Periodicity in Streams

Funda Ergun, Hossein Jowhari, Mert Sağlam

April 18, 2010

Abstract

In this work we study sublinear space algorithms for detecting periodicity over data streams. A sequence of length n is said to be periodic if it consists of repetitions of a block of length p for some $p \leq \frac{n}{2}$. In the first part of this paper, we give a 1-pass randomized streaming algorithm that uses $O(\log^2 n)$ space and reports the shortest period if the given stream is periodic. At the heart of this result is a 1-pass $O(\log n \log m)$ space streaming pattern matching algorithm. This algorithm uses similar ideas to Porat and Porat's algorithm in FOCS 2009 but it does not need an offline pre-processing stage and is considerably simpler.

In the second part, we study distance to p-periodicity under the Hamming metric, where we estimate the minimum number of character substitutions needed to make a given sequence p-periodic. In streaming terminology, this problem can be described as computing the cascaded aggregate $L_1 \circ F_1^{res(1)}$ over a matrix $A_{p\times d}$ given in column ordering. For this problem, we present a randomized streaming algorithm with approximation factor $2 + \epsilon$ that takes $\tilde{O}(\frac{1}{\epsilon^2})$ space. We also show a $1 + \epsilon$ randomized streaming algorithm which uses $\tilde{O}(\frac{1}{\epsilon^{5.5}} p^{1/2})$ space.

1 Introduction

A sequence, informally speaking, is said to be *periodic* if it consists of repetitions of the same block of characters. In this work we study detecting periodicity over a sequence given as a stream. We present 1-pass randomized algorithms for discovering various periodic properties of a given stream that use sublinear (in most cases polylogarithmic) space and per-character running time.

The study of periodic sequences and patterns has been important in its own right in many fields such as algorithms, data mining, and computational biology. It has also generated fundamental algorithmic tools for solving problems on sequences/strings. In particular, periodicity has been exploited as a central tool in many efficient pattern matching algorithms [AG97]. For instance, the textbook Knuth-Morris-Pratt algorithm [KMP77] computes the periods of all prefixes of the pattern in its pre-processing stage. Periodicity has remained central to almost all pattern matching algorithms to this day.

Formally, a sequence s of length n is said to be p-periodic if $s[i] = s[i+p]$ for all $i = 1, \dots, |s| - p$. The *smallest* $p > 0$ for which s is p-periodic is referred to as *the period* of s. By convention, if the length of the period of s is at most $n/2$, then s is said to be *periodic*, otherwise it is *aperiodic*.

Given the intimate relationship between periodicity and pattern matching, we first investigate sublinear space solutions for finding patterns. Recently Porat and Porat in a breakthrough result presented a polylogarithmic space randomized algorithm for pattern matching that does not require the storage of the entire pattern [PP09]. Briefly, given a pattern u of length m, in an off-line step, they preprocess prefixes $u_1, \dots, u_{\log m}$ where $u_i = u[1, 2^i]$. Then using the Rabin-Karp fingerprinting method [KR87], they build $O(\log n)$-size sketches of all u_i and use them to find occurrences of u in the stream. One idea for computing the period is to use this algorithm but unfortunately due to the offline preprocessing a straightforward adaptation only leads to a k-pass $O(n^{1/k}\text{polylog}(n))$-space algorithm. In order to develop a better technique for finding the period, we start off by developing a more streaming-friendly algorithm for pattern matching. While our solution utilizes ideas similar in essence to those used by [PP09], it does not require an offline preprocessing stage. In fact we show that taking only the Rabin-Karp fingerprints of the prefixes $u_1, \dots, u_{\log d}$ is sufficient to get the same $O(\log n \log m)$ bit space bound. Moreover our pattern matching algorithm enjoys a very clean and simple description.

Now armed with a simple, small-space, streaming pattern matching algorithm, we develop a randomized streaming algorithm for computing the period of s. Our algorithm makes a single pass over s and uses $O(\log^2 n)$ space to find the period of s granted that s is periodic, otherwise it reports that s is aperiodic. The limitation in computing the period for aperiodic sequences (i.e., where the period is longer than half the sequence) turns out to be necessary as we later show a lower bound that computing the period in 1-pass for these sequences requires linear space. On the other hand we show that an additional pass will give us a $O(\log^2 n)$ space solution for periods of any length.

In addition to periodicity, our pattern matching algorithm enables us to get sublinear solutions for frequency moments defined over substrings.

We finally focus on distance to periodicity: we define the distance of s to p-periodicity under the Hamming distance as the minimum number of character substitutions required to make s p-periodic.

$$D_p(s) = \min_{x \text{ is p-periodic}} \{\mathcal{H}(s, x)\}.$$

It turns out that $D_p(s)$ can be expressed as a product-sum of a certain function defined over rows of a matrix $A_{p\times d}$ where $n/p = d$. The problem then is to compute $L_1 \circ F_1^{res(1)}(A) = \sum_{i=1}^{p} F_1^{res(1)}(A_i)$ where A_i is the ith row of A and $F_1^{res(1)}(s)$, known as the *residual tail* of sequence s, equals $|s| - F_\infty(s)$. In general $F_k^{res(r)}(s) = \sum_{i>r}^{m} f_i^k$, where $f_1, \dots, f_m$ are the character frequencies in decreasing order. Note that when $r = 0$ this is the same as F_k, the kth frequency moment of s. While there are space efficient algorithms for approximating $F_1^{res(1)}$ and $F_2^{res(r)}$ [CCF04, GKS05, BCIS09], aggregate computation of $F_1^{res(1)}$ over multiple streams is a new and interesting challenge. In fact this problem can be viewed as a generalization of the Hamming distance to multiple vectors (when $d = 2$, we get the classical Hamming distance) and thus might be of independent interest. For this problem, we present two 1-pass randomized algorithms. The first algorithm approximates $L_1 \circ F_1^{res(1)}$ within $2 + \epsilon$ factor and uses $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$ words of space. The second gives a $1 + \epsilon$ approximation and uses $O(\frac{1}{\epsilon^{5.5}}(p \log p)^{1/2} \log n)$ words of space. For constant alphabet size, the space bound is $O(\frac{1}{\epsilon^3}(p \log p)^{1/2})$.

Related Work The streaming model is well studied; see [M09] for a recent survey. Aside from the implicit implications of [PP09], to our knowledge, our paper is the first to investigate the space complexity of computing the period in the streaming model. In a related direction, Ergun et al. [EMS04] gave an $O(\sqrt{n})$ tester for distinguishing periodic strings from highly aperiodic ones under the Hamming distance in the property testing model. Subsequently Lachish and Newman [LN05] showed a lower bound of $\Omega(\sqrt{n})$ for testing periodicity in the query model. With a focus on time complexity, Czumaj and Gasieniec [CG00] presented an average case analysis for computing the exact period. In a related work, Bar-Yossef et al. [BJKK04] studied the sketching complexity of pattern matching. The work of Indyk et al. [IKM00] focuses on mining periodic patterns and trends in data streams while reading data in large chunks from secondary memory. Numerous studies have been done in the data mining community for detecting periodicity in *time-series databases* and online data (e.g. see [EAE06]), typically with quite different space considerations than in our model. Streaming complexity of cascaded norms $L_k \circ L_p$ over matrices is investigated in depth by Jayram and Woodruff in [JW09], also see [CM05, MW10].

2 Preliminaries

Throughout this paper $[n]$ denotes the set of integers $\{1, \ldots, n\}$. We assume the input stream is a sequence of length n over the alphabet $\Sigma = \{0, 1, \ldots, L\}$. We represent the length of a string s with $|s|$, the ith element of s with $s[i]$, and the substring of s between locations i and j (inclusive) with $s[i, j]$. A d-substring is a substring of length d. The concatenation of two sequences (or vectors) u, v is written as $u \circ v$ and u^i represents the concatenation of i instances of u.

The smallest $p > 0$ for which s is p-periodic, i.e., $s[i] = s[i + p]$ for all $i = 1, \ldots, |s| - p$, is called *the* period of s and is denoted $\mathrm{per}(s)$. The following lemma is folklore.

