MEASURING TRENDS AND REPETITIONS IN DATA STREAMS

by

Hossein Jowhari
M.S., Sharif University of Technology, 2005
B.S., Teacher Training University, 2002

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in the
School of Computing Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Spring 2012

APPROVAL

Name:	Hossein Jowhari
Degree:	Doctor of Philosophy
Title of Thesis:	Measuring Trends and Repetitions In Data Streams
Examining Committee:	Dr. Cenk Sahinalp Chair
	Dr. Funda Ergun, Senior Supervisor Associate Professor
	Dr. Gábor Tardos, Supervisor Professor
	Dr. Petra Berenbrink, SFU Examiner Associate Professor
	Dr. Ronitt Rubinfeld, External Examiner Professor, CSAIL, MIT
Date Approved:	April 10, 2012

Partial Copyright Licence

Abstract

This thesis is concerned with the study of problems related to the measurement of disorder in the data stream model where the input, accessed in sequential manner, is a long sequence and there are strict memory limitations. In particular, we study sublinear-space algorithms for the following problems.

- Measuring sortedness. In this category, we study the problem of approximating the length of the longest increasing subsequence of a stream. Its dual problem, known as distance to monotonicity, is another measure that is considered in this work.

- Measuring periodicity. We study detecting periodicity and estimating closedness to periodicity in time-series data streams.

- Finding duplicates. In contrast to periodicity which concerns consecutive repetitions, we study standard notion of repetition as well and give algorithms for reporting a duplicate in data streams.

In this thesis, algorithms and lower bounds on the memory requirements are presented for the above problems. In particular, in our main contributions, a tight lower bound on the space complexity of deterministic estimation of the length of the longest increasing subsequences is shown; a streaming algorithm for pattern matching and computing the period of a sequence are given, and finally we show tight bounds on the space complexity of finding duplicates in long streams. To accomplish the latter, we have designed an efficient streaming samplers that works in a general setting and has broad applications in processing dynamic data streams.

Socrates: you see, Meno; I am not teaching the boy anything. All I do is question him. And now he thinks he knows the length of the line on which an eight square foot figure is based.

Acknowledgements

I would like to express my utmost gratitude to my thesis advisor, Funda Ergun. From the beginning of my studies at SFU, Funda has encouraged and helped me to shape my research directions and focus on more important problems. Funda has been a wonderful collaborator, a patient listener and I have to say rather tolerant with my mumblings and unintelligible talk. Thanks to her continuous and unconditional support, I have enjoyed a comfortable flexibility in pursuing my interests and ideas. I feel incredibly indebted to her.

I have also been very fortunate to have Gábor Tardos as my supervisor and co-author. Gábor's mathematical precision and brilliance has been a joyous inspiration for me. Many thanks to Cenk Sahinalp for numerous insights and discussions on research questions, specially on the edit distance problem (unfortunately I ended up disappointing him by doing very little progress there.) I should also thank Ronitt Rubinfeld for kindly hosting my visit to MIT and serving as the external examiner in my thesis defence.

Special thanks to my co-author Mert Sağlam. Some of the results in this thesis are obtained owing to the fresh wave of enthusiasm he brought to our little group. I also thank him for allowing me to use some material of his own thesis.

During my studies I have benefited from the enlightening insights and the help of many researchers. Thanks to Mohammad Ghodsi, Morteza Monemizadeh, Ravi Sundaram, Ravi Kumar, David Woodruff, Seshadhri Comandur, Petra Berenbrink, Artur Czumaj, Piotr Indyk and Alexandr Andoni. In particular my gratitude to Koods (Ravi Sundaram) for the opportunity of the collaboration while I was a master student at Sharif University and later his assistance with my admission to SFU.

This thesis would not have been possible without my friends at SFU and Vancouver. Thanks to Majid Bagheri, Lisa Brunner, Sophie Burrill, Bradley Coleman, Phuong Dao,

Faraz Hach, Iman Hajirasouliha, Alireza Hajkhodabakhshi, Farhad Hormozdiari, Fereydoun Hormozdiari, Basti Koya, Marjan Marzban, Azhvan Sheikhahmady, Shilo StCyr, Fazil Sultan, Kouhyar Tavakolian and Alexander Zaganas.

Finally I would like to thank my parents, Amineh and Abdullah, my brothers and sisters for their love and support. I am grateful to my brother in law Hamed Rahimi for his generous support and encouragement in pursuing my studies.

Contents

List of Figures

Chapter 1

Introduction

In the past few decades, we have witnessed an explosive growth in the size of the data sets. In many areas such as meteorology, genomics, complex physics simulations, internet research, finance and business informatics, there are nowadays data sets with terabytes, exabytes and soon zettabytes of data. Further, besides the increase in the volume, data sets are generated and stored in various forms and fashions which has made the conventional algorithms and data access models inefficient and at cases even obsolete. In particular, challenges and limitations in handling large amount of data has stretched the traditional notions of polynomial time algorithms into linear and sublinear algorithms with probabilistic guarantees and loose approximations. Accordingly processing massive data sets has been the subject of extensive research in the past two decades.

Data streams are ubiquitous these days. Many data sources generate data from observations that evolve over time. Further these sources generate data series which require on-demand processing either due to the storage limitations or the pressing requirements of the application. For example, financial databases depict stock prices continuously to assist investors and answer global demands. Telecommunication and network databases generate numerous data series derived from the usage of the network links such as the duration of the calls and web traffic statistics. These data series can bring about useful information for the providers that seek to balance the load on their servers. Popular websites are flooded with endless streams of queries which require immediate service. Common to all these applications is a demand for data processing tools and algorithms that tackles with huge volume

and rapid data rates.

Identification and measurement of patterns and trends capture valuable insights about the data. For instance, in stock market data, one might be interested in detecting an increasing pattern of investments of a certain stock over a given time interval. Similarly, in a network monitoring application, one may be interested in detecting a fixed consecutive pattern or a near-periodic trend for the sake of intrusion detection and security reasons. Commercial mega websites are interested in finding duplicates and frequent patterns in their click-streams to better advertise their products. In some applications that handle large amount of data, algorithms are often engineered to employ a combination of different methods. For instance, smart engineering of sorting algorithms often can take advantage of information about the the amount of disorder in a sequence [ECW92].

Numerous studies, on detection and measurement of trends in data streams, (often applied) has been done by Database and Data Mining community (e.g. [IKM00, EAE06, RLG$^+$10].) These works however are either domain-specific or do not take storage and access limitations into account. In this thesis, we examine some basic notions of trends in data streams from a theoretical standpoint that takes space limitations into account. We study three notions of trends: monotonicity, periodicity and duplicity. Our studies are in the theme of typical theoretical works in data stream model.

In this section we start with an introduction to the data stream model along with a quick recap of its history. Then we introduce our problems, our contributions and previous works. We conclude this chapter with a brief map of the thesis.

1.1 Data Stream Model

In this thesis we study finite data streams. A typical problem defined in data stream model takes as input a stream or equivalently a sequence of elements $s_1, \ldots, s_n$ from a certain domain. Depending on the application, the domain of the input could be a set of positive integers, pairs of IP addresses, or edges of a massive graph. For extensive discussions and examples on types of data streams, we refer the reader to the survey [Mut05].

A *streaming algorithm* is a randomized algorithm that reads the input stream once (or few times) in sequential manner and outputs a correct answer with some constant probability

greater than 1/2. What is accepted as a correct answer (or an approximate answer) varies depending on the definition the problem. In particular, in this thesis and frequently in the literature, we are interested in randomized relative error approximations. Namely, assuming z as the right answer, the algorithm is required to output some z' where $|z - z'| \leq \epsilon z$ with probability at least $1 - \delta$. For exact algorithms the error parameter ϵ is zero while for deterministic algorithms δ is zero.

The space complexity of a streaming algorithm is the main criterion in our studies. The space is measured as the maximum number of bits used by the algorithm taken over all different input streams and random coins. Naturally we require that the streaming algorithms to work in sublinear space, preferably in polylogarithmic space. Another important criterion for streaming algorithms is *per-item running time* which is defined as the maximum processing time of a stream item taken over all possible inputs and random coins. Number of passes over the stream is another measure that is common in classifying streaming algorithms.

1.1.1 A brief overview of streaming results.

The celebrated paper of Alon, Matias and Szegedy [AMS96], marks the official birth of a long series of streaming results lasting to this day (although the terms "data stream model" and "streaming algorithms" were coined later.) Motivated by certain database applications, [AMS96] studied the streaming complexity of approximating certain functions known as the frequency moments. These functions include some basic statistical inquiries such as the number of distinct elements in a sequence (known as F_0) and the the second frequency moment $F_2 = \sum |x_i|^2$.

Following the work of Alon *et al*, various problems were studied or revisited under the streaming framework. Some notable results include problems with statistical flavour such as approximation of L_p norms [Ind00, FKSV02, IW05, KNPW11], most frequent elements and iceberg queries [KSP03, CCFC04], median and quantile estimations [GM06, CCM08] and histogram constructions [GGI$^+$02, MS03], problems over geometric data such as clustering and classifications of points [GMMO00, COP03, FS05], estimating the diameter of a point set, bichromatic matching and minimum spanning tree on the plane [FKZ04, FIS05],

graph problems including finding near-maximum matchings and approximate shortest path [McG05, FKM+08], sequence problems such as longest increasing subsequence [LNVZ05, GJKK07, GG07, EJ08], and pattern matching and periodicity [PP09, EJS10, CM11]

In parallel with algorithms, lower bounds for space complexity of streaming problems have also been given considerable attention. In fact, some recent advances in communication complexity (e.g information theoretic methods [BYJKS02]) are motivated by data stream applications. In this direction, communication complexity of the multi-player set-disjointness [BYJKS02, CKS03] and the Gap-Hamming problem [CR11, BC09], with application in lower bounding L_p norm estimations, are prominent examples.

Since its inception, the streaming model has undergone various developments and extensions. The sliding window model [DGIM02, BO07] is essentially identical with the data stream model with the difference that the algorithm is required to compute some function over a recent window of the stream. In probabilistic streams [JMMV07] the input is assumed to be generated with respect to some known distribution. In contrast, in random-order model [GM09], the order of the data-items is assumed to be random. In distributed streams and sensor networks [GT01, GK04, CMY08] sketches collected from various sources are combined and processed to culminate in a final aggregate for the entire network. Private streaming algorithms is another new direction that takes privacy and security matters into account [MMNW11, Woo11].

The data stream model has interacts with various neighbouring branches of computer science. Sublinear time and sampling algorithms [RS11, BYKS01], property testing [GGR98], low distortion embeddings [Ind00], sparse recovery and compressed sensing [IR08] have had a close connection with the field.

