on Foundations of Computer Science, FOCS*, pages 166–173, 1998. 1

Kal02. Adam Kalai. Efficient pattern-matching with don't cares. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 655–656, 2002. 1

KMP77. Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. 1, 1.2, 2

KNW10. Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 41–52, 2010. B

KR87. Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. 1.3, 2

LN11. Oded Lachish and Ilan Newman. Testing periodicity. *Algorithmica*, 60(2):401–420, 2011. 1.2

LNV14. Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter. Space-efficient string indexing for wildcard pattern matching. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 506–517, 2014. 1, 1.2

MG82. Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982. B

MMT14. Florin Manea, Robert Mercas, and Catalin Tiseanu. An algorithmic toolbox for periodic partial words. *Discrete Applied Mathematics*, 179:174–192, 2014. 1.2

MR95. S. Muthukrishnan and H. Ramesh. String matching under a general matching relation. *Inf. Comput.*, 122(1):140–148, 1995. 1

PL07. Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Annual Symposium on Combinatorial Pattern Matching*, pages 173–182, 2007. 1.2

PP09. 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*, pages 315–323, 2009. 1.2

RS17. Jakub Radoszewski and Tatiana A. Starikovskaya. Streaming k-mismatch with error correcting and applications. In *2017 Data Compression Conference, DCC*, pages 290–299, 2017. 1.2

A One-Pass Algorithm to Compute Small Wildcard-Periods

In this section, we address the problem of computing any wildcard-period p that satisfies $p < \frac{n}{2}$, under the condition that no wildcard character appears in the last p symbols of the string. As in Section 3, we run two algorithms in parallel. The first algorithm will return any wildcard-period that satisfies $p \leq \frac{n}{4}$ and the second algorithm will return any wildcard-period that satisfies $\frac{n}{4} \leq p < \frac{n}{2}$. In the first process, we identify all indices i such that $\Delta\left(S\left[i+1, i+\frac{n}{2}\right], S\left[1, \frac{n}{2}\right]\right) \leq k$. We simultaneously track the positions of the wildcard characters and the symbol that is i positions away from each wildcard character, so that we know the assignment of each wildcard character with respect to each candidate period. Unfortunately, the second process cannot use the same paradigm, since the k-Mismatch algorithm reports candidate periods too late for fingerprints to be built. As a result, we must pre-emptively guess the candidate periods.

A.1 Computing Small Wildcard-Periods

In this section, we describe the algorithm that finds any wildcard-period p with $p \leq \frac{n}{4}$. We first designate wildcard characters as unique characters and run the k-mismatch algorithm to find

$$\mathcal{T} = \left\{ i \,\middle|\, i \leq \frac{n}{4}, \Delta\left(S\left[1, \frac{n}{2}\right], S\left[i+1, i+\frac{n}{2}\right]\right) \leq k \right\}.$$

When the k-mismatch algorithm finds indices $i \in \mathcal{T}$, we use the fingerprints for $S\left[1, \frac{n}{2}\right]$ and $S\left[i+1, i+\frac{n}{2}\right]$ to simultaneously build the fingerprint for $S[1, n-i]$ and continue building the fingerprint for $S[i+1, n]$ respectively. Concurrently, we also track the positions of each wildcard character. For some position w of a wildcard character, we identify any arbitrary non-wildcard character that is at a position $w \pmod i$. By Lemma 2, we can do this in $\mathcal{O}\left(k^2 \log n\right)$ space, and thus replace the wildcard characters in the fingerprints of $S[1, n-i]$ and $S[i+1, n]$.

The k-mismatch algorithm outputs $i \in \mathcal{T}$ upon reading character $i + \frac{n}{2} - 1$. Thus for $i \leq \frac{n}{4}$, it follows that $i + \frac{n}{2} - 1 < \frac{3n}{4} \leq n - i$ so we can identify i in time to build $S[1, n-i]$. From Theorem 6, we can build each of these fingerprints from a sequence of compressed fingerprints.

A.2 Computing Large Wildcard-Periods

We now describe an algorithm for identifying all wildcard-periods p such that $\frac{n}{4} < p \leq \frac{n}{2}$. Let I_m be the interval $\left[\frac{n}{2} - 2^m + 1, \frac{n}{2} - 2^{m-1}\right]$ of length 2^{m-1} for $1 \leq m \leq \log n - 1$ and again define a set of candidate periods:

$$\mathcal{T}_m = \{ i \,|\, i \in I_m, \Delta\left(S[1, 2^m], S[i+1, i+2^m]\right) \leq k \}.$$

Let π_m be a wildcard-period of $S[1, 2^m]$. We first consider the case where $\pi_m \geq \frac{2^m}{4}$ and then the case where $\pi_m < \frac{2^m}{4}$.