Lemma 1 *If s is both p-periodic and q-periodic where $p + q \leq |s|$, then s is also $\gcd(p, q)$-periodic.*

We use $M_s(t)$ to denote the set of all positions in s where an exact occurrence of string t starts; i.e., $M_s(t) = \{i \mid s[i, i + |t| - 1] = t\}$. The following lemma, whose proof can be found in Appendix A.1, shows the relation between $\mathrm{per}(t)$ and $M_s(t)$.

Lemma 2 *Let $i \in M_s(t)$ and let $U = M_s(t) \cap [i, i + |t| - 1]$. The following are true.*

i. Let $j \in U$ where $j > i$ and there is no $k \in U$ such that $i < k < j$. If $|i - j| \leq |t|/2$ then $|i - j| = \mathrm{per}(t)$.

ii. There is at most one $j \in U$ such that $|i - j|$ is not a multiple of $\mathrm{per}(t)$. Moreover if $|i - j|$ is not a multiple of $\mathrm{per}(t)$, then $j = \max(U)$.

Fingerprints In Section 3 we use Rabin-Karp fingerprints [KR87], a standard sketching tool which allows us to compare strings of arbitrary length in constant time. Fix an integer alphabet Σ. Let $q > |\Sigma|$ be a prime and $r \in \mathbb{Z}_q^*$ be arbitrary. The Rabin-Karp fingerprint of a string $s \in \Sigma^*$ is defined as

$$\Phi_{q,r}(s) = \sum_{i=1}^{|s|} s[i] \cdot r^{i-1} \pmod{q}$$

The following facts are well-known and the reader is referred to [KR87, PP09] for the proofs.

(P1) $\Phi_{q,r}(s)$ can be computed in one pass over s using $O(\log q)$ bits of space.

(P2) Let $s \neq t$ be two strings and $l = \max(|s|, |t|)$. $\Pr_r[\Phi_{q,r}(s) = \Phi_{q,r}(t)] \leq \frac{l}{q-1}$.

(P3) Given $\Phi_{q,r}(s)$ and $\Phi_{q,r}(t)$, we can obtain $\Phi_{q,r}(st)$ by constant arithmetic operations in $\mathbb{Z}_q$.

(P4) Given $\Phi_{q,r}(st)$ and $\Phi_{q,r}(s)$, we can obtain $\Phi_{q,r}(t)$ by constant arithmetic operations in $\mathbb{Z}_q$.

Henceforth we set $q = \Theta(n^4)$ and assume that r is chosen uniformly at random from $\mathbb{Z}_q^*$ at the beginning of the respective algorithm. We also omit the subscripts and denote the fingerprint of s by $\Phi(s)$.

3 Periodicity and pattern matching

In this section first we show a streaming algorithm for pattern matching and then we present our results for periodicity and frequency moments over substrings.

3.1 The pattern matching algorithm

We assume the input stream $S = u \circ s$ is the concatenation of the pattern u of length m and the text s of length n. Here we present a 1-pass streaming algorithm that *generates* the starting positions of the matches of u in s (equivalently, $M_s(u)$), on the fly using logarithmic space and per-item time. Strictly speaking, if $s[i-m+1, i] = u$, after receiving $s[i]$ our algorithm reports a match with high probability. Also, the probability that our algorithm reports a match where there is no occurrence of u is bounded by n^{-1}.

While it is easy to generate $M_s(u)$ when u is small, the problem is non-trivial for large u. The following lemma implies that given a streaming algorithm that finds length-m patterns, by taking advantage of the Rabin-Karp fingerprints, we can obtain a streaming algorithm for length-cm patterns using only $O(c \log n)$ extra space.

Lemma 3 *Let k be an integer greater than m. Let $\mathcal{A}$ be a 1-pass algorithm that generates $M_s(u)$ using $O(g)$ bits space. Given $\mathcal{A}$ and $\Phi(u)$, there is a 1-pass algorithm that outputs $\Phi(s[i, i+k])$ at position $i+k$ for all $i \in M_s(u)$ using space $O(g + \frac{k}{m}\log n)$ bits.*

PROOF: The algorithm partitions the sequence of positions in $M_s(u)$ (as generated by $\mathcal{A}$) into maximal contiguous subsequences where in each subsequence the distance between consecutive positions is at most $\frac{m}{2}$. To do this we only need to keep track of the last position in $M_s(u)$. If the next position is more than $\frac{m}{2}$ characters apart then we start a new maximal subsequence, otherwise the new position is appended to the last subsequence.

Now let $a_1, a_2, \ldots, a_h \in M_s(u)$ be a maximal sequence of consecutive positions in $M_s(u)$ where $|a_{l+1} - a_l| \leq \frac{1}{2}m$ for all $l \in [h-1]$. We claim that for this sequence we need to maintain at most four fingerprints to generate $\Phi(s[a_l, a_l+k])$ for all $l \in [h]$. To do this, first we launch an individual process to generate $\Phi(s[a_1, a_1+k])$ and $\Phi(s[a_2, a_2+k])$. By Property (P3) from Section 2, this can be done by adding $\Phi(s[a_1, a_1+m-1])$ and $\Phi(s[a_1+m, a_1+k])$. Now if $h < 3$, our claim is proved. So suppose $h \geq 3$.

First we note that by Lemma 1, we should have $|a_{l+1} - a_l| = \mathrm{per}(u)$ for all $l \in [h-1]$. As a result, when we reach the position $a_2 + m - 1$, we have obtained the value of $\mathrm{per}(u)$. Now let $x = u[1, \mathrm{per}(u)]$. We show that it is possible to compute $\Phi(x)$ when we reach $a_3 + m - 1$. To this end, when we are in $a_1 + m - 1$, starting from the next character we build a fingerprint until we reach a_2+m-1. This gives us $\Phi(s[a_1+m, a_2+m-1])$. Note that if $\mathrm{per}(u)$ divides d, then $s[a_1+m, a_2+m-1] = x$ and we are done. Otherwise $s[a_1+m, a_2+m-1]$ is x shifted r times to the left (cyclic shift), where $r = m \pmod{\mathrm{per}(u)}$. Therefore

$$s[a_1+m, a_2+m-1] = x[r+1, \mathrm{per}(u)] \circ x[1, r].$$

Likewise we have $s[a_2+m, a_3+m-1] = x[r+1, \mathrm{per}(u)] \circ x[1, r]$. Therefore at location a_2+m, we know the value of r and $\mathrm{per}(u)$ and consequently using this information, we can build the fingerprints $\Phi(x[r+1, \mathrm{per}(u)])$ and $\Phi(x[1, r])$ when we go over $s[a_2+m, a_3+m-1]$. Note that here we have used the properties (P4) and (P5) from Section 2. It follows that we are able to construct $\Phi(x)$ when we get to a_3+m-1.

Now observe that $s[a_l, a_l+k]$ is equivalent to the substring $s[a_{l-1}, a_{l-1}+k]$ after removing a block of length $\mathrm{per}(u)$ from the left-end of it and adding $s[a_{l-1}+k, a_l-1]$ to the right-end. Therefore we can generate $\Phi(s[a_l, a_l+k])$ by having $\Phi(s[a_{l-1}, a_{l-1}+k])$, $\Phi(s[a_{l-1}+k, a_l-1])$, and $\Phi(x)$. This proves our claim.

It should be clear that at each point in time, we run at most $\frac{4k}{m}$ parallel fingerprint computations. Each fingerprint takes $O(\log n)$ space. This finishes the proof of the lemma. □

Our pattern matching algorithm is the result of a recursive application of Lemma 3. First as we go over u, we build $\Phi(u[1, 2^i])$ for all $i \in [\log m]$. By Property (P3) this can be done in 1-pass and using $O(\log m \log n)$ bits of space. Let $\mathcal{A}_i$ be an algorithm that generates $M_s(u[1, 2^i])$ in space g_i. When $i < c$ where c is a small constant, we can use the naive solution of storing the entire pattern which gives $g_i = O(\log n)$. By Lemma 3, we get an algorithm $\mathcal{A}_{i+1}$ for $M_s(u[1, 2^{i+1}])$ in space $O(g_i + \log n)$ by fingerprint comparisons. Applying this $O(\log |u|)$ times we obtain an algorithm for $M_s(u)$ using space $O(\log |u| \log n)$ bits. The success probability is at least $1 - \log m/n^2$ and this is due to the Property (P4) in Section 2 and the observation that we make at most $O(n \log |u|)$ fingerprint comparisons.