1.2 Measuring disorder in data streams

In this thesis, we investigate two natural notions of order: monotonicity (aka sortedness) and periodicity. In our main approach, we are interested in measuring the amount of disorder in data streams. More precisely, let the set of sequences `ORDER` represent our idea of order and let d be a distance function defined on pair of sequences. Measures for closeness to `ORDER` can be defined as minimum distance toward this set. In other words, for sequence x,

the amount of disorder measured with respect to d is defined as

$$\min \{d(x, y) \mid y \in \texttt{ORDER}\}.$$

In this thesis, we study algorithms for computing the above distance. On the other hand, we are interested in reconsigning a pattern of monotonicity (e.g. a longest increasing subsequence) or a pattern of periodicity (a periodic substring) in the given stream. In what follows, we lay out the formal definitions and introduce our results for measuring monotonicity and periodicity.

Before we proceed, we remark that in this section by a sequence or stream we mean a series of elements from a finite discrete alphabet such as $\{0, 1, \ldots, m\}$. Although a total order on our domain is not always required, we assume this is the case since it does not effect the generality of our ideas.

1.2.1 Estimating sortedness

In this section we study algorithms that estimate how close a sequence is to being sorted. Following our general framework introduced in the previous section, here our target set representing order is defined as

$$\texttt{SORTED} = \{x \in \Sigma^n \mid x(i) \geq x(i-1)\ \forall i \in [n]\}.$$

In the following we present some well-known measure and distance functions in connection with sortedness.

L_p based distances. For $p \geq 0$, distance to sortedness under L_p is defined as

$$\min\{\ ||x - y||_p^p \mid y \in \texttt{SORTED}\}.$$

In this category, the most and (perhaps) only known example is the *Spearman's footrule distance* defined for permutations. Here $p = 1$ and SORTED is restricted to the identity permutation. Namely $F(x) = \sum_{i=1}^{n} |x(i) - i|$. Computation of $F(x)$ is straightforward in data streams. Diaconis and Garaham [DG77] have shown that the Spearman's footrule estimates the number of inversions in a permutation within a factor of two (see below.)

Kendall tau distance. Kendall distance between two sequences is the number of disagreements in the order of their pair of elements. Formally we have

$$K(x, y) = |\{ (i, j)_{i<j} \mid (x_i - x_j)(y_i - y_j) < 0 \}|.$$

It can be verified that distance to sortedness under Kendall is equivalent with counting inversions in a sequence. Namely,

$$K(x) = \min \{K(x, y) \mid y \in \texttt{SORTED}\} = |\{ (i, j)_{i<j} \mid x_i < x_j \}|.$$

Edit distance. Standard edit distance (or *Levenshtein* distance) between x and y is defined as the minimum number of insertions, deletions and substitution operations needed to convert x to y. Due to the wide range of applications, efficient algorithms for edit distance has been the subject of extensive research (see [AKO10] for the most recent development.) In our context, distance to sortedness under edit distance is known as distance to monotonicity and is defined as

$$\text{ed}(x) = \min \{\text{ed}(x, y) \mid y \in \texttt{SORTED}\}.$$

Similar to the Kendall distance, there is an alternative definition of $\text{ed}(x)$ which is the minimum number of deletion operations needed to turn x into a sorted sequence. Since one way to accomplish this is by keeping a longest increasing subsequence and deleting the rest, $\text{ed}(x)$ equals $|x| - lis(x)$ where $lis(x)$ is the length of the longest increasing subsequence of x.

Variants of edit distance. In addition to the insertion and deletion of single characters as operation, one can consider more general operations such as moving or copying a block of characters or reversing the order of a block. For instance under *edit distance with moves* [CM02], in addition to single character operations of standard edit distance, moving an entire block to any location has a unit cost.

Longest increasing subsequence An increasing subsequence in x is a subsequence $i_1 < \ldots < i_k$ such that $x_{i_1} \leq \ldots \leq x_{i_k}$. Let $\text{lis}(x)$ denote the length of the longest increasing subsequence in x. Here we study streaming algorithms for the following problems. Given

x as a stream, the problem k-LIS asks whether $\text{lis}(x) \leq k$ or not; the problem ϵ-LIS is to output $1+\epsilon$ approximation of $\text{lis}(x)$.

1.2.2 Distance to periodicity

Formally, a sequence s of length n is said to be p-periodic if $s[i] = s[i+p]$ for all $i = 1, \ldots, |s| - p$. 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*. We define `(p)-PERIODIC` to be the set of all p-periodic sequences of length n. Let `PERIODIC` be the set of all periodic sequences:

$$\texttt{PERIODIC} = \bigcup_{p=1}^{n/2} \texttt{(p)-PERIODIC}.$$

All the measures previously defined for sortedness (with the exception of Kendall distance which is more suitable for sortedness) can be extended to measuring the distance to periodicity as well. Here we consider L_p based distances, and in particular Hamming distance, for measuring closeness to periodicity. Formally for $k \geq 0$ and sequence x, let

$$D_{k,p}(x) = \min \{\ ||x - y||_k^k \mid y \in \texttt{(p)-PERIODIC}\}.$$

In the same manner, let $D_k(x)$ denote distance the distance of x to the closest periodic sequence under L_k. In our formulation, the Hamming distance corresponds to the case $k = 0$ which is simply equivalent to the number the number of substitutions needed to make a sequence p-periodic.

1.2.3 Our Contributions on Measuring Disorder

In the first contribution of this thesis, a streaming algorithm for approximating distance to monotonicity is shown. Specifically, a deterministic streaming algorithm which approximates $\text{ed}(x)$ within a factor of $(2+\epsilon)$ using $O(\frac{1}{\epsilon^2}\log^2 \epsilon n)$ space is presented (Theorem 3.1); this improves the previous $4+\epsilon$ factor randomized approximation algorithm of [GJKK07]. This algorithm uses an improved version of the estimator used in [ACCL07, GJKK07] and is based on the characterization of distance to monotonicity by inversions.

Next, a lower bound of $\Omega(\sqrt{n/\epsilon})$ for approximating the length of LIS within $1+\epsilon$ factor is given (Theorem 3.7). The proof of this result is obtained by showing a lower bound

for multi-player communication complexity of LIS via a direct-sum approach suggested in [GJKK07].

On the measures related to periodicity, a streaming algorithm is given for estimating distance to p-periodicity under Hamming distance. For this problem, two streaming algorithms with different guarantees and space usage is given. The first algorithm is obtained by reducing the problem to computing Hamming distance of two vectors that are generated on the fly. This algorithm approximates $D_{0,p}(x)$ within $2+\epsilon$ factor using $\tilde{O}(\frac{1}{\epsilon^2})$ bits of space (Theorem 4.9). The second algorithm returns a $1+\epsilon$ approximation using $\tilde{O}(\frac{1}{\epsilon^{10.5}})$ space (Theorem 4.12). This result is obtained through a combination of the first algorithm, sampling, and exact sparse recovery.

Further on the periodicity trend, a polylog space algorithm is shown for computing the shortest period. Given the reciprocal relation between periodicity and pattern matching, this result is derived from a randomized streaming algorithm for pattern matching. After a single pass over a pattern of m characters, our pattern matching algorithm finds all occurances of the pattern in a stream of length n using only $O(\log n \log m)$ bits (Theorem 4.1). This algorithm is built on deployment of KR-fingerprints and ideas from the work of [PP09]. Equipped with such an algorithm as a black box, the shortest period algorithm makes a single pass over stream x and uses $O(\log^2 n)$ space to find the period of x granted that x is periodic, otherwise it reports that x is aperiodic (Theorem 4.3). The limitation in computing the period for aperiodic sequences turns out to be necessary as we later prove that computing the period in 1-pass for aperiodic sequences requires linear space (Theorem 4.4). On the other hand it is shown that an additional pass will give us a $O(\log^2 n)$ space solution for periods of any length.

In addition to periodicity, the given pattern matching algorithm enables us to get sublinear solutions for frequency moments defined over substrings. In particular a $\tilde{O}(1/\epsilon\sqrt{n})$ space streaming algorithm is shown for counting distinct substrings of length d.

1.2.4 Previous Works on Measuring Disorder

First we review the works on the estimation of sortedness. In this direction, Chan and Pătraşcu [CP10] have given a $O(n\sqrt{\log n})$-time algorithm for counting inversions in a per-

mutation. In general sequences, the best algorithm runs in $O(n \log n / \log \lg n)$ [Die89]. In [AJKS02] Ajtai *et al* have studied estimating $K(x)$ in the context of data streams. It has been shown that for the permutations, $K(x)$ can be approximated within a $1+\epsilon$ factor by a deterministic algorithm that uses only $O(\frac{1}{\epsilon}\log\log n)$ space. In a subsequent work, Gupta and Zane [GZ03] have given a $O(\frac{1}{\epsilon^3}\log^3 n)$ space randomized streaming algorithm for general sequences.

Distance to monotonicity has been studied in the context of property testing. In [EKK+00] the authors have given a $O(\frac{1}{\epsilon}\log n)$ sampling algorithms that distinguishes between sorted sequence and sequences with distance to monotonicity bigger than ϵn. Assuming $\mathrm{ed}(x) = \epsilon_f n$ and some fixed $\delta > 0$, Ailon *et al* [ACCL07] have given another sampling algorithm that returns a value in the interval $[(1/2-\delta)\epsilon, \epsilon]$ that encloses ϵ_f by probing the sequence in $O(1/\epsilon_f \log\log(1/\epsilon_f)\log n)$ locations. Later [GJKK07] adapts the solution of [ACCL07] to data streams and derives a $4+\epsilon$ randomized approximation algorithms using $O(\frac{1}{\epsilon^3}\log^2 n)$ space.

Cormode, Muthukrishnan and Sahinalp [CMS01, CM02] has shown streaming algorithms for approximating $ed_M(x,y)$ (edit distance of with moves) where both x and y are given as a stream in arbitrary order. One can obtain an algorithm for estimating distance of x to sortedness under ed_M by feeding their algorithm the input stream x and the identity permutation. This however works only for permutations.