Observation 8 *[CFP+ 16] If p is a k-period for $S[1, n/2]$, then each i such that*

$$\Delta\left(S\left[1, \frac{n}{2}\right], S\left[i+1, i+\frac{n}{2}\right]\right) \leq \frac{k}{2}$$

must be at least p symbols apart.

By Observation 8, if $\pi_m \geq \frac{2^m}{4}$, then $|\mathcal{T}_m| \leq 4$. Moreover, we can detect whether $i \in \mathcal{T}_m$ by index $\frac{n}{2} - 2^{m-1} + 2^m$. On the other hand, $n - i \geq \frac{n}{2} + 2^m + 1$, and so we can properly build the fingerprint of $S[1, n-i]$.

Now, consider the case where $\pi_m < \frac{2^m}{4}$. [EGSZ17] show that we can compute the fingerprint of $S\left[\frac{n}{2} + 1, n - i\right]$ by storing the fingerprints and positions of $\mathcal{O}\left(k^2 \log n\right)$ substrings.

Thus, we can build the fingerprint of $S[1, n-i]$ regardless of whether $\pi_m < \frac{2^m}{4}$ or $\pi_m \geq \frac{2^m}{4}$. In both cases, we again simultaneously track the positions of each wildcard character. For some position w of a wildcard character, we identify any arbitrary non-wildcard character that is at a position $w \pmod i$.

By a similar reasoning to Lemma 2, we can do this in $\mathcal{O}\left(k^2 \log n\right)$ space, and thus replace the wildcard characters in the fingerprints of $S[1, n-i]$ and $S[i+1, n]$.

Theorem 9. *There exists a one-pass algorithm that outputs all the wildcard-periods p of a given string with $p \leq \frac{n}{2}$, and uses $\mathcal{O}\left(k^3 \log^9 n\right)$ bits of space.*

Proof. The k-mismatch subroutine that identifies candidate wildcard-periods uses $\mathcal{O}\left(k^2 \log^8 n\right)$ bits of space. We also maintain $\mathcal{O}\left(k^2 \log n\right)$ fingerprints for any set of indices in $\mathcal{T}_m$, and there are $\mathcal{O}\left(k \log n\right)$ indices in $\mathcal{T}_m$ for each $1 \leq m \leq \log n$, for a total of $\mathcal{O}\left(k^3 \log^3 n\right)$ fingerprints. In addition, we store the $\mathcal{O}\left(k^2 \log n\right)$ assignments for all the wildcard positions in each interval $H_j^{(m)}$, where $1 \leq m \leq \log n$ and $0 \leq j < 2k \log n + 2$. Thus, $\mathcal{O}\left(k^3 \log^9 n\right)$ bits of space suffice.

B Distance to p-Periodicity

In this section, we address the problem of finding distance $\delta_p(S)$ to p-periodicity in a string S of length n containing wildcard characters. That is, we find the minimum number of character changes in S to obtain a string that has wildcard-period p.

Suppose without loss of generality that p divides n, so that $n = ap$ for some integer $a > 0$. Then S can be visualized as a $p \times a$ matrix M so that $M_{i,j} = S[(j-1)p + i]$. Intuitively, $\delta_p(S)$ is the smallest number of changes to entries in matrix M so that all the characters in each row are the same. Let $f_{-1}(M_i)$ be the frequency vector of the entries in M_i, the i^{th} row of M, excluding both the most frequent character of M_i and any wildcard characters that appear in M_i. Then it follows that

$$\delta_p(S) = \sum_{i=1}^{p} f_{-1}(M_i).$$

It remains to estimate $f_{-1}(M_i)$ using one of several well-known techniques. Indeed, [EJS10] uses several references to obtain results that directly translate to strings containing wildcard characters. For example, [EJS10] use a heavy-hitter algorithm from [MG82] to approximate $f_{-1}(M_i)$. We can slightly modify the technique by ignoring wildcard characters to obtain the following result:

Theorem 10. *There exists a deterministic one-pass streaming algorithm that provides a $(1+\epsilon)$-approximation of $\delta_p(S)$ using $\mathcal{O}\left(\frac{p \log n}{\epsilon}\right)$ bits of space.*

Similarly, [EJS10] use a distinct-elements algorithm from [KNW10] to approximate $f_{-1}(M_i)$. Again, the technique can be modified by ignoring wildcard characters to obtain the following result:

Theorem 11. *There exists a one-pass streaming algorithm that provides a $(2+\epsilon)$-approximation of $\delta_p(S)$ with probability at least $1-\delta$, using $\mathcal{O}\left(\frac{\log n}{\epsilon^2} \log \frac{1}{\epsilon} \log \frac{1}{\delta}\right)$ bits of space.*

C Full Algorithms

In this section, we provide the full algorithms for finding wildcard-periods $p > \frac{n}{2}$. We detail the first pass in full in Algorithm 3. We present the second pass in Algorithm 4.

Algorithm 3 (To determine any wildcard-period p if $p > \frac{n}{2}$) First pass

Input: A stream S of symbols $s_i \in \Sigma$ with at most k wildcard characters.
Output: A succinct representation of all candidate wildcard periods and the position of the wildcard characters.

initialize $\pi_j^{(m)} = -1$ for each $0 \leq j < 4k \log n + 2$ and $0 \leq m \leq \log n$.
initialize $\mathcal{T}_m^C = \emptyset$.
for each index i, let r be the largest m such that $\frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^r} \leq i$. **do**
    using the k-mismatch algorithm, check whether

$$\Delta\left(S\left[1, \frac{n}{2^r}\right], S\left[i+1, i+\frac{n}{2^r}\right]\right) \leq 2k.$$

    if so, let $R = \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{r-1}}$. **then**
        let j be the integer for which i is in the interval

$$H_j^{(r)} = \left[R + \frac{nj}{2^{r+1}(2k \log n + 1)} + 1, R + \frac{n(j+1)}{2^{r+1}(2k \log n + 1)}\right)$$

        if there exists no candidate $t \in \mathcal{T}_r^C$ in the interval $H_j^{(r)}$ **then**
            add i to $\mathcal{T}_r^C$.
        else
            let t be the smallest candidate in $\mathcal{T}_r^C \cap H_j^{(r)}$ and either $\pi_j^{(r)} = -1$ or $\pi_j^{(r)} > 0$.
            if $\pi_j^{(r)} = -1$ **then**
                set $\pi_j^{(r)} = i - t$.
            else
                set $\pi_j^{(r)} = \mathsf{gcd}\left(\pi_j^{(r)}, i - t\right)$.
record the positions $\mathcal{W}$ of all wildcard characters

Algorithm 4 (To determine any wildcard-period p with $p > \frac{n}{2}$) Second pass

Input: A stream S of symbols $s_i \in \Sigma$ with at most k wildcard characters, a succinct representation of all candidate wildcard periods and the position of the wildcard characters.
Output: All wildcard-periods $p > \frac{n}{2}$.

for each t and any r such that $t \in \mathcal{T}_r^C$ **do**
    Let $R = \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{r-1}}$
    Let j be the integer for which t is in the interval

$$H_j^{(r)} = \left[R + \frac{nj}{2^{r+1}(2k \log n + 1)} + 1, R + \frac{n(j+1)}{2^{r+1}(2k \log n + 1)}\right)$$

    if $\pi_j^{(r)} > 0$ **then** ▷ $H_j^{(r)}$ has multiple values in $\mathcal{T}_r^C$
        record up to $128k^2 \log n + 1$ unique fingerprints of length $\pi_j^{(r)}$, starting from t.
    else ▷ $H_j^{(r)}$ has one value in $\mathcal{T}_r^C$
        record up to $128k^2 \log n + 1$ unique fingerprints of length t, starting from t.
    check if $S[1, n-t] = S[t+1, n]$ and return t if this is true.
for each t which is in interval $H_j^{(r)} = \left[R + \frac{nj}{2^{r+1}(2k \log n+1)} + 1, R + \frac{n(j+1)}{2^{r+1}(2k \log n+1)}\right)$, for some integer j **do**
    if there exists an index in $\mathcal{T}_r^C \cap H_j^{(r)}$ whose distance from t is a multiple of $\pi_j^{(r)}$ **then**
        check if $S[1, n-t] = S[t+1, n]$ and return t if this is true.