Theorem 4 *There is a 1-pass streaming algorithm that generates $M_s(u)$ in $O(\log |u| \log n)$ bits of space and $O(\log |u|)$ per-item processing time. The error probability is bounded by n^{-1}.*

Since our pattern matching algorithm only requires the fingerprints of a small set of prefixes of the pattern, it can be used to generate $M_s(s[1, m])$ (where the pattern itself is a prefix of the text) in one pass and in space $O(\log m \log n)$ bits. This property of our algorithm will be essential in Section 3.3. Furthermore, in addition to $M_s(u)$, our algorithm generates $M_s(u[1, 2^i])$ for each $i = 1, \ldots, \log m$, which leads to further space economy in our algorithms in the next section. On the other hand, we note that any algorithm that generates matches for $\log m$ prefixes of the pattern must necessarily use $\Omega(\log m \log n)$ space (see Appendix A.4).

3.2 Finding the period

In this section we describe an algorithm for testing periodicity and finding the period. First we start with a simple solution that gives a weaker bound and proceed to the general case afterwards. To make the presentation simpler, we assume that n is power of 2 (this assumption can be discarded by minor modifications to the parameters).

Testing whether the string s is periodic or not is equivalent to testing if there is a suffix $s[t, n]$ of size at least $n/2$ that matches a prefix of s. Clearly in this case $s[1, n/2]$ would be a prefix of $s[t, n]$. Defining

$$T = M_s(s[1, n/2]),$$

we can say that s is periodic if there exists $i \in T$ where $s[i+1, n] = s[1, n-i]$. Now if $i \leq n/4$, we can build both $\Phi(s[i+1, n])$ and $\Phi(s[1, n-i])$ in one pass over s and thus we can test whether $\mathrm{per}(s) \leq n/4$ or not as follows.

Run the pattern matching algorithm to find $i = \min(T \cap [1, n/4])$. Build $\Phi(s[i+1, n])$ and $\Phi(s[1, n-i])$. If $\Phi(s[i+1, n]) = \Phi(s[1, n-i])$ then $\mathrm{per}(s) = i$ otherwise output that $\mathrm{per}(s) > n/4$.

The reason that we only perform the test for $\min(T \cap [1, n/4])$ is a consequence of Lemma 2. We do not need to check whether $s[i+1, n] = s[1, n-i]$ for $i = c \min(T)$ when c is an integer greater than 1. This is because, in this case $s[1, i]$ would be of the form $u \circ \ldots \circ u$ (a cyclic string) and thus can not be the period of s. From these observations we get the following lemma.

Lemma 5 *There is a 1-pass streaming algorithm that decides whether* $\mathrm{per}(s) \leq n/4$ *or not in space* $O(\log^2 n)$ *bits. The algorithm also outputs the exact period if* $\mathrm{per}(s) \leq n/4$.

For $i \in T$ where $i > n/4$, checking whether $s[i+1, n] = s[1, n-i]$ is not straightforward. This is because when we find out that $i \in M_s(s[1, n/2])$, we have already crossed the point $n - i$ and lost the opportunity to build $\Phi(s[1, n-i])$. To solve this problem we conservatively maintain a superset of T and prune it as we learn more about the input stream. First observe that, for $i \in T$, it is enough to build $\Phi(s[n/2+1, n-i])$. This is because $s[1, n-i] = s[1, n/2] \circ s[n/2+1, n-i]$. Now for $i \in [1, n/2]$, let $s_i = s[n/2+1, n-i]$. At each point in time, we maintain a dynamic set of positions R that will contain T and for each $i \in R$ we collect enough information to be able to construct $\Phi(s_i)$. Also in parallel we run a pattern matching process to generate T. Finally for each position in $\{i \in R \cap T \mid i \neq c \min(T) \text{ for } c \in \mathbb{N}\}$ we check whether $\Phi(s[i+1, n]) = \Phi(s[1, n-i])$. If $\Phi(s[i+1, n]) = \Phi(s[1, n-i])$ holds in one case, then we declare s to be periodic otherwise it is reported aperiodic.

Let $H = H_1 \cup H_2 \cup \ldots \cup H_{\log(n/4)}$ where H_k is defined as follows. Let $I_k = [n/2 - 2^k + 1, n/2 - 2^{k-1}]$ and $H_k = M_s(s[1, 2^k]) \cap I_k$. In other words, H_k is the positions of all occurrences of $s[1, 2^k]$ that start within the interval I_k. We have chosen the boundaries of the I_k so that there are few of them and also their union covers T: we have $T \subseteq H$. More importantly, to process each interval we only use logarithmic space.

In what follows, for a fixed k we show how to compute $R_k \subseteq H_k$ and more importantly how to maintain $\Phi(s_i)$ for each $i \in R_k$. Also we guarantee that every member of T will be added to $R = R_1 \cup \ldots \cup R_{\log(n/4)}$ at some point. Initially all R_k are empty. First we distinguish two main cases. In both cases, we use the pattern matching algorithm described in Section 3 to get the sequence of positions in H. Also, when we detect $i \in H_k$, we add it to R_k. However we might prune R_k and remove some unnecessary elements. In the following let $p = \mathrm{per}(s[1, 2^k])$.

In the first case we have $p > \frac{1}{4}2^k$. By Lemma 2, we get $|H_k| < 4$. Moreover we detect $i \in H_k$ before reaching the end of s_i and thus we can build $\Phi(s_i)$ at the right time. In this case we let $R_k = H_k$. Clearly we can maintain R and the associated fingerprints in $O(\log n)$ space.

The case where $p \leq \frac{1}{4}2^k$ is a bit more complicated. First we consider that, by Lemma 1, the positions in H_k have a succinct representation as the distance between each consecutive pair of positions is p. It follows that we can store R_k in $O(\log n)$ space. However, in this case H_k could be large and if we maintain $\Phi(s_i)$ for each $i \in H_k$ individually, this might take linear space. To solve this problem, we take advantage of the periodic structure of $s[1, 2^k]$ and possibly the substring $s[2^k + 1, 2^{k+1}]$. Consider that for $i \in H_k$, s_i is a substring of $s[i, i + 2^{k+1} - 1]$. Now (informally) if the substrings $\{s_i\}$ fall in a periodic region, we can maintain all $\Phi(s_i)$ by saving a constant number of fingerprints. On the other hand, if the substring $s[i, i + 2^{k+1} - 1]$ is not periodic then we use the period information of $s[1, 2^{k+1}]$ to prune R_k. To do this, we collect the following information when we process the first half of the stream.

- Using the tester from Lemma 5, we compute p. If it is reported that $p > \frac{1}{4}2^k$, then I_k falls into the previous case. We also compute $\Phi(s[1, p])$ and $\Phi(s[2^k - p + 1, 2^k])$.
- Let $u_1 \circ u_2 \circ \ldots \circ u_t \circ u'$ be a decomposition of $s[2^k + 1, 2^{k+1}]$ into consecutive blocks of length p except possibly for the last block. Let x to be the maximum j such that $s[1, 2^k] \circ u_1 \circ \ldots \circ u_j$ is p-periodic. We compute x.