There is a classical $O(n \log n)$-time deterministic algorithm for computing $\mathrm{lis}(x)$ which is the outcome of a method for sorting cards called the *patience sort*. (see [AD99] for backgrounds and extensive discussions.) The patience algorithm gives an $O(k \log M)$ space streaming algorithm for k-LIS. Liben-Nowell, Vee and Zhu [LNVZ05] were the first to study LIS in data stream model. They showed that any streaming algorithm for k-LIS requires $\Omega(n^{1/2})$ space. Later Gopalan *et al* [GJKK07] gave a $O(\sqrt{\frac{n}{\epsilon}}\log M)$ space deterministic algorithm for ϵ-LIS. Their algorithm works through prunning the patience sort array at roughly $O(\sqrt{n})$ stages. Following this work, Gal and Gopalan in [GG07], in parallel with a similar result in this thesis and via a different proof, has shown a lower bound of $\Omega(\sqrt{\frac{n}{\epsilon}}\log(M/(\epsilon N)))$ for ϵ-LIS.

In [PP09], Porat and Porat 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 and a text of length n, in an off-line step, they preprocess u and build $O(\log n)$ bit sketches for $\log m$ prefixes of u (of geometrically increasing lengths) and use them to find occurrences of the pattern in the length n stream. This algorithm is not fully streaming in the sense that it requires an off-line processing stage over the pattern. Independent of our work on pattern matching [EJS10], Breslauer and Galil [BG11] have gavin a $O(\log n \log m)$ bits pattern matching algorithm with worst case $O(1)$ per-item processing time and one-sided error guarantee. In specific their algorithm never misses a match, whereas our algorithm may miss some occurrences with polynomially small probability. In comparison, the algorithm presented in this thesis takes $O(\log m)$ time per item, hence is slower.

Distance to periodicity has been studied in random access model [LS93, AELL11]. However this thesis is the first to study periodicity in the context of data stream model. Subsequent to our results [EJS10], Crouch et al. studied the distance periodicity problem [CM11]; although their choice of the distance function is the ℓ_2 norm. They give one pass streaming algorithms that compute $(1+\epsilon)$ approximation to distance periodicity using $O(\epsilon^{-2} \operatorname{polylog} n)$ space.

In a related direction, Ergün 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.

Bar-Yossef et al. [BYJKK04] 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.

1.3 Finding duplicates and L_p samplers

The problem of finding the most frequent element in data streams and variations of it have been studied extensively (See the recent survey [CH10] for references.) It is known that finding the most common element or even deceting a repetition requires linear space [AMS96]. However under the assumption that input sequence is long enough to contain at least a repetition, the lower bounds do not hold. Therefore one may ask whether there is a sublinear algorithm for finding duplicates in long sequences. More precisely, in this thesis we study the streaming complexity of the following problem.

Definition 1. *The* `DUPLICATE` *Problem: Given a sequence* $x_1, x_2, \ldots, x_{n+1}$ *where each* $x_i \in [n]$ *report an item that appears at least twice in the input.*

Observe that by the pigeon-hole principle, the existence of such an item is guaranteed. The goal here is to design algorithms for `DUPLICATE` that use small space and make one pass over the input sequence. In this thesis, we give such an algorithm. Toward this result, we reduce the problem of `DUPLICATE` to sampling from dynamic streams (a sequence of insertions and deletions of data items.) To give a flavour of the idea, let $f = (f_1, \ldots, f_n)$ be the vector associated with the frequency of each element in the stream. We are interested in outputting some $i \in [n]$ such that $f_i > 1$. Now let $f' = (f_1 - 1, f_2 - 1, \ldots, f_n - 1)$. Alternatively we are now interested in finding a strictly positive f'_i. One way to achieve this is to sample the coordinates of f' (under addition and subtraction of the coordinates) so that the i-th coordinate is sampled proportional to its weight, and hence no zero coordinate is picked. More formally, given a stream of updates to the coordinates of an underlying vector $x \in \mathbb{R}^n$, we need an algorithm that processes the stream (using sublinear space) and outputs a sample coordinate of x where the i-th coordinate is picked with probability proportional to $|x_i|$. In [MW10] Monemizadeh and Woodruff introduced the idea of L_p samplers which is a generalization of the setting that we just described. Roughly speaking, given non-zero $x \in \mathbb{R}^n$, an approximate L_p-sampler is a streaming algorithm that outputs the index i with probability $(1 \pm \epsilon)|x_i|^p/\|x\|_p^p \pm n^{-c}$, where c is an arbitrary constant. Observe that this guarantee is essentially enough for our purpose.

1.3.1 Our contributions

In this thesis, an L_p-sampler is given that requires only $O(\epsilon^{-p}\log^2 n)$ space for $p \in (1,2)$. For $p \in (0,1)$, the space bound is $O(\epsilon^{-1}\log^2 n)$, while for the $p = 1$ case we have an $O(\log(1/\epsilon)\epsilon^{-1}\log^2 n)$ space algorithm (Theorem 5.1). Similar to the precision sampling algorithm in [AKO10], the generic idea is to scale the input vector with random coefficients so that the i-th coordinate becomes the maximum with probability roughly proportional to $|x_i|^p$. Moreover the maximum (heaviest) coordinate is found through a small-space heavy hitter algorithm.

Additionally, it is shown that sampling from 0, ± 1 vectors requires $\Omega(\log^2 n)$ space. In this special case p is not relevant for L_p-sampling, hence this shows that the proposed L_0-sampling algorithm uses the optimal space up to constant factors, and our L_p-sampler for $p \in (0,2)$ has the optimal space (up to constant factors) for $\epsilon > 0$ a constant.

With the application of the L_1 sampler developed in this thesis wet get an optimal space streaming algorithm for the DUPLICATE problem. This algorithm uses one pass and works in $O(\log^2 n)$ space (Theorem 5.3). Subsequently here it is also shown that any algorithm for outputting a repeated element requires $\Omega(\log^2 n)$ space.

1.3.2 Previous Works

Finding duplicates in streams was first considered in the context of fraud detection in click streams [MAA05]. Muthukrishnan in [Mut] also listed the above solutions and asked whether there exists an algorithm which uses polylog n space while simultaneously taking a constant number of passes. In [Tar07], Tarui showed that any p-pass deterministic algorithm must use $\Omega(n^{1/(2p-1)})$ space to find a duplicate, thereby answering Muthukrishnan's question negatively for deterministic algorithms. Observe that this lower bound comes quite close to the p-pass upper bound, however there is still a gap of p versus $2p-1$ in the exponent, which likely comes from the limitations of 2-player games used in the lower bound proof. In [GR09] a one-pass randomized algorithm with $O(\log^3 n)$ space was presented which outputs a duplicate with constant probability.

In [MW10], it was observed that L_p-samplers lead to alternative algorithms for many known streaming problems, including heavy hitters and frequency moment estimation. Our

L_p samplers work and often give better space performance for all applications listed in [MW10]. We refer the reader to [AKO10, AGM12] for further applications of L_p-samplers.

In [BDM02, BOZ09], the authors have studied sampling from sliding windows, and the recent paper of Cormode et al. [CMYZ10] generalizes the classical reservoir sampling to distributed streams. These works only consider insertion streams. The basic idea of random scaling used in [AKO10] and in our paper has appeared earlier in the priority sampling technique [DLT07, CDK$^+$09], where the focus is to estimate the weight of certain subsets of a vector, defined by a sequence of positive updates.

1.4 Organization

The following chapter explains some preliminary concepts and required backgrounds for the results in this thesis. Each subsequent chapter reflects the results of a published paper. Chapter 3 studies the problem of estimating sortedness and it is based on the work Ergun and Jowhari [EJ08]. Following that in Chapter 4, we present our results on periodicity and pattern matching. The content of this chapter is from the paper Ergun, Jowhari and Sağlam [EJS10] with the exception of Theorem 4.12 which is from unpublished material. Finally in Chapter 5 we present our algorithms for L_p samplers, finding duplicates and related lower bounds. These results are mainly from the work Jowhari, Sağlam and Tardos [JST11].

Chapter 2

Preliminaries

Throughout this thesis $[n]$ denotes the set of integers $\{1, 2, \ldots, n\}$. While $[a, b]$ represents the real interval between a and b, here by $[a \ldots b]$ we mean the set $\{a, a+1, \ldots, b\}$ where $a \leq b$ are two integers. A *string* is a finite length sequence $x_1, \ldots, x_n$, where $x_i \in \Sigma$ for some set Σ. In this thesis, we use strings and sequences interchangeably. We denote by Σ^* the set of all strings over Σ. The length of a string x is denoted by $|x|$. We denote the concatenation of x and y by $x \circ y$. If x and y have the same length, then we speak of the Hamming distance between the two strings, which equals the number of positions in which x and y differ and is denoted $\mathrm{Ham}(x, y)$. In the case we have $\Sigma = \{0, \ldots, q-1\}$ for some integer q, the weight of $x \in \Sigma^*$ is defined, which equals to the number of non-zero positions in x and is denoted by $\mathrm{wt}(x)$.

2.1 Data stream and communication complexity

Specifically we are interested in a general class of data streams known as *turnstile streams* which are series of pairs, i.e

$$(i_1, u_1), \ldots, (i_m, u_m),$$

where (i, u) translates to an addition of u to the i-th coordinate of some initially zero vector $x \in \mathbb{R}^n$. This setting can encode various sorts of problems. For example, *time-series data* in one-dimensional case, is essentially a sequence of integers; here the indices of the pairs, coming in increasing order, represent the time-stamp (or the order) of the data-item.

This setting particularly is suitable for sequence problems where the ordering of the input is relevant. In slightly more general setting, the pair (i, u) can encode the insertion of u occurrences of character i while the vector x represents the frequency of the characters. In this case, where the updates are restricted to positive values, the model is known as the *cash-register model* or *insertion-only* streams. When the final value of the coordinates (after the application of updates) are promised to be non-negative, it is referred to as *strict turnstile* model. For a more extensive discussion and examples, we refer the reader to the survey [Mut05].

A *streaming algorithm* is a randomized algorithm that reads the input stream once (or few times) in sequential manner and outputs a correct answer with some constant probability greater than 1/2. What is accepted as a correct answer (or an approximate answer) varies depending on the definition the problem. In particular, in our thesis and frequently in the literature, we are interested in (ϵ, δ)- approximations. Namely, assuming z as the right answer, the algorithm is required to output some z' where $|z - z'| \leq \epsilon z$ with probability at least $1 - \delta$. For exact algorithms the error parameter ϵ is zero while for deterministic algorithms δ is zero.

The space complexity of a streaming algorithm is the main criterion in our studies. The space is measured as the maximum number of bits used by the algorithm taken over all different input streams and random coins.

Definition 2. *Given problem Q in the data stream model, we define the δ-error streaming complexity of Q, denoted by $\mathcal{SC}_\delta(Q)$, as the minimum taken over all the space complexities of δ-error streaming algorithms for Q.*

Naturally we require that the streaming algorithms to work in sublinear space, preferably in polylogarithmic space. Another important criterion for streaming algorithms is *per-item running time* which is defined as the maximum processing time of a stream item taken over all possible inputs and random coins. Number of passes over the stream is another measure that is common in the classification of the streaming algorithms.

Communication complexity, introduced in [Yao79] by Yao, studies the communication requirements of computing a function whose input is distributed among several parties. Intuitively, communication complexity abstracts away the time requirements of computation,

allowing us to prove lower bounds for computational problems by exploiting information transfer barriers and space bottlenecks.

In the most basic and standard setting, two players, referred to as Alice and Bob, receive different inputs and are required to answer some function defined over both inputs through communicating messages. More generally in a multi-player communication game, k players $P_1, \ldots, P_k$ hold $x_1, \ldots, x_k$ respectively where they are required to find out the answer to $f(x_1, \ldots, x_k)$ for some function f by communicating messages. The players take turns communicating bit strings according to a predetermined protocol. The form of communication in two player games is somewhat straightforward. The players take turn in sending messages to each other. However when there are more than two players, the form of communication may vary and requires clarifications. There are some general classes of protocols for multi-player games. Here we review two main classes that are relevant to the data stream model.

In *blackboard protocols*, also known as *public message* protocols, each player writes his message on a common blackboard seen by all. Moreover the players write in rounds. Each round starts with P_1 speaking and ends with P_k's message. The games terminates when P_k announces the answer in the last round. In *private message* protocols, the player P_i sends private messages to $P_{i+1 \mod (k)}$. Let f be function associated with a k-player game. The communication complexity of a deterministic protocol for f is defined as the maximum total number of bits exchanged taken over all different inputs. Similarly in a δ-error randomized protocol for f, the last players outputs $f(x_1, \ldots, x_k)$ with probability at least $1 - \delta$. The communication complexity of this protocol is the maximum total number of bits written on the blackboard taken over all inputs and random bits. It is assumed that the players use a shared source of randomness.

Definition 3. *The deterministic communication complexity of f under the blackboard model, denote by $\mathcal{CC}(f)$, is defined as the minimum taken over the communication complexity of all blackboard deterministic protocols for f. Similarly $\mathcal{CC}^{PRIVATE}(f)$ is defined for private communication complexity of f. In the same manner, we define the δ-error communication complexity of f and it is denote by $\mathcal{CC}_\delta(f)$. In this case it is the minimum taken over the communication complexity of all δ-error blackboard protocols for f. Like-*

wise, $\mathcal{CC}_\delta^{\texttt{PRIVATE}}(f)$ denotes the δ-error communication complexity of f for private message protocols.

Clearly $\mathcal{CC}_\delta(f)$ is a lower bound on $\mathcal{CC}_\delta^{\texttt{PRIVATE}}(f)$. Same can be said about the deterministic communication complexity. There is a rich theory on communication complexity of various functions and methods for obtaining lower bounds on $\mathcal{CC}_\delta(f)$. We refer the interested reader to the book by Kushilevitz and Nisan [KN97].

The connection between data stream model and multi-player communication games should be clear. Informally speaking, we reduce the problem Q in data stream model into a k-player communication game Q_k^{cc} where the input stream x is divided among k players. The game Q_k^{cc} asks for a communication protocol where the players exchange information in order to output the answer to problem Q over instance x. The main observation here that (abusing the notation) $\mathcal{CC}^{\texttt{PRIVATE}}(Q_k^{cc})$, and consequently $\mathcal{CC}(Q_k^{cc})$, provide a lower bound on $\mathcal{SC}(Q)$. Hence by obtaining a lower bound on the communication complexity of the multi-player games associated with Q, we get a lower bound for its streaming complexity.

2.2 Probability

A *probability distribution* μ is a function $\mu : S \to \mathbb{R}$ for some countable set S, such that $\mu(s) \geq 0$ for all $s \in S$ and $\sum_{s\in S} \mu(s) = 1$. We say that S is the support of μ and write $S = \text{supp}(\mu)$. A *random variable* is, roughly speaking, a variable which equals x with probability $\mu(x)$ for each $x \in \text{supp}(\mu)$, where μ is a probability distribution. We say that X is distributed according to μ and write $\text{dist}(X) = \mu$. Further, we let $\text{supp}(X) := \text{supp}(\text{dist}(X))$, i.e., the support of a random variable is defined to be the support of its distribution.

A distribution is called *Bernoulli* if it is supported on the set $\{0, 1\}$. We say that a random variable is *real-valued* if its support is a subset of $\mathbb{R}$. For a real-valued random variable X, the expectation of X, denoted $\mathbb{E}[X]$, is defined as

$$\mathbb{E}[X] = \sum_{s\in S} s\mu(s).$$

where $\mu = \text{dist}(X)$. For any linear (univariate) function f, by the above fact, we have $\mathbb{E}[f(X)] = f(\mathbb{E}[X])$. The reader may expect that $\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$ holds whenever f is a

convex function. This is indeed true as shown by *Jensen's inequality.*

Lemma 2.1 (Jensen [Jen06])**.** *Let X be real-valued random variable and f be convex function. We have $\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$.*

2.3 Information theory

Let μ and ν be two probability distributions, supported on the same set S. The Kullback-Leibler divergence between μ and ν is denoted by $\mathbf{D}(\mu \,\|\, \nu)$ and defined as

$$\mathbf{D}(\mu \,\|\, \nu) = \sum_{s \in S} \mu(s) \log \frac{\mu(s)}{\nu(s)} \,.$$

Here, we take $0 \log 0$ to be 0. The divergence is undefined if there is an $s \in S$ such that $\mu(s) > 0$ and $\nu(s) = 0$. Some properties of the divergence are given in the next lemma.

Lemma 2.2. *Let X, Y, U, V be arbitrary random variables such that* $\operatorname{supp}(X) = \operatorname{supp}(U)$ *and* $\operatorname{supp}(Y) = \operatorname{supp}(V)$ *and E be an event determined by X. Set $\mu = \operatorname{dist}(X)$ and $\nu = \operatorname{dist}(U)$. The following hold.*

(i) $\mathbf{D}(\mu \,\|\, \nu) \geq 0$.

(ii) $\mathbf{D}(\operatorname{dist}(X \,|\, E) \,\|\, \operatorname{dist}(X)) = -\log \Pr[E]$.

(iii) $\mathbf{D}(\operatorname{dist}(X, Y) \,\|\, \operatorname{dist}(U, V)) = \mathbf{D}(\mu \,\|\, \nu) + \mathbb{E}_{x \sim X} \left[\mathbf{D}(\operatorname{dist}(Y \,|\, X = x) \,\|\, \operatorname{dist}(V \,|\, U = x)) \right]$

Let X and Y be two random variables. The mutual information between X and Y, denoted $\mathrm{I}(X : Y)$, is defined as

$$\mathrm{I}(X : Y) = \mathbf{D}(\operatorname{dist}(X, Y) \,\|\, \operatorname{dist}(X) \operatorname{dist}(Y)).$$

The mutual information of a random variable with itself, i.e., the quantity $\mathrm{I}(X : X)$ is called the self information or the Shannon entropy of X, which plays a central role information and coding theory, among other areas.

Shannon entropy is also denoted by $\mathrm{H}(X)$ and can be extended to a conditional version in the usual way:

$$\mathrm{H}(X \,|\, Y) = \mathbb{E}_{y \sim Y} \left[\mathrm{H}(X \,|\, Y = y) \right].$$

It satisfies the following useful properties, whose proofs can be found in Section 2.1 in [CT06].

Lemma 2.3. *Let X and Y random variables. The following hold.*

(i) $\mathrm{H}(X) \geq 0$*, with equality if and only if X is fixed to a single value.*

(ii) $\mathrm{H}(X) \leq \log |\operatorname{supp}(X)|$*, with equality if and only if X is uniformly distributed on its support.*

(iii) $\mathrm{H}(X, Y) = \mathrm{H}(X \mid Y) + \mathrm{H}(Y)$*. This is called the chain rule for entropy.*

(iv) $\mathrm{H}(X \mid Y) \leq \mathrm{H}(X)$*.*

(v) $\mathrm{H}(X \mid f(Y)) \geq \mathrm{H}(X \mid Y)$*, for any function f. This is called the data processing inequality.*

For convenience, we also introduce the following two functions. Let $p \in [0, 1]$ and $q \in (0, 1)$ be two reals. Define

$$\mathbf{D}_2(p \,\|\, q) = p \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q}$$
$$\mathrm{H}_2(p) = p \log \frac{1}{p} + (1 - p) \log \frac{1}{1 - p} = 1 - \mathbf{D}_2(p \,\|\, 1/2)$$

where $0 \log 0$ is taken to be 0 as above. We will also need the following lemma, which is often referred to as Fano's inequality.

Lemma 2.4 (Fano [Fan61])**.** *Let X and Y be two random variables and let $f : \operatorname{supp}(Y) \to \operatorname{supp}(X)$ be a function. Define $\delta = \Pr[f(Y) \neq X]$. It holds that*

$$\mathrm{H}_2(\delta) + \delta \log(|\operatorname{supp}(X)| - 1) \geq \mathrm{H}(X \mid Y).$$

Proof. Let F be the indicator random variable for the event $f(Y) \neq X$. By Lemma 2.3 we have

$$\begin{aligned}
\mathrm{H}(X \mid Y) &\leq \mathrm{H}(X, F \mid Y) \\
&= \mathrm{H}(X \mid Y, F) + \mathrm{H}(F \mid Y) \\
&\leq \mathrm{H}(X \mid f(Y), F) + \mathrm{H}(F \mid Y) \qquad \text{(by Lemma 2.3 (v))} \\
&= \delta \,\mathrm{H}(X \mid f(Y), F = 1) + (1 - \delta)\,\mathrm{H}(X \mid f(Y), F = 0) + \mathrm{H}(F \mid Y)
\end{aligned}$$

Note that conditioned on $F = 0$, we have $X = f(Y)$ and hence the middle term in the above sum is zero. Thus,

$$\begin{aligned} \mathrm{H}(X \mid Y) &\leq \delta \, \mathrm{H}(X \mid f(Y), F = 1) + \mathrm{H}(F \mid Y) \\ &\leq \delta \log(|\operatorname{supp}(X)| - 1) + \mathrm{H}_2(\delta) \end{aligned}$$

where the last inequality follows from Lemma 2.3 (ii), (iii) and (iv). □

2.4 Concentration bounds

Let $X_1, \ldots, X_n$ be random variables such that $\mathbb{E}[X_i] = \epsilon$ for some $0 \leq \epsilon \leq 1$. Define $X = X_1 + \cdots + X_n$. By linearity of expectation we have $\mathbb{E}[X] = \epsilon n$. The classical results of Chernoff [Che52] and Hoeffding [Hoe63] state that if each X_i is chosen independently, then X is tightly concentrated around its expectation and the exact quantification is as follows.