Now let $b_1, b_2, \ldots, b_r$ be the elements of H_k in increasing order. Since $|I_k| \leq \frac{1}{2}2^k$, we have $|b_{i+1} - b_i| = p$ for all $i \in [r-1]$. Let $v_1 \circ v_2 \circ \ldots \circ v_l \circ v'$ be a decomposition of the substring $s[b_r + 2^k, n/2 + 2^k]$ into consecutive blocks of length p except possibly the last block (see Figure 1 for a pictorial presentation of the substrings). Note that the right endpoint of s_i for $i \in H_k$ is located within $[b_r + 2^k, n/2 + 2^k]$. Now let y be the maximum j such that $s[b_r, b_r + 2^k - 1] \circ v_1 \circ \ldots \circ v_j$ is p-periodic. We consider two cases. If $y = l$ then

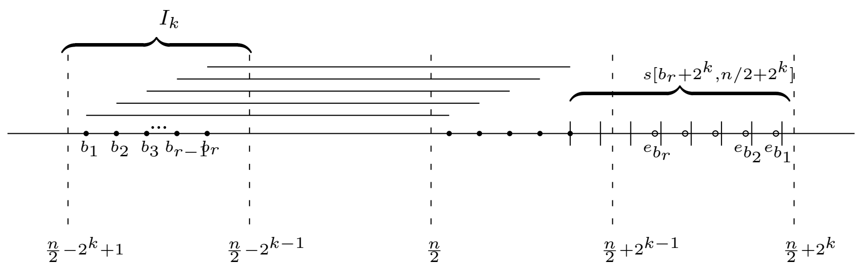

Figure 1: A sample run of the algorithm in Section 3.2.

$\{s_i \mid i \in H_k\}$ are substrings of a periodic interval. Let e_{b_1} be the right endpoint of s_{b_1}, i.e. $e_{b_1} = n - b_1$. Note that we have $e_{b_1} > e_{b_2} > \ldots > e_{b_r}$. In this case, all the following substrings (except possibly the last one) are equal: $s[e_{b_r}+1, e_{b_{r-1}}], s[e_{b_{r-1}}+1, e_{b_{r-2}}], \ldots, s[e_{b_2}+1, e_{b_1}]$. Therefore to compute $\Phi(s_{b_j})$, we just need to maintain $\Phi(s_{b_1})$ and $\Phi(s[e_{b_2}+1, e_{b_1}])$. We compute $\Phi(s_{b_1})$ individually. So in this case $R_k = H_k$. In the other case, we have $y < l$. We make the following claim.

Claim 6 *If $y < l$ and $|r-j| + y \neq x$ then $b_j \notin T$.*

PROOF: If $b_j \in T$ then $s[b_j, b_j + 2^{k+1} - 1] = s[1, 2^{k+1} - 1]$ and the claim follows from the definition of x and y. □

The main consequence of Claim 6 is that at most one member of H_k is a member of T. Given this, we can ignore the rest and just maintain $\Phi(s_{b_j})$ when $|j-r| + y = x$ and $\Phi(s_{b_1})$. So in this case $|R| = 2$.

It remains to state how to compute x and y. To compute x, we need to know p and $\Phi(s[2^k - p + 1, 2^k])$. This information can be obtained in one pass (see the observations before Lemma 5). Computation of y is similar to x. In this case, before reaching the block v_{y+1}, we maintain the fingerprint of the s_i (that we passed so far) using the periodic structure we described in the previous case.

Given the above discussion, for each $k \in \{1, 2, \ldots, \log(n/4)\}$, we need to keep $O(1)$ number of fingerprints to maintain R_k and its associated fingerprints which makes the total space $O(\log^2 n)$ bits. Hence, we get the following result.

Theorem 7 *There is a 1-pass randomized streaming algorithm that given $s \in \Sigma^n$ outputs $\mathrm{per}(s)$ if s is periodic, otherwise it reports that s is aperiodic. The algorithm uses $O(\log^2 n)$ bits of space and has $O(\log n)$ per-item running time. The error probability is at most $O(n^{-1})$.*

The following theorem shows that in general finding the period in one pass requires linear space. With one additional pass however, the period of an arbitrary string can be found in $O(\log^2 n)$ space, as we show in Appendix A.2.

Theorem 8 *Every 1-pass randomized algorithm that computes $\mathrm{per}(s)$ requires $\Omega(n)$ space.*

PROOF: Consider the communication game between Alice and Bob, respectively holding strings a and b, both of length n, where the goal of the game is to compute $\mathrm{per}(a \circ b)$. We show that any one-way protocol that computes $\mathrm{per}(a \circ b)$ requires $\Omega(n)$ communication by a reduction from the augmented indexing problem (see Appendix A.3). Suppose Alice and Bob are given an instance of IND_2^n as follows. Alice gets an $x \in \{0,1\}^n$, and Bob gets an index $i \in [n-1]$ and $y \in \{0,1\}^i$ with the promise that $y = x[1, i-1]$. Alice sets $a = x$ and Bob sets $b = w \circ y \circ 1$, where w is $n - i$ repetitions of the binary negation of $y[1]$. One can show that $\mathrm{per}(a \circ b) = i$ if and only if $x[i+1] = 1$ w.h.p. This proves our theorem. □

3.3 Frequency moments over substrings

Let s be a string of length n, and $k \geq 0$, $d \leq n$ be integers. We define the kth frequency moment of d-substrings of s as

$$F_{k,d}(s) = \sum_{u \in \Sigma^d} |M_s(u)|^k.$$

To approximate $F_{k,d}$, one can create a fingerprint for each d-substring and feed this stream of fingerprints to a standard F_k algorithm. Thus, using the algorithms of [IW05, BGKS06, I06, KNW10] one can $(1+\epsilon)$-approximate $F_{k,d}$ with $\tilde{O}(d + n^{1-2/k})$ space and $\tilde{O}(1)$ per item processing time for any $k \geq 0$. It is not possible to obtain a $o(d)$ algorithm however, if we insist on constructing a fingerprint for each d-substring[1]. We note that by replacing the reservoir sampling procedure of [AMS96] with the pattern matching algorithm above, one can $(1+\epsilon)$-approximate $F_{k,d}$ using space $\tilde{O}(\frac{1}{\epsilon^2} n^{1-1/k})$, in particular independent of d.

Unfortunately, the estimator of [AMS96] does not give a bound for $F_{0,d}$ which is perhaps the most commonly used moment for substrings, also known as the q-gram measure. Here we present an $\tilde{O}(\frac{1}{\epsilon}\sqrt{n})$ space randomized algorithm that $(1+\epsilon)$-approximates $F_{0,d}$.

Theorem 9 *There exists a 1-pass streaming algorithm that $(1+\epsilon)$-approximates $F_{0,d}$ using $\tilde{O}(\frac{1}{\epsilon}\sqrt{n})$ space.*

PROOF: Let $s \in \Sigma^n$ be the stream. Let K be the set of all d-substrings of s and $n' = n - d + 1$. Our basic estimator X is defined as follows. Let i be random position between 1 and n'. We set $X = 0$ if there exists a $j > i$ such that $s[i, i+d-1] = s[j, j+d-1]$, we set $X = n'$ otherwise. We have $\mathbb{E}[X] = \frac{1}{n'} \sum_{w \in K} n' = F_{0,d}$. Also, $\mathrm{Var}(X) \leq \mathbb{E}[X^2] = \frac{1}{n'} \sum_{w \in K} n'^2 \leq n \cdot F_{0,d}$. Let Y be the average of $\frac{3}{\epsilon}\sqrt{n}$ repetitions of X. By Chebyshev's inequality,

$$\Pr[|Y - F_{0,d}| \geq \epsilon F_{0,d}] \leq \frac{\mathrm{Var}(Y)}{\epsilon^2 F_{0,d}^2} \leq \frac{\sqrt{n}}{3\epsilon F_{0,d}}.$$

Right hand side is smaller than $1/3$ when $F_{0,d} \geq \frac{1}{\epsilon}\sqrt{n}$. Note that we can compute each X in $O(\log n \log d)$ space in one pass using the pattern matching algorithm of Section 3.1. In Lemma 19 (see Appendix A.1) we show that $F_{0,d}$ can be calculated exactly, in space $\tilde{O}(F_{0,d})$. Therefore we compute $\frac{3}{\epsilon}\sqrt{n}$ estimates for X, while we run the exact algorithm in parallel. If at any point in the stream the exact algorithm detects that $F_{0,d} \geq \frac{1}{\epsilon}\sqrt{n}$ we terminate it and output the sampling estimate, otherwise we output the value computed by the exact algorithm. $\square$