Theorem 2.1 (Chernoff [Che52])**.** *Let $X = X_1 + \cdots + X_n$, where X_i for $i \in [n]$ are independent binary random variables with expectation ϵ. Then for any $\epsilon \leq \gamma \leq 1$ we have $\Pr[X \geq \gamma n] \leq 2^{-n \, \mathbf{D}_2(\gamma \,\|\, \epsilon)}$.*

The following result is standard, although the specific constants stated in the theorem follows from Stanica's relatively recent lower bound on the binomial coefficients.

Theorem 2.2 (Stanica [Sta01])**.** *Let $X = X_1 + \cdots + X_n$, where X_i for $i \in [n]$ are independent binary random variables with expectation ϵ. Let $0 \leq k \leq n$ be an integer and define $\gamma = k/n$. We have $\Pr[X = k] \geq 2^{-n \, \mathbf{D}_2(\gamma \,\|\, \epsilon)} / \sqrt{\gamma(1-\gamma)cn}$, where $c = 2\pi e^{1/4n}$.*

Theorem 2.1 will be enough to argue concentration in all of our impossibility results. In these settings, there is no limit to the randomness we can use hence we enjoy the full power of Theorem 2.1. However, in our streaming algorithms, we often do not have enough space to store the random bits required to generate fully independent random variables. In such cases k-wise independence comes in handy. Recall that random variables $X_1, \ldots, X_n$ are called k-wise independent if for any $I = \{i_1, \ldots, i_t\} \subseteq [n]$ having at most k elements, $X_{i_1}, \ldots, X_{i_t}$ are distributed according to the product of their marginals. In [SSS95], Schmidt et al. give analogous results to Theorem 2.1 for k-wise independent random variables. We will use the following result (see Lemma 3 and Theorem 4 in [SSS95]).

Theorem 2.3 (Schmidt et al. [SSS95]). *Let $X = X_1 + \cdots + X_n$, where $X_1, \ldots, X_n$ are k-wise independent binary random variables, each having expected value ϵ.*

(i) For any $\epsilon \leq \gamma \leq \epsilon + (1-\epsilon)k/n$ we have $\Pr[X \geq \gamma n] \leq 2^{-n\,\mathbf{D}_2(\gamma \,\|\, \epsilon)}$.

(ii) For any $\epsilon + (1-\epsilon)k/n \leq \gamma \leq 1$ we have $\Pr[X \geq \gamma n] \leq (e\epsilon/\gamma)^k$.

We will also need concentration bounds for continuous random variables. Intuitively, among all random variables Y with expectation ϵ that take values in the real interval $[0,1]$, Bernoulli variables have the most spread out distribution. Hence it is plausible that above concentration bounds hold for sum of arbitrary variables supported on $[0,1]$ too. This intuition is verified in the following theorem, which is immediate from Theorem 5 in [SSS95].

Theorem 2.4 (Schmidt et al. [SSS95]). *Let $X = X_1 + \cdots + X_n$, where $X_1, \ldots, X_n$ are k-wise independent real valued random variables such that $\mathrm{supp}(X_i) = [0,1]$ and $\mathbb{E}[X_i] = \epsilon$ for $i \in [n]$. Then for any $2\epsilon + ek/n \leq \gamma \leq 1$ we have $\Pr[X \geq \gamma n] \leq e^{-\lfloor k/2 \rfloor}$.*

Chapter 3

Estimating Sortedness

In this chapter we present two results on the streaming complexity of estimating the sortedness of a sequence. First we present an algorithm for estimating distance to monotonicity and then we show a lower bound for the space of complexity of approximating the length of LIS in data streams. Both of our results concern deterministic algorithms.

3.1 Approximating the distance to monotonicity

In this section we present a deterministic algorithm for approximating distance to monotonicity. Given a sequence σ, we show a streaming algorithm that approximates $ed(\sigma)$ within a factor of $(2+\epsilon)$ and uses $O(\frac{1}{\epsilon^2}\log^2 \epsilon n)$ space.

Our general approach. We first define an estimator which approximates $ed(\sigma)$, then design an algorithm which approximates the value of the estimator itself. On a high level, our estimator, which is based on a modification to an estimator in [GJKK07], identifies a set of disjoint non-increasing subsequences in σ and uses the sum of the lengths of these sequences as a lower bound for $ed(\sigma)$. The following lemma shows the relationship between these non-increasing subsequences and $ed(\sigma)$.

Lemma 3.1. *Let $P = \{\sigma_1, ..., \sigma_t\}$ be a set of disjoint and non-increasing subsequences in σ. We have $ed(\sigma) \geq (\sum_{i=1}^{t} |\sigma_i|) - t$.*

Proof. Let π be a longest increasing subsequence in σ. It is easy to see that in any non-increasing subsequence of length k in σ, at least $k-1$ of the items do not belong to π. $\square$

Later, we show how to estimate the sum of lengths of these subsequences and how to use this to bound $ed(\sigma)$.

3.1.1 An improved estimator

We now design an estimator which gives a 2-approximation to $ed(\sigma)$. Let σ be a sequence of length n. Let $inv(i)$ be the set of indices $j < i$ such that $\sigma(j) > \sigma(i)$. We say $R \subset [n]$ is a *red set* for σ if $\forall i \in R$ at least one of the following is true:

(i) $i - 1 \in inv(i)$, or,

(ii) there is an interval $I = [j, i-1]$ such that $|inv(i) \cap I| > |R \cap I|$.

When $i \in R$ we say i is a red index; every red index has a witness in the form of another index (if (i) is satisfied) or an interval (if (ii) is satisfied). We call the red set R *total* if $\forall i \notin R$, i does not have a witness. The above definition is similar to the definition used in [**?**] except that the membership of an index in R depends not only on inversions but also on the number of red indices to its left in σ. Our main observation is the following lemma, which links the size of the red set to the distance to monotonicity.

Lemma 3.2. *Let R be a red set for the sequence σ. We have $|R| \leq ed(\sigma)$. Moreover if R is total then $|R| \geq \frac{1}{2} ed(\sigma)$.*

Proof. For the first part, suppose the set $R = \{j_1, ..., j_t\}$ is a red set for σ. Let $G = (V, E)$ be a graph where $V = \{\sigma(1), ..., \sigma(n)\}$. We now introduce an inductive procedure that defines the edge set E. Initially $E = \emptyset$. We scan σ from left to right, and for every index i which is in R, find some $k < i$ such that $k \in inv(i)$ and the indegree of $\sigma(k)$ is zero in G. We then add a directed edge $(\sigma(i), \sigma(k))$ to G. By induction over the indices in R, we prove that this procedure is possible at every step. The base case is trivial and we can add an arbitrary edge $(\sigma(j_1), \sigma(j_1'))$ to E when $j_1' \in inv(j_1)$. Suppose the claim is true for up to j_{r-1}. Consider the index j_r. Since $j_r \in R$, by definition, we are in one of two cases. The first case is when $j_r - 1 \in inv(j_r)$, in which case we can add the edge $(\sigma(j_r), \sigma(j_r - 1))$ to E.

The second is when there exists an interval $I = [l, j_r - 1]$ such that $|inv(j_r) \cap I| > |R \cap I|$. Suppose for $\forall x \in I \cap inv(j_r)$, indegree$(x)$ is nonzero. This is not possible since the edges only originate from the vertices that belong to R and this implies that $|R \cap I| \geq |I \cap inv(j_r)|$ which contradicts our assumption. Therefore we will be able to add an edge starting from j_r as well.

Now consider the graph $G = (V, E)$ at the end of the above process and make the edges undirected (with a little notational abuse, we call the new graph G as well). It is easy to observe that G is composed of a set of disjoint paths. Consider any maximal path $p = (\sigma(j_i), ..., \sigma(j_{i+k}))$; p represents a decreasing subsequence of length k. Additionally, we have $\{j_{i+1}, ..., j_{i+k}\} \in R$. By using Lemma 3.1, we can conclude that $|R|$ is a lower bound for $ed(\sigma)$.

Now we prove the second part of the lemma. Let R be a total red set for σ. We define an iterative pruning procedure that deletes at most $2|R|$ elements from σ and leaves a sorted sequence, similar to that used in [**?**] for showing the lower bound. First let $i = n + 1$ and $\sigma(n + 1) = m$ where m is larger than all of the elements in the sequence. Then iteratively do the following until σ is exhausted: If $i - 1 \notin R$ then proceed to $i - 1$ and repeat. If $i - 1 \in R$; let j be the largest index such that $j < i$ and $j \notin R \cup inv(i)$. Prune the interval $[j + 1, i - 1]$, proceed to j, and repeat.

It is easy to see that at the end of this procedure the resulting sequence is sorted. To bound the number of elements pruned, observe that when we delete an interval, at least half of the elements in the interval belong to R. Thus in total we delete at most $2|R|$ elements. The proof follows. □

3.1.2 Approximating the estimator

We now show a deterministic algorithm for approximating our estimator. Even though the estimator has similarities to that in [GJKK07], due to its definition involving the comparison of $|inv(i) \cap I|$ with $|R \cap I|$, we must design a novel algorithm to compute it. Notice that the exact computation of this quantity can be quite costly if the two quantities are close to each other, or are very small. We show below that an efficient deterministic algorithm which gives an inexact estimate of the quantities suffices to construct a good approximation

algorithm. To be precise, we use an algorithm, which, instead of comparing $|inv(i) \cap I|$ and $|R \cap I|$, checks for two conditions: whether the number of inversions is in the majority and the red indices are far from being majority in an interval. If the test for i passes for any one interval then we make the index i a red index. The detected red set might not be total, however we show that it is large enough to give us a $2 + O(\epsilon)$ approximation.

The majority test. Given some x and an interval I, we want to check whether the number of elements in I that are larger than x is more than $|I|/2$ or not. One can perform this test by comparing x with the median of the elements in I. Since we only require a relaxed version of the majority test, we can use an approximate median for this purpose. The problem of finding the approximate median (and other quantiles) deterministically in a stream is well studied in the literature. Since we need to obtain the approximate median for all widow sizes over the stream, we use a special algorithm from Lin *et al* [LLXY04], whose properties we describe below.