4 Approximating the distance to periodicity

Recall that $D_p(s)$ is the minimum number of character changes on $s \in \Sigma^n$ to make it p-periodic. Assume WLOG that p divides n where $n = dp$, and view s as a $p \times d$ matrix A where $A(i,j) = s[(i-1)p + j]$. If p does not divide n, s can be represented by two matrices. Then, $D_p(s)$ is the the minimum number of substitutions in A to make every row consist of d repetitions of the same character. Also, $D_p(s) = L_1 \circ F_1^{res(1)}(A) = \sum_{i=1}^{p} F_1^{res(1)}(A_i)$. It is challenging to compute this quantity since we receive A in the column order: $A(1,1), \ldots, A(p,1), A(1,2), \ldots, A(p,2), \ldots$ To compute $L_1 \circ F_1^{res(1)}(A)$ exactly, one can compute the residual tail of each row in parallel using independent counters, in $O(|\Sigma| p)$ words of space. On the other hand, one can estimate $F_i^{res(1)}(A_i)$ within $1-\epsilon$ factor in $O(1/\epsilon)$ words of space in several ways. For instance, using the Heavy Hitters algorithms in [MG82, BKMT03] we can approximate $F_\infty(A_i)$ with additive error $\epsilon F_1^{res(1)}(A_i)$, giving the following bound.

[1]An easy information theoretic observation shows that sliding a fingerprint for d-substrings that preserves equality with high probability requires $\Omega(d)$ space.

Theorem 10 *There is a deterministic streaming algorithm that approximates $L_1 \circ F_1^{res(1)}(A)$ within $1-\epsilon$ factor using $O(\frac{p}{\epsilon})$ words of space.*

Now we turn our attention to randomized algorithms. In the following, let $F(A)$ denote $L_1 \circ F_1^{res(1)}(A)$.

4.1 A $(2+\epsilon)$ algorithm

The idea of this algorithm is to reduce $F(A)$ to L_0 of a vector where each item in s represents a set of updates to this vector. Let $f_i(a)$ be the number of occurrences of $a \in [m]$ in A_i. We first observe the following.

Fact 11 $F_1^{res(1)}(A_i) \geq \frac{1}{d}\sum_{a<b} f_i(a)f_i(b) \geq \frac{1}{2}F_1^{res(1)}(A_i)$.

PROOF: Notice that $\frac{1}{d}\sum_{a<b} f_i(a)f_i(b) = \frac{1}{2}(d - \frac{1}{d}\sum_a f_i^2(a))$. Clearly $\frac{1}{d}\sum_a f_i^2(a) \leq \max\{f_i(a)\}$. This proves the right hand side inequality. To prove the left inequality, we need to show $d \geq 2\max\{f_i(a)\} - \frac{1}{d}\sum_a f_i^2(a)$. This is true because the RHS is maximized when $\max\{f_i(a)\} = d$. $\square$

One way to produce $\sum_{a<b} f_i(a)f_i(b)$ is to compare each location of A_i with all other locations and sum up the mismatches. To express this in terms of L_0, let v_i be an all zero vector of length d^2 with a coordinate for each $(j,k) \in [d]\times[d]$. Given $A_i(j) = l$, add l to $v_i(j,k)$ and subtract l from $v_i(k,j)$ for all $k \in [d]$. Then, $L_0(v_i) = 2\sum_{a<b} f_i(a)f_i(b)$. We generate the updates to vector $v = v_1 \circ \ldots \circ v_p$ as we go over A and estimate L_0 using the following result by Kane et al. [KNW10].

Theorem 12 *[KNW10] Let $x = (x_1, \ldots, x_n)$ be an initially zero vector. Let the input stream be a sequence of t updates to the coordinates of x of the form (i,u) where $u \in \{-M, \ldots, M\}$ for an integer M and i is an index. There is a 1-pass streaming algorithm for $(1+\epsilon)$-approximating $L_0(x)$ using space $O(1/\epsilon^2 \log n(\log(1/\epsilon) + \log\log(tM)))$, with success probability 7/8, and with $O(1)$ per-item processing time.*

By Theorem 12 and Fact 11, we get a $2+\epsilon$ approximation for $F(A)$ in space $O(1/\epsilon^2 \log(1/\epsilon)\log(n))$ bits. However, per-item processing time is $\Omega(d)$. To overcome this, we pick a random set S of coordinates from $[d]$ of size $O(\frac{1}{\epsilon^2}\log p)$ and align $A_i(j)$ with entire A_i only when $j \in S$, obtaining a new vector v_i' with dimension $d|S|$. Now fix an i and consider random variable $L_0(v_i')$. Let Y_j be an indicator random variable which is 1 iff $j \in S$. We have $\mathbb{E}[L_0(v_i')] = \sum_{j=1}^{d}\mathbb{E}[Y_j]\sum_{k=1}^{d}\mathcal{H}(A_i(j), A_i(k)) = \frac{2|S|}{d}\sum_{a<b} f_i(a)f_i(b)$. Since $\{Y_j\}$ are independent, using Chernoff bounds,

$$\Pr\left[|L_0(v_i') - \mathbb{E}[L_0(v_i')]| > \epsilon\,\mathbb{E}[L_0(v_i')]\right] \leq \frac{1}{8p}.$$

By the union bound, the probability that $\frac{1}{2|S|}L_0(v')$ is away from $\frac{1}{d}\sum_{i=1}^{p}\sum_{a<b} f_i(a)f_i(b)$ by a factor of ϵ is at most $1/8$. Given this and the fact that the underlying L_0 estimation itself gives a $1+\epsilon$ approximation we get a $(1+\epsilon)^2 = 1+\theta(\epsilon)$ approximation using polylogarithmic space and $O(1/\epsilon^2 \log p)$ per-item processing time.

Theorem 13 *Let $\epsilon > 0$. There is a 1-pass randomized streaming algorithm that approximates $L_1 \circ F_1^{res(1)}(A)$ within $2+\epsilon$ factor using $O(1/\epsilon^2 \log(1/\epsilon))$ words of space. The error probability is at most $1/4$.*

4.2 A $(1+\epsilon)$ algorithm

To find a better estimate for $F(A)$ we use a combination of naive sampling and exact sparse recovery. If $F(A)$ is high, naive sampling gives us a good estimate. If $F(A)$ is low, then A has few non-uniform rows (we call A_i non-uniform if $F_1^{res(1)}(A_i) > 0$) and in space roughly proportional to the number of non-uniform rows, we can use sparse recovery to find all non-uniform rows with high probability. In the latter case, we obtain $F(A)$

exactly, or with a large alphabet, to within $1+\epsilon$ factor. A generic implementation of this gives a $\tilde{O}(n^{1/2})$ space solution, where $n = dp$. Below we describe a $\tilde{O}(p^{1/2})$ space algorithm which is in line with this this approach but uses a combination of sampling, exact sparse recovery, and the $2+\epsilon$ algorithm described earlier.

Let $F'(A_i) = 1/d \sum_{a<b} f_i(a) f_i(b)$. Recall that in the previous algorithm we used $F'(A_i)$ as an approximation for $F_1^{res(1)}(A_i)$. The worst case for this approximation happens when $F_1^{res(1)}(A_i)$ is maximized, i.e., $F_\infty(A_i) = d/F_0(A_i)$. On the other hand, when $F_1^{res(1)}(A_i)$ is low, the above quantity gives us a good estimate. This is because $F'(A_i)$ is lowerbounded by $\frac{1}{d}(d - F_\infty(A_i))F_\infty(A_i)$ which implies the following.

Fact 14 *Let $\epsilon \geq 0$. Suppose $F_1^{res(1)}(A_i) \leq \epsilon d$. We have $F'(A_i) \geq (1-\epsilon)F_1^{res(1)}(A_i)$.*

Define $F'(A) = \sum_{i=1}^{p} F'(A_i)$. From the definitions, we get

$$F'(A) + \frac{1}{2d}\sum_{i=1}^{p}((F_1^{res(1)}(A_i))^2 + F_2^{res(1)}(A_i)) = F(A). \tag{1}$$

Now let $F''(A_i) = \frac{1}{2d}((F_1^{res(1)}(A_i))^2 + F_2^{res(1)}(A_i))$. From (1) it follows that if we are given $F''(A) = \sum_{i=1}^{p} F''(A_i)$, by using the algorithm from the previous section, we get a $1+\epsilon$ approximation for $F(A)$. On the other hand, Fact 14 tells us that we only need to compute $F''(A_i)$ for rows with high contribution. For $t \leq d$ define H_t to be the set $\{j \mid F_1^{res(1)}(A_j) \geq t\}$. The following is a consequence of Fact 14 and (1).

$$F(A) \geq F'(A) + \sum_{i \in H_{\epsilon d}} F''(A_i) \geq (1-\epsilon)F(A). \tag{2}$$

In our algorithm we do not compute $F''(A_i)$ for $H_{\epsilon d}$ but approximate them with error proportional to $F(A_i)$. This is achieved by sampling a few columns from A and using a sparse recovery procedure to find non-uniform rows in the sampled matrix. For our sparse recovery procedure, we use the following result from [LP07].

Theorem 15 *[LP07] Let $x, y \in \Sigma^n$. There is a randomized 1-pass streaming algorithm that, given the coordinates of x and y in arbitrary order, can check if $\mathcal{H}(x,y) > r$ or not using $O(r(\log n + \log|\Sigma|))$ bits of space and $O(\log n)$ per-item time. Moreover in case $\mathcal{H}(x,y) \leq r$, the algorithm finds all pairs $(x[i], y[i])$ where $x[i] \neq y[i]$. The probability of error is at most n^{-1}.*

Now we are ready to describe our algorithm. Let ϵ be an arbitrary constant smaller than 1. Let $\delta < \epsilon$ (we determine the value of δ later) and $k \geq \frac{8 \log n}{\delta^2}$. For $r \leq p$, denote by $\mathrm{SR}(r)$ the exact sparse recovery algorithm from Theorem 15. We run the following three threads in parallel.

T1 Run the $2+\epsilon$-approximation algorithm from Section 4.1. Let t_1 be the output.

T2 Let $K \geq 8k(\frac{p \log p}{\epsilon})^{1/2}$. Let B be K sampled rows of A (picked uniformly and independently). Compute a $1-\epsilon$ approximation of $F(B)$ using the algorithm from Theorem 10. Let t' be the answer. Let $t_2 = \frac{pt'}{K}$.

T3 Let $r_0 > 4 \log p$ be an odd integer. Run the following r_0 times in parallel.

In run j, let $r = \frac{8}{\epsilon^{1.5}}(\frac{p}{\log p})^{1/2}$. Sample k columns of A uniformly and independently, obtaining matrix C. Run $\mathrm{SR}(r)$ over consecutive columns in C. If more than r non-uniform rows are detected, abort the run. Otherwise for each non-uniform row C_i do the following. Let $f_{C_i}(a)$ be the frequency of a in C_i. Use $f'_i(a) = \frac{d}{k} f_{C_i}(a)$ to estimate $f_i(a)$. Let A'_i be a sequence corresponding to the frequency vector f'_i. Compute $X_i = \frac{d^2}{2\binom{k}{2}} \sum_a f_{C_i}(a)(f_{C_i}(a) - 1)$ and let $Y_i = \frac{1}{2d}(F_1^{res(1)}(A'_i))^2 + \frac{1}{2d}(X_i - F_\infty^2(A'_i))$. At the end we let $G_j = \{(i, Y_i) \mid F_1^{res}(A'_i) \geq \epsilon d\}$.

In the end, if the majority of the runs have aborted, the algorithm outputs t_2. Otherwise, WLOG, assume the first $l > 2\log n$ runs have survived. Let G be the set of pairs $(i, g(A_i))$, where i appears in all $G_1, \ldots, G_l$ and $g(A_i)$ is the median of Y_i's produced by the surviving runs. Then we output $t_3 = t_1 + \sum_{i \in G} g(A_i)$.

Lemma 16 *Assuming p is greater than a large enough constant, the above algorithm gives a $1 \pm 3\epsilon$ approximation for $F(A)$ with probability is at least $3/4$.*

PROOF: We first consider the case when we ignore T3 and take the answer of T2. For each aborting run we have $F(C) > r$. Based on the observation that $\mathbb{E}[\frac{d}{k}F(C)] \leq F(A)$, and by Markov inequality, we have $\Pr[\frac{d}{k}F(C) > 8F(A)] < 1/8$. Since in this case more than half of the runs have aborted, using Chernoff bound, with probability at least $1 - 1/p^2$ we have $F(A) \geq \frac{rd}{8k}$. On the other hand, by Chebyshev's bound, we have $\Pr[|t_2 - F(A)| \geq 2\epsilon F(A)] < pd/(\epsilon^2 F(A)K)$. Assuming the event $F(A) \geq \frac{rd}{8k}$ and after plugging the values of r and K, we get that the probability is bounded by $1/8 + 1/p^2$.

Now consider the case where the output is t_3. In this case, we need to analyze the quality of the approximation of $F''(A_i)$ produced by a fixed run. The below claim follows by Chernoff bounds.

Claim 17 *For $a \in \Sigma$, with probability at least $1 - \frac{1}{8n^2}$, $|f'_i(a) - f_i(a)| \leq \delta d$.*

From Claim 17 it follows that, with probability at least $1 - 1/(8np)$, the error of the first term in Y_i, i.e., $\frac{1}{2d}(F_1^{res(1)}(A'_i))^2$, is bounded by $2\delta d$. To bound the error of the second term in Y_i, we use Chebyshev bound and the variance analysis of [BKS01] (cf. Lemma 5.3) to estimate F_2. From [BKS01], we have $\mathbb{E}[X_i] = F_2(A_i)$ and $\mathrm{Var}(X_i) \leq \frac{d}{k}(F_2(A_i))^{3/2}$. Using Chebyshev's inequality, we get

$$\Pr[|X_i - F_2(A_i)| > \delta d^2] \leq \frac{(F_2(A_i))^{3/2}}{\delta^2 k d^3}.$$

Given that $k > \frac{8}{\delta^2}\log n$, this probability is bounded by $1/(8\log n)$. Therefore, with probability at least $1 - 1/(8\log n)$, the second term of Y_i has error at most $3\delta d$. Since we took the median of at least $2\log p$ outcomes, with probability at least $1 - 1/(p^2 \log n)$, for $i \in G$, we have $|g(A_i) - F''(A_i)| < 5\delta d$. Also with probability at least $1 - (\log p)/(2n)$, we have

$$H_{(\epsilon+\delta)d} \subseteq G, \quad ([p] \setminus H_{(\epsilon-\delta)d}) \cap G = \emptyset \tag{3}$$

Now we choose δ so that $5\delta d \leq \epsilon(\epsilon - \delta)d$. This gives us $\delta = O(\epsilon^2)$ and now we guarantee that, for all $i \in G$, $g(A_i)$ is away from $F''(A_i)$ by at most $\epsilon F_1^{res}(A_i)$. Putting these observations and (1),(2), and (3) together we get $|t_3 - F(A)| \leq 3\epsilon F(A)$. This proves our lemma. $\square$

Threads T2 and T3 dominate our space complexity. The sampling algorithm in T2 takes $O(\frac{1}{\epsilon}K)$ space with $O(1)$ time per item. The runs in T3 take $O(r_0 rk)$ space in total. However since the decoding time of the sparse recovery is $O(r\log n)$, this makes the worst-case per-item time $O(r_0 r \log^2 n)$. Since $\delta = O(\epsilon^2)$, our final space bound becomes $O(1/\epsilon^{5.5}(p\log p)^{1/2}\log n)$. Note that with a consant alphabet, eliminating the repetitions in T3 and choosing parameters differently, we can get $O(1/\epsilon^3(p\log p)^{1/2})$ space.