Let S be a set with N elements. A ϕ-quantile ($\phi \in (0, 1]$) of S is the element of rank $\lceil \phi N \rceil$. An element is said to be ϵ-approximate ϕ-quantile if its rank is in $[\lceil (1-\epsilon)\phi N \rceil, \lceil (1+\epsilon)\phi N \rceil]$. The below lemma is a modified version of the theorem of [LLXY04]

Lemma 3.3. *[LLXY04] There is a deterministic streaming algorithm which, given an input stream of length N, using $O(\frac{1}{\epsilon^2}\log^2(\epsilon N))$ space forms a sketch of the stream and can output on demand, using this sketch, an ϵ-approximate quantile of the n most recent elements of the given stream in for any n.*

Let A be the algorithm described in the above theorem. Our algorithm will be making queries to A as follows. Let the output of $A(S, k, \phi)$ be an ϵ-approximation for the ϕ-quantile of the k most recent elements in stream S. While going over the sequence, we assume that we generate a binary sequence that represents the red elements detected so far. let R' be the sequence of these elements, i.e. $R'(i) = 1$ if and only if the algorithm has identified i^{th} element as a red element. We apply the subroutine A to both the input stream (σ) and the sequence R'. We present the algorithm below for a particular interval size $i - j$.

In the following lemma, we analyze the procedure RedTest.

Procedure $RedTest(j, i)$

1. Let $a = A(\sigma, i-j, \frac{1}{2}-\epsilon)$.
2. If $a \leq \sigma(i)$ return FALSE.
3. Let $a' = A(R', i-j, \frac{1}{2}+\epsilon)$.
4. If $a' = 0$ then return TRUE otherwise return FALSE.

Figure 3.1: Description of the Red-Test procedure.

Lemma 3.4. *Let $I = [j, i-1]$. If the majority of elements in I are not in $inv(i)$ then $RedTest(j, i)$ returns FALSE. If more than $(\frac{1}{2}+2\epsilon)|I|$ of the elements in I are in $inv(i)$ and the number of (detected) red elements in I is less than $(\frac{1}{2}-2\epsilon)|I|$ then $RedTest(j, i)$ returns $TRUE$.*

Proof. First part: if the majority of the elements in I are not in $inv(i)$ then the median of I is at most $\sigma(i)$ and since the rank of a is in the range $(\frac{1}{2}-2\epsilon, \frac{1}{2})|I|$ then we should have $a \leq \sigma(i)$; the test returns FALSE. Second part: since more than $(\frac{1}{2}+2\epsilon)$ fraction of I are in $inv(i)$ and the rank of a is in the range $(\frac{1}{2}-2\epsilon, \frac{1}{2})|I|$, it follows that $a > \sigma(i)$. Also since the rank of a' is in the range $(\frac{1}{2}, \frac{1}{2}+2\epsilon)|I|$, we should have $a' = 0$ and the test returns TRUE. □

We now give the main algorithm shown in Figure 3.1.2.

Lemma 3.5. *At the end of Main Algorithm we have $(\frac{1}{2}-O(\epsilon))ed(\sigma) \leq d \leq ed(\sigma)$.*

Proof. Let $i \in R'$. By Lemma 3.4 there exists an interval $I = [j, i-1]$ such that $|inv(i) \cap I| > |R' \cap I|$. It follows that this interval is a witness for i and hence R' is a red set for σ. By Lemma 3.2 $d = |R'| \leq ed(\sigma)$.

Now we show the lower bound. The set R' is not necessarily total. However we show that it is big enough to be bigger than $(\frac{1}{2}-2\epsilon)ed(\sigma)$. We use the same pruning procedure that we used in the proof of Lemma 3.2. Consider the point where $i \notin R'$ and we eliminate the interval $I = [j, i-1]$. By definition of pruning procedure, the elements in I are either in $inv(i)$ or in R'. Suppose $|I \cap R'| < (\frac{1}{2}-2\epsilon)$. Then we should have $|inv(i) \cap I| > (\frac{1}{2}+2\epsilon)$ and hence by Lemma 3.4 $RedTest(j, i)$ should output TRUE. This contradicts $i \notin R'$. It

Main Algorithm.

Upon arrival of element $\sigma(i)$ do the following.

1. For each $j \in [1, i-1]$, do $RedTest(j, i)$. If there exists j such that $RedTest(j, i) = TRUE$ then let $d = d + 1$ and $R'(i) = 1$; otherwise $R'(i) = 0$

2. Proceed to $i + 1^{st}$ element.

At the end, output d.

Figure 3.2: General description of the algorithm for approximating $ed(\sigma)$.

follows that in every interval that we delete, the fraction of red elements is at least $\frac{1}{2} - 2\epsilon$ and hence in total we delete at most $2 + O(\epsilon)|R'|$ elements from the sequence and we get a sorted subsequence. This proves the lower bound. □

Improving the running time. The running time of the above algorithm is $\tilde{O}(n)$ per-item because the algorithm checks every interval. An observation shows that for some small enough $\epsilon_1 < 1$, checking $O(\frac{1}{\epsilon_1} \log n)$ number of intervals is enough. To see this, note that an ϵ_1-approximate ϕ-quantile of an interval with length $|I|$ is also an $(\epsilon_1 + \epsilon_2)$-approximate ϕ-quantile for all intervals with lengths $|I| + 1, ..., (1 + \epsilon_2)|I|$. Hence with the appropriate choice of ϵ_1 and ϵ_2 (where $\epsilon_1 + \epsilon_2 < \epsilon$) and by checking only intervals of length $1, 2, ..., (1+\epsilon')^i, (1+\epsilon')^{i+1}, ..., n$, we can obtain an ϵ-approximate quantile for every interval. Given this, we can state the following theorem.

Theorem 3.1. *Given a sequence σ of length n, there a deterministic streaming algorithm that outputs a $2 + O(\epsilon)$ approximation of $ed(\sigma)$ and uses space $O(\frac{1}{\epsilon^2} \log^2(\epsilon n))$ and $O(\frac{1}{\epsilon^2} \log^3 n)$ per-item running time.*

3.2 Space lower bound for approximating LIS

In this section we prove a lower bound of $\Omega(\sqrt{n})$ for the space complexity of deterministic streaming algorithms that approximate the length of LIS within a small constant factor. Our proof is derived through establishing a lower bound for a multi-player communication game ombined via a direct-sum approach that is suggested in [GJKK07]. Before we begin with the details, we briefly describe the initial framework and an the outline an alternative

proof by Gal and Gopalan [GG07] which directly uses this framework.

3.2.1 The framework of Gopalan *et al.*

Toward deriving a lower bound for the streaming complexity of ϵ-LIS, Gopalan *et al.* [GJKK07] studied a t-player communication ϵ-LIS$^{\text{GJKK}}$ which is defined as follows. First we define the input to the problem. Let t and r be two integers so that $n = rt$. Let $T = T_0 \cup T_1 \subset \Sigma^{rt}$ be the union of two disjoint sets satisfying the following conditions. To simplify the definition, we represent the input sequence $x \in \Sigma^{rt}$ by an $r \times t$ matrix A. The elements of x are listed in A_x through a column-ordering fashion.

$$A_x = \begin{bmatrix} x(1) & x(r+1) & \dots & x(tr-r+1) \\ x(2) & x(r+2) & \dots & x(tr-r+2) \\ \vdots & \vdots & & \vdots \\ x(i) & x(r+i) & \dots & x(tr-r+i) \\ \vdots & \vdots & & \vdots \\ x(r) & x(2r) & \dots & x(tr) \end{bmatrix}.$$

Let x_i denote a subsequence of x that occupies the i-th row of A_x. The first condition is that for each $x \in T$, the columns of the corresponding matrix are strictly increasing, *i.e.* $x(kr+i) > x(kr+i-1)$ for $0 \le k \le t-1$ and $i \in [r]$. Second, for all $x \in T_0$, every row of A_x is strictly decreasing while in contrast, in every $x \in T_1$ there exists i such that $\text{lis}(x_i) \ge \epsilon t$. Given this definition, it is easy to verify that the length of LIS of every sequence in T_1 is at most r while it is at least $r + (\epsilon t - 2)$ in T_2.

The communication game ϵ-LIS$^{\text{GJKK}}$ consists of t players such that the i-th player holds the i-th column of A_x for $x \in T$. The objective is to distinguish between inputs from T_1 and T_2 via communication between the players. The following fact is immediate from the definitions.

Fact 3.2. $\mathcal{SC}(\epsilon\text{-LIS}) \ge \frac{1}{t}\mathcal{CC}(\epsilon\text{-LIS}^{GJKK})$.

To show a lower bound for ϵ-LIS$^{\text{GJKK}}$, Gopalan *et al* proposes a direct-sum approach. Namely, ϵ-LIS$^{\text{GJKK}}$ is expressed as a boolean function g which is defined as the disjunction

(OR) of t instances of a *primitive function* h where $g = \vee_{i=1}^{t} h(x_i)$. The primitive function h is defined as follows.

$$\begin{cases} h(x_i) = 1 & \text{if } \mathrm{lis}(x_i) \geq \epsilon t \\ h(x_i) = 0 & \text{otherwise.} \end{cases}$$

Clearly from the above definitions, it follows that g is zero on T_0 while it is one over T_1.

A (direct-sum) lower bound for g is proved via showing a lower bound for h and then arguing that g needs to solve all the instances $h(x_1) \dots h(x_r)$ simultaneously to get the correct answer. Clearly dealing with h is a simpler task and it turned out to be the right approach. However in deploying this framework, Gopalan *et al.* only succeeded in obtaining a lower bound of $\Omega(rt)$ for communication complexity of g restricted to a subclass of protocols named *Natural* protocols. As result it implies a lower bound of $\Omega(t)$ for ϵ-LIS through the so called Natural algorithms which are one-pass streaming algorithms that can only store some subset of input symbols and nothing else with additional few auxiliary bits (see [GJKK07]).

Gal and Gopalan's proof. Subsequently in [GG07], Gal and Gopalan showed a lower bound for ϵ-LIS$^{\text{GJKK}}$ through restricting the communication between the players to only private messages.

Fact 3.3. $\mathcal{CC}^{\textit{PRIVATE}}(\epsilon\text{-LIS}^{\textit{GJKK}}) = \Omega(rt \log(m/n))$.

Given this fact and the connection between streaming and communication (see Section 2.1), the result below follows.

Corollary 3.4. *[GG07]* $\mathcal{SC}(\epsilon\text{-LIS}) = \Omega(\sqrt{n/\epsilon} \log(m/n))$.

In the next section we use the general (direct-sum) framework of Gopalan *et al.* [GJKK07] and we derive an alternative proof for lower bounding ϵ-LIS. The idea behind our proof is defining a new primitive function h which admits a large fooling set (defined below). We show the existence of such fooling set through probabilistic method. Finally we show that the fooling set for h can be extended to create a large fooling set for g.

0	0	0	0	0	0	1	0	0	1
0	0	1	1	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1
0	0	1	0	1	1	0	1	0	1
1	1	1	1	0	0	0	1	0	0
0	0	1	1	0	1	1	1	1	0
0	0	0	0	0	1	1	1	0	0
0	0	1	1	1	0	1	0	0	0
0	1	1	1	0	1	0	1	0	1

Figure 3.3: A (0,1)-matrix with a monotone path shown in grey color.

3.2.2 An alternative proof

In this section we focus on a certain variant of LIS expressed in the language of $(0,1)$-matrices. Let A be a $(0,1)$-matrix with r rows and t columns where $rt = n$. We define $\mathcal{P}_{xy}(A)$ as the set of all monotone paths starting from the location x and ending at the location y. A monotone path is a sequence of locations that nevers goes downward or to the left. An example is given in Figure 3.2.2. Now let

$$\xi(A) = \max \{w(p) \mid p \in \mathcal{P}_{(r,1)(1,t)}(A) \},$$

where $w(p)$ is the number of 1-cells in p. Similar to ϵ-LIS$^{\text{GJKK}}$, We define the t-player game ϵ-LIS$^{\text{EJ}}$, where i-th player has the possession of the i-th column of A. The objective of the game is to distinguish between two set of inputs which we introduce next.

We define the sets $H_0, H_1 \subset \{0,1\}^t$ as follows.

$$H_0 = \{x \in \{0,1\}^t \mid x(1) = x(t) = 1 \text{ and } x(i) + x(i+1) \leq 1 \; \forall i \in [t-1] \},$$

$$H_1 = \{x \in \{0,1\}^t \mid x(1) = x(t) = 1 \text{ and } w(x) \geq 0.6t \}.$$

In words, H_0 contains only sequences with no consecutive 1s while every sequence in H_1 has weight at least $0.6t$. Now Let $H = H_0 \cup H_1$. We define the boolean function $h : H \to \{0,1\}$ where h is 0 on H_0 and 1 on H_1. Now let $g : H^r \to \{0,1\}$ where $g(X) = \vee_{i=1}^{r} h(X_i)$. Note

that $X \in H^r$ is a (0,1)-matrix with r rows and t columns. An important observation about the function g is that it separates matrices with low ξ from matrices with longer monotone paths. To be precise, we have if $g(X) = 0$ then $\xi(X) \leq r + 0.5t$ otherwise $\xi(X) \geq r + 0.6t$. This gives us the following fact.

Proposition 3.5. *Let $\epsilon \leq \frac{0.1t}{r+0.6t}$. We have $\mathcal{SC}(\epsilon\text{-LIS}) \geq \mathcal{CC}(\epsilon\text{-LIS}^{EJ})$.*

In the following we show a lower bound on $\mathcal{C}(\epsilon\text{-LIS}^{\text{EJ}})$. To prove this bound, we establish the existence of a large fooling set for h and this consequently gives us a large fooling set for g. First we need few definitions.

Let U be some finite universe. Let $R \in U^{k\times t}$. Let $C_j(R)$ be the set of distinct elements that appear in the j^{th} column of R. We define

$$Span(R) = \{\ y \in U^t \mid \forall i \in [t] \quad y_i \in C_i(R)\ \}$$

and we call it the span of the matrix R. Now we define the notion of a *k-fooling set* for function $f : U^t \to \{0, 1\}$ which is a generalization of a standard fooling set definition (cf [**?**]).

Definition 4. *Let $S \subset U^t$ and let k be a positive integer. S is a k-fooling set for f iff f is $c \in \{0, 1\}$ on S but for each subset of S' of S with cardinality k, $Span(S')$ contains y such that $f(y) \in \{0, 1\}/\{c\}$.*

We have the following fact regarding the k-fooling sets.

Fact 3.6. *Let S be a k-fooling set for $f : U^t \to \{0, 1\}$. Let $\mathcal{F}$ be the t-player game corresponding to f. We have $\mathcal{C}(\mathcal{F}) \geq \log(|S|/(k - 1))$.*

In the following lemma, we show the existence of a large fooling set for function h and consequently a large fooling set for g. The latter part is the direct-sum section of our proof strategy.

Lemma 3.6. *There is a k-fooling set for function h of size c^{t-4} where $c \geq (1 - \frac{1}{k^2}) \exp \frac{1}{100k}$. Moreover let F be a k-fooling set for h. Then F^r is a k^r-fooling set for g.*

Proof. (First part) We show the existence of a large $S \subset U^t$ where $h(x) = 0$ for all $x \in S$ and the span of any k elements in S contains a sequence y such that $h(y) = 1$. We use

probabilistic method to prove the existence of such collection of inputs. Let $t' = t - 4$. We pick $x \in \{0,1\}^{t'}$ randomly where $x_i = 1$ with probability p and independent of other coordinates. Let $b(t')$ be the probability that no two consecutive x_i and x_{i+1} are 1 together. When a sequence satisfies this property we refer to it as a good sequence. Using induction we can show that

$$b(t') \geq (1-p^2)^{t'}.$$

To see the above, consider the recurrence $b(t') = (1-p)b(t'-1) + p(1-p)b(t'-2)$ where $b(1) = 1$ and $b(2) = 1 - p^2$. Now suppose we pick L binary sequences in this manner and let the random variable Z represent the number of good sequences. It follows that $E(Z) = (1-p^2)^{t'}L$.

Now we need to find a upper bound on the probability that the span of k random sequences contains a sequence with weight at least $0.6t'$. Let $J_1, ..., J_k$ be k random sequences. We can view these binary sequences as representatives of the subsets of $[t']$. Note that $i \in [t']$ is included in $J = J_1 \cup J_2 \cup ... \cup J_k$ with probability $\gamma = 1 - (1-p)^k$. Hence the expected size of J is $\gamma t'$. Since the inclusion of the elements are independent, we can use Chernoff bound to bound the probability that $|J| < 0.6t'$.

$$Pr(|J| < (\gamma - \alpha)t') \leq \delta = e^{(\frac{-\alpha^2\gamma}{2})t'}.$$

We now set $p = \frac{1}{k}$. Since $\gamma > 1 - 1/e$ setting $\alpha \leq 1 - 1/e - 0.6$ satisfies our requirement. Now since we demand the union of any k random subsets to cover α fraction of the universe, using the union bound, if $\binom{L}{k}\delta < 1$ then with a positive probability there are $(1-p^2)^{t'}L$ many good sequences. We append 10 and 01 to the beginning and the end of these sequences and that would be the set S, our k-fooling set for h. By plugging in the values we get $(1 - \frac{1}{k^2})^{t'}(\frac{1}{\delta})^{\frac{1}{k}} = (e^{\frac{\alpha^2\gamma}{2k}}(1 - \frac{1}{k^2}))^{t'}$ many good subsets. One can inspect that this is bigger than $((1 - \frac{1}{k^2})\exp\frac{1}{100k})^{t'}$. This completes the proof.

(Second part) Note that the members of F are (0,1)-matrices with r rows and t columns. First of all, by the definition of fooling set for h and the definition of g, we have that $g(B) = 0$ for all $B \in F^r$. Now let F' be an arbitrary collection of k^r members of F^r. Let H_i be the set of elements from F_i that appear in F'. Note that H_i is a subset of inputs for h_i. Since the size of F' is at least k^r there exists some $j \in [t]$ such that $|H_j| \geq k$. Now let W be

some subset of k members of F' that cover H_j. Here we regard W as a set of k matrices of order $r \times t$. (Note that g is defined as the OR of h's applied to the rows of those matrices). Consider $Span(W)$. Note that in each $B \in Span(W)$, the i^{th} column of B is picked from the i^{th} column of the one of the matrices in W. From the fact that F_j is a fooling set for h_j, it follows that there exists $y \in Span(H_j)$ (the span of H_j) such that $h_j(y) = 1$. It implies that there exists some $B \in Span(W)$ such that $g(B) = 1$. We conclude that F^r is a k^r-fooling set for g.

□

From the above lemma and the preceding discussion we derive the following theorem.

Theorem 3.7. *Let $\epsilon \in (0, 1/6)$. We have $\mathcal{SC}(\epsilon\text{-LIS}) = \Omega(\sqrt{\frac{n}{\epsilon}})$.*

Proof. To finish the proof of the Theorem 3.7, consider that by Fact 3.6, Lemmas 3.6, and $k = O(1)$, we get

$$\mathcal{C}(\epsilon\text{-LIS}^{\text{EJ}}) \geq \log \frac{|F|^r}{k^r} = \Omega(tr) = \Omega(n).$$

Consequently we get $\mathcal{SC}(\epsilon\text{-LIS}) \geq \frac{1}{t}\mathcal{CC}(\epsilon\text{-LIS}^{\text{EJ}}) = \Omega(n/t)$, where $\epsilon \leq \frac{0.1t^2}{n+0.6t^2}$. By substituting $t = O(\sqrt{\epsilon n})$, we derive the theorem. □

Chapter 4

Periodicity and Consecutive Repetitions

In this section we present our results on measuring the repetitiveness of a data stream in terms of periodicity and closedness to being periodic. We start with some preliminaries and tools that are needed for our algorithms. Then we present a streaming algorithm for pattern matching which is our main component in computing the period of a sequence. After presenting our result on computing the exact short period, we investigate approximate periodicity in terms of measuring closedness to being periodic under substations (Hamming distance).

4.1 Few preliminaries

In this chapter 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 $u \circ v$ or sometimes uv if concatenation is understood from the context. In this chapter we denote by u^i the concatenation of i instances of u.

The following lemma is well-known and a proof of it can be found in [LS62, FW65].

Lemma 4.1 (Lyndon et al. [LS62])**.** *If s is both p-periodic and q-periodic where $p+q \leq |s|$, then s is also* $\gcd(p, q)$*-periodic.*

We denote by $M_s(t)$ 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 shows the relation between $\mathrm{per}(t)$ and $M_s(t)$.

Lemma 4.2. *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 $j = \max(U)$.

Proof. First we prove claim *(i)*. Let $p = \mathrm{per}(t)$. 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 4.1, 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 4.1 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 4.1 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 $|j_2 - j_1| < p$. □

Fingerprints. In Section 4.2 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, BG11] 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 $\Phi_{q,r}(s \circ t)$ by constant arithmetic operations in $\mathbb{Z}_q$.

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

From now on, 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 $\Phi(s)$.