Theorem 18 *There is a randomized 1-pass streaming algorithm that outputs a $1 \pm \epsilon$ approximation of $L_1 \circ F_1^{res}(A_{p\times d})$ with probability at least $3/4$ using $O(1/\epsilon^{5.5}(p\log p)^{1/2}\log n)$ words of space.*

References

[AMS96] N. Alon. Y. Matias and M. Szegedy. Space complexity of approximating the frequency moments. *STOC 1996.*

[AG97] A. Apostolico, Z. Galil. Pattern matching algorithms. *Oxford University Press*. 1997

[BKS01] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar. Sampling algorithms: lower bounds and applications. *CCC 2002.*

[BJKS02] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar. Information theory methods in communication complexity. *STOC 2001.*

[BJKK04] Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, R. Kumar. Sketching complexity of pattern matching. *RANDOM 2004.*

[BCIS09] R. Berinde, G. Cormode, P. Indyk, and M. Strauss. Space-optimal heavy hitters with strong error bounds. *PODS 2009.*

[BGKS06] L. Bhuvanagiri, S. Ganguly, D. Kesh, C. Saha. Simpler algorithm for estimating frequency moments of data streams. *SODA 2006.*

[BKMT03] P. Bose, E. Kranakis, P. Morin, and Y. Tang. Bounds for frequency estimation of packet streams. *Proceedings of the 10th International Colloquium on Structural Information and Communication Complexity*. 2003.

[CCF04] M. Charikar, K. Chen, M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.* 312(1): 3-15 (2004).

[CM05] G. Cormode, S. Muthukrishnan. Space efficient mining of multigraph streams. *PODS 2005.* 271-282

[CT91] T.M. Cover and J.A. Thomas. Elements of information theory. *John Wiley & Sons Inc.* 1991.

[CG00] A. Czumaj, L. Gasieniec. On the complexity of determining the period of a string. *CPM 2000.*

[EAE06] M. G. Elfeky, W. G. Aref, A. K. Elmagarmid. STAGGER: periodicity mining of data streams using expanding sliding windows *ICDM 2006.*

[EMS04] F. Ergun, S. Muthukrishnan, C. Sahinalp. Sublinear methods for detecting periodic trends in data streams. *LATIN 2004.*

[GKS05] S. Ganguly, D. Kesh, C. Saha, Practical algorithms for tracking database join sizes. *FSTTCS 2005.*

[I06] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM,* 53(3): 307-323 (2006).

[IKM00] P. Indyk, N. Koudas and S. Muthukrishnan. Identifying representative trends in massive time series datasets using sketches. *VLDB 2000.*

[IW05] P. Indyk, D. Woodruff. Optimal approximations of the frequency moments of data streams. *STOC 2005.*

[JW09] T.S. Jayram, D. Woodruff. The data stream space complexity of cascaded norms. *FOCS 2009.*

[KNW10] D. M. Kane, J. Nelson, D. Woodruff. An optimal algorithm for the distinct elements problem. *PODS 2010.*

[KR87] R.M. Karp and M.O. Rabin. Efficient randomized pattern matching algorithms. *IBM Journal of Res. and Dev.*, p 249:260, 1987.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.* 6:323–350, 1977.

[LN05] O. Lachish and I. Newman. Testing periodicity. *APPROX-RANDOM 2005.*

[LZ77] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory.* Vol. 23 (1977), pp. 337-343.

[LP07] O. Lipsky, E. Porat. Improved sketching of hamming distance with error correcting. *CPM 2007.*

[M09] S. Muthukrishnan. Data stream algorithms. *The Barbados Workshop on Computational Complexity.* 2009.

[MG82] J. Misra and D. Gries. Finding repeated elements. *Technical Report, Cornell University.* 1982.

[MW10] M. Monemizadeh, D. Woodruff. 1-Pass relative-error Lp-sampling with applications. *SODA 2010.*

[PP09] B. Porat, E. Porat. Exact and approximate pattern matching in the streaming model. *FOCS 2009.*

A Appendix

A.1 Missing proofs

PROOF: (Lemma 2) First we prove claim *(i)*. Let $p = \operatorname{per}(t)$. By the definition of period, t is $|i-j|$-periodic. This implies that $p \leq |i-j|$. Suppose $p < |i-j|$. We prove that there exists a $k \in M_s(t)$ where $i < k < j$. Since $|i-j| \leq 1/2|t|$, by Lemma 1, we get that t is gcd$(p, |i-j|)$-periodic and thus $|i-j|$ is a multiple of p. This means that all the consecutive blocks

$$s[i, i+p-1], s[i+p, i+2p-1], \ldots, s[j-p, j-1], s[j, j+p-1]$$

are equal. Take $k = j-p$. Clearly $k \in M_s(t)$. This contradicts with our assumption and proves the first claim.

To prove the second claim, we proceed as follows. Let $|t| = lp + r$, where l is an integer and $0 \leq r < p$. Define $i_0 = i + |t| - r - p$. If $j \in U$ and $j < i_0$, then $|j-i|$ is a multiple of p by applying Lemma 1 twice. Suppose for contradiction that there are $j_1 < j_2$ in U such that both $|j_1 - i|$ and $|j_2 - i|$ are indivisible by p. From the previous sentence $j_1 \geq i_0$. Also, by definition of period $|j_1 - j_2| \geq p$. Let $s_1 = s[i_0, i + |t| - 1]$. Since s_1 is both $|j_1 - i_0|$- and p-periodic and $|j_1 - i_0| + p \leq |s_1|$, by Lemma 1 s_1 is $\gcd(|j_1 - i_0|, p)$-periodic. In particular, $s[i, i+p-1] = s[i_0, i_0 + p - 1] = u^m$ for some $m > 1$, a contradiction. This proves that there can be at most one $j \in U$ such that $|i-j|$ is not divisible by p.

Now we show $j_1 = \max(U)$ if $|i - j_1|$ is not divisible by p. Assume for contradiction that there is a $j_2 \in U$ such that $j_2 > j_1$. From the previous paragraph, $|j_2 - i|$ is a multiple of p and $j_1 > i_0$. Hence $j_2 = i_0 + p$. This means that t is $(j_2 - j_1)$-periodic, which is a contradiction since $|j_2 - j_1| < p$. □

Lemma 19 *There is a 1-pass randomized algorithm that computes $F_{0,d}(s)$ with high probability using space $O(F_{0,d}(s) \log^2 n)$ bits.*

PROOF: First we show for $|s| \leq 2d$, how to build the fingerprints of every distinct d-substrings of s in $O(F_0, d(s) \log^2 n)$ bits of space, and handle the general case afterwards. Suppose $|s| \leq 2d$. We claim that s can be divided into three substrings $s = u_1 \circ u_2 \circ u_3$, where $|u_1|$ and $|u_3|$ are $O(F_{0,d}(s))$ and $\operatorname{per}(u_2) \leq F_{0,d}$. Assume $F_{0,d}(s) < d/4$, otherwise the claim is trivially true. Now let $t = F_{0,d}(s) + 1$ and let $s_1, \ldots, s_h$ be the consecutive d-substrings of s. By assumption there exists s_i and s_j such that $i < j \leq t$ and $s_i = s_j$. Again by assumption there exists s_k and s_l where $(j+d-3t-1) \leq k < l \leq (j+d-2t)$ and $s_k = s_l$. This implies that s_l overlaps with s_j in at least $2t-1$ characters. Moreover both $\text{per}(s_j)$ and $\text{per}(s_l)$ are less than or equal to $t-1$. Using Lemma 1, it can be shown that any r-substring of a string with the period p, has period p if $r \geq 2p$. By this fact, we conclude that the last $2t-1$ characters of s_j has period $\text{per}(s_j)$. Consequently $\text{per}(s_j) = \text{per}(s_l)$. Therefore $\text{per}(s[j, l+d-1]) = \text{per}(s_j) \leq t-1 = F_{0,d}(s)$. We let $u_1 = s[1, j-1], u_2 = s[j, l+d-1]$, and $u_3 = s[l+d, |s|]$. This proves our claim.

By the properties of u_1, u_2 and u_3, it should be clear that s can be encoded using at most $O(F_{0,d}(s) \log n)$ bits. Such encoding can be constructed in space $O(F_{0,d}(s) \log^2 n)$ bits in 1-pass using a compression procedure. In fact one can show that $\text{LZ}(s) = O(F_{0,d}(s) \log^2 n)$ where $\text{LZ}(s)$ is the number of codewords output for s by the Lempel-Ziv compression algorithm [LZ77]. This algorithm can be implemented in space $O(\text{LZ}(s))$ with $\tilde{O}(1)$ per-item processing time. To count the number of distinct substrings, as we compress we generate the next fingerprint and we add it to the stack of stored fingerprints if it does not already exist.

For $|s| > 2d$, we divide s into blocks of length at most $2d$ where each d-substring of s belongs to exactly one block and moreover constant number of blocks intersect with each other. We handle each block separately but we keep a unique storage for all the fingerprints. Since constant number of blocks overlap and clearly the number of distinct substrings in a block is less than $F_{0,d}(s)$, we use at most $O(F_{0,d}(s) \log^2 n)$ space in total. This proves our lemma. □

A.2 The 2-pass algorithm for the period

Here we present a 2-pass $O(\log^2 n)$ space algorithm using the pattern matching algorithm we described in Section 3. Let x and y be two strings of length n. Let $\lambda(x, y)$ be the length of largest suffix of $y[2, n]$ that is a prefix of x. We show how to compute $\lambda(x, y)$ in two passes when the input stream is the following sequence.

$$x[1], y[1], x[2], y[2], \ldots, x[n], y[n]$$

By the definition of λ and $\text{per}(\cdot)$, if we replace y with x, we get a 2-pass algorithm for $\text{per}(x)$. Now suppose there is a streaming algorithm $\mathcal{A}$, using $d(n)$ space, for function $\lambda'(x, y)$ defined below.

$$\lambda'(x, y) = \begin{cases} \lambda(x, y) & \text{if} \quad \lambda(x, y) \geq \lfloor \frac{n}{2} \rfloor \\ 0 & \text{otherwise.} \end{cases}$$

It must be clear that given $\mathcal{A}$, we get a $O(d(n) \log n)$ space solution for streaming computation of $\lambda(x, y)$. In fact in what follows we give a simple 2-pass $O(\log^2 n)$ space algorithm to compute $\lambda'(x, y)$ using the result of Theorem 4. Observe that if $\lambda'(x, y)$ is non-zero then the longest suffix of $y[2, n]$ that matches a prefix of x must have $x[1, n/2]$ as a prefix. Let $R = M_{y[2,n]}(x[1, n/2])$. Therefore to find the location of such longest suffix, it is enough to check all $i \in R$. Formally $\lambda'(x, y) = \max(S)$ where

$$S = \{n - i \mid y[i, n] = x[1, n-i] \text{ and } i \in R\}$$

If S is empty then $\lambda'(x, y) = \max(S) = 0$. By Theorem 4, we can output R in the first pass using $O(\log^2 n)$ space. However to compute S we need to know R which might have linearly many elements. However, by Lemma 2, the elements of R form an arithmetic progression (except possibly the last element), hence can be encoded using $O(\log n)$ bits. Given this encoding, in the second pass we compute S and finally the value of $\lambda'(x, y)$. In a naive manner, by running this procedure for $\lambda'(x[1, 2^i], y[n - 2^i + 1, n])$ in parallel, we obtain a $O(\log^3 n)$ space algorithm for $\lambda(x, y)$. However we do not really need the parallel runs. This is because of the property of the pattern matching algorithm. Remember that the algorithm in Section 3 not only finds the occurrences of u in s but also the occurrences of the prefixes $u[1, 2^i]$ as well. This is exactly what we need in each parallel run.

A.3 Augmented indexing problem

Consider the following variant of indexing problem. Alice is given a string s of length n over the alphabet $\Sigma = [k]$. Bob is given an index $i \in [n]$ and the string $s[1..i-1]$. Bob is required to output $s[i]$. We denote this problem by IND_k^n. In the following lemma, we show a lower bound for $CC_\delta^{\rightarrow}(\mathrm{IND}_k^n)$, the δ-error one-way communication complexity of IND_k^n. The proof is a straightforward adaptation of the method from [BJKS02, BJKK04].

Lemma 20 *The one-way communication complexity of* IND_k^n *is* $\Omega((1-\delta)n\log k)$.

PROOF: Alice is given a string X chosen uniformly at random from $[k]^n$. Bob is given an uniformly random integer I from $[n]$ and the prefix of X of length $I-1$. Assume there is an δ-error private coin protocol. Let $A(X)$ denote the message Alice sends when X is given as input. By Fano's inequality (see [CT91])

$$\begin{aligned} H_2(\delta) + \delta\log(k-1) &\geq H(X_I \mid A(X), X_1X_2\dots X_{I-1}, I) && (4)\\ &= \frac{1}{n}\sum_{i=1}^{n} H(X_I \mid A(X), X_1X_2\dots X_{I-1}, I=i) && (5)\\ &= \frac{1}{n}\sum_{i=1}^{n} H(X_i \mid A(X), X_1X_2\dots X_{i-1}) && (6)\\ &= \frac{1}{n}H(X \mid A(X)) && (7)\\ &\geq \frac{1}{n}\left(n\lg k - H(A(X))\right) && (8) \end{aligned}$$

At step (7) we used the chain rule for entropy $H(Z|Y) = H(Z,Y) - H(Y)$. Since we chose X uniformly at random from $[k]^n$, we have $H(X) = n\lg k$. Arranging, we obtain

$$CC_\delta^{\rightarrow}(\mathrm{IND}_k^n) \geq H(A(X)) \geq (1-\delta)n\log k - nH_2(\delta)$$

□

A.4 A lower bound for pattern matching

We define a two-player communication game G as follows. Alice and Bob are given strings of length m and n respectively over the alphabet Σ. Call these strings s and t respectively. Alice sends a single message to Bob and Bob is required to output $\log m$ binary vectors $b_1, b_s, \dots b_{\log m}$ of size n such that $b_i[j] = 1$ iff $s[1, 2^i] = t[j, j+2^i-1]$. A protocol is said to have constant error if with constant probability all vectors b_i are correct in each position.

Theorem 21 *The one-way communication complexity of G is* $\Omega(\log m \log n)$, *provided that* $m^{1+\epsilon} < n$ *for some* $\epsilon > 0$.

PROOF: Suppose Alice and Bob are given an instance of the $\mathrm{IND}_{n/m}^{\log m}$ problem as follows. Alice gets a string $s \in [n/m]^{\log m}$ and Bob gets an integer i and a string $t \in [n/m]^{i-1}$ with the promise that $s[1, i-1] = t[1, i-1]$. Bob is required to output $s[i]$. Alice and Bob construct an instance of G as follows. Alice creates a pattern x by writing s_j, 2^{j-1} times next to each other for $j = 1, \dots, \log m$. Namely,

$$x = s_1 \circ s_2^2 \circ \dots \circ s_{\log m}^{m/2}$$

Bob creates n/m strings $y_1, \ldots, y_{n/m}$, each of length at most m as follows.

$$y_k = t_1 \circ t_2^2 \circ \ldots \circ t_2^{2^{i-2}} \circ k^{2^{i-1}}$$

Bob's text is the concatenation of all y_k for $k = 1, \ldots, n/m$. Using a protocol for G, Bob is able to deduce $s[i]$. Lemma 20 implies a lower bound of $\Omega(\log m \log \frac{n}{m})$, which is $\Omega(\log m \log n)$, when $m^{1+\epsilon} < n$. With a slight modification of this proof one can show that G requires $\Omega(\log m \log n)$ communication even for binary alphabets. $\Box$

Given a 1-pass pattern matching algorithm that outputs $M_s(u[1, 2^i])$ for $i = 1, \ldots, \log m$, we convert it to a one-round protocol that solves G. Therefore Theorem 21 implies a lower bound of $\Omega(\log m \log n)$ for such pattern matching algorithms.