4.2 Streaming pattern matching

We assume the input stream 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. To be precise, 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 4.3. *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 4.1, 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 4.2, we should have $|a_{l+1} - a_l| = \text{per}(u)$ for all $l \in [h-1]$. As a result, when we reach the position $a_2 + m - 1$, we have obtained the value of $\text{per}(u)$. Now let $x = u[1, \text{per}(u)]$. 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 $\text{per}(u)$ divides m, 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 $r = m \pmod{\text{per}(u)}$. Therefore

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

Likewise, we have $s[a_2 + m, a_3 + m - 1] = x[r + 1, \text{per}(u)] \circ x[1, r]$. Therefore, at location $a_2 + m$, we know the value of r and $\text{per}(u)$, and consequently using this information, we can build the fingerprints $\Phi(x[r+1, \text{per}(u)])$ 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 (P3) and (P4) from Section 4.1. 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 $\text{per}(u)$ 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 4.3. First as we go over u, we build $\Phi(u[1, 2^i])$ for all $i \in [\log m]$. By Property (P1) 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 4.3, 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 (P2) in Section 4.1 and the observation that we make at most $O(n \log |u|)$ fingerprint comparisons.

Theorem 4.1. *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 4.4. 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 periodicity algorithms in the next section.

4.3 Finding the period

Testing whether the sequence s is periodic or not is equivalent to testing if there is a suffix of s of length at least $\frac{n}{2}$ that matches a prefix of s. Hence for finding the period of s, we just need to check the positions that match a certain prefix of s. Our algorithms for testing periodicity has two stages. In the first stage, which we call the searching stage, the

algorithm finds the positions where they match the first half of s. This is done by using the pattern matching algorithm of the previous section. Then, in the second stage, which we call the verification stage, we check if the detected position can be the start of a suffix that matches a prefix of s. However these stages are performed in parallel as the search and verification of different positions might overlap. In the following, to demonstrate the idea, first we present a weaker bound and then we handle the general case.

Let $T = M_s(s[1, n/2])$. [1] By definition, 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 $\text{per}(s) \leq n/4$ 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 $\text{per}(s) = i$ otherwise output that $\text{per}(s) > n/4$.

The reason that we only perform the test for $\min(T \cap [1, n/4])$ is a consequence of Lemma 4.2. 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 as, 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 4.4. *There is a 1-pass streaming algorithm that decides whether* $\text{per}(s) \leq n/4$ *or not in space* $O(\log^2 n)$ *bits. The algorithm also outputs the exact period if* $\text{per}(s) \leq n/4$.

For $i > n/4$, checking whether $s[i+1, n] = s[1, n-i]$ is not straightforward. This is because when we find out $i \in T$, 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 $i \in T$, since $s[1, n-i] = s[1, n/2] \circ s[n/2+1, n-i]$, it is enough to build $\Phi(s[n/2+1, n-i])$. Now for $i \in [1, n/2]$, let $s_i = s[n/2+1, n-i]$. Roughly speaking, 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

[1] To make the presentation simpler, we assume n is a power of 2.

case, then we declare s to be periodic, otherwise it is reported aperiodic.

The dynamic set Let $I_k = [n/2-2^k+1, n/2-2^{k-1}]$ and let $H = H_1 \cup H_2 \cup \ldots \cup H_{\log(n/4)}$ where $H_k = M_s(s[1,2^k]) \cap I_k$. In other words, H_k is the positions of all occurrences of $s[1,2^k]$ that start within the interval I_k. Clearly $T \subseteq H$. 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 4.2 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 $p = \text{per}(s[1,2^k])$.

The case $p > \frac{1}{4}2^k$. By Lemma 4.2, 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 $p \leq \frac{1}{4}2^k$. Here things get a bit complicated. 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, first we note that, by Lemma 4.2, the positions in H_k have a succinct representation as the distance between consecutive positions is exactly p. As result, we can encode R_k using $O(\log n)$ space. Further, 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, loosely speaking, 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 4.4, 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

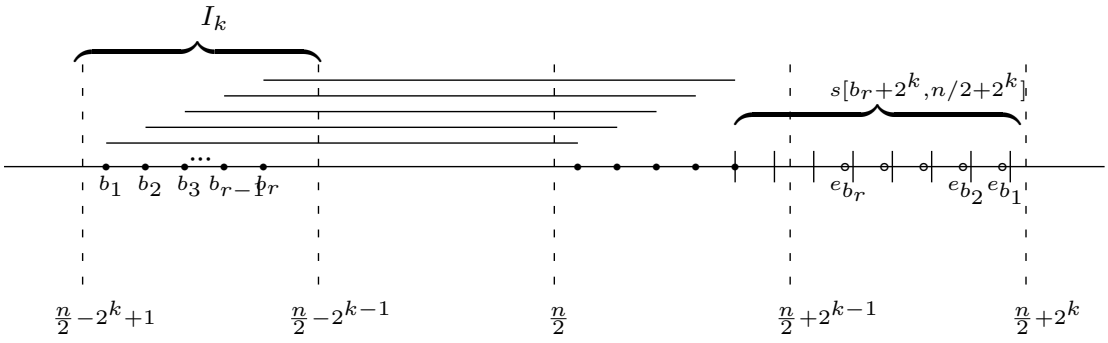

Figure 4.1: A sample run of the algorithm in Section 4.3.

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). 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 $\{s_i \mid i \in H_k\}$ 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 4.2. *If $y < l$ and $|r - j| + y \neq x$ then $b_j \notin T$.*

By Claim 4.2, $|H_k \cap T| \leq 1$. Consequently it is enough to maintain $\Phi(s_{b_j})$ where $|j - r| + y = x$ and $\Phi(s_{b_1})$. So in this case $|R_k| \leq 2$.

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 4.4). Computation of y is similar to x. Finally, 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 4.3. *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.

Theorem 4.4. *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 Section 5.4.1). Suppose Alice and Bob are given an instance of AIND_2^n. In this problem 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 $a = x \circ 2$ and Bob sets $b = 0^{n-i} \circ y \circ 1$. Clearly, $\mathrm{per}(a \circ b) = i$ if and only if $x[i+1] = 1$. Hence by Theorem 5.9 an $\Omega(n)$ communication bound holds for any 1-way protocol. The claim of the theorem follows by noting that any 1-pass algorithm that computes $\mathrm{per}(s)$ can be converted to a protocol for the above communication game. □

4.4 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 [KNW10a,

KNW10b, Gan11] we 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[2]. 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 4.5. *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 4.2.

Now we show that $F_{0,d}$ can be computed exactly using space $\tilde{O}(F_{0,d})$. 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 $\mathrm{per}(u_2) \leq F_{0,d}$. 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 $\mathrm{per}(s_j)$ and $\mathrm{per}(s_l)$ are

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

less than or equal to $t-1$. Using Lemma 4.1, 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.

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.

Hence 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. □

4.5 Approximating the distance to periodicity

Recall that $D_{0,p}(s)$ 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, $D_{0,p}(s)$ is the the minimum number of substitutions in A to make every row consist of d repetitions of the same character. Also, $D_{0,p}(s) = L_1 \circ F_1^{res(1)}(A) = \sum_{i=1}^{p} F_1^{res(1)}(A_i)$. 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.

Theorem 4.6. *There is a deterministic streaming algorithm that approximates $D_{0,p}(s)$ within $1-\epsilon$ factor using $O(\frac{p}{\epsilon})$ words of space.*

Now we turn our attention to randomized algorithms. In the following, to simplify notation, we use $F(A_i) = F_1^{res(1)}(A_i)$ and $F(A) = L_1 \circ F_1^{res(1)}(A)$.

4.5.1 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 4.7. $F(A_i) \geq \frac{1}{d}\sum_{a<b} f_i(a)f_i(b) \geq \frac{1}{2}F(A_i)$.

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 $\max\{f_i(a)\} = d$. □

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. [KNW10a].

Theorem 4.8. [KNW10a] *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 4.8 and Fact 4.7, we get a $2+\epsilon$ approximation for $F(A)$ using 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 subset S from $[d]$ of size $O(\frac{1}{\epsilon^2}\log p)$ and, for $j \in S$, we compare $A_i(j)$ with all the coordinates of A_i. Now this gives us a vector v_i' with dimension $d|S|$. 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 $\mathbb{E}[L_0(v'_i)] = \sum_{j=1}^{d} \mathbb{E}[Y_j] \sum_{k=1}^{d} \mathrm{Ham}(A_i(j), A_i(k)) = \frac{2|S|}{d} \sum_{a<b} f_i(a) f_i(b)$. 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 $\frac{L_0(v')}{2|S|}$ 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 4.9. *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 $1/4$.*

4.5.2 A $(1+\epsilon)$ algorithm

We start by making few observations that are crucial to the algorithm in this section. Then we proceed with the description of our algorithm. 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 $F(A_i)$. The worst case for this approximation happens when $F(A_i)$ is maximized, i.e., $F_\infty(A_i) = d/F_0(A_i)$. On the other hand, when $F(A_i)$ is low, the above quantity gives us a good estimate. This is because $F'(A_i)$ is lower bounded by $\frac{1}{d}(d - F_\infty(A_i))F_\infty(A_i)$ which implies the following.

Fact 4.10. *Let $\epsilon \geq 0$. Suppose $F(A_i) \leq \epsilon d$. We have $F'(A_i) \geq (1-\epsilon)F(A_i)$.*

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

$$F'(A) + \frac{1}{2d} \sum_{i=1}^{p} (F(A_i)^2 + F_2^{res(1)}(A_i)) = F(A). \tag{4.1}$$

Now let $F''(A_i) = F(A_i) - F'(A_i) = \frac{1}{2d}(F(A_i)^2 + F_2^{res(1)}(A_i))$. From (4.1), it follows that if we are given an estimate of $F''(A) = \sum_{i=1}^{p} F''(A_i)$, by using the algorithm described in the previous section, we get a $1 + \Theta(\epsilon)$ approximation for $F(A)$. On the other hand, Fact 4.10 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 4.10

and (4.1).

$$F(A) \geq F'(A) + \sum_{i \in H_{\epsilon d}} F''(A_i) \geq (1-\epsilon)F(A). \tag{4.2}$$

To estimate $\sum_{i \in H_{\epsilon d}} F''(A_i)$, we estimate $|H_{\epsilon d}|$ and we also take uniform samples from the rows in $H_{\epsilon d}$. Now if the contribution of $H_{\epsilon d}$ in $F(A)$ is high, our samples give us a good estimate of $F''(H_{\epsilon d})$, otherwise we can neglect the contribution of these rows.

Our algorithm has two main threads running in parallel. In one thread, we run the $2+\epsilon$ approximation algorithm over A and in the other thread we run the sampling procedure which we describe below. At the end, we add up the outcome of these threads and that is the final output. In the following, we present our sampling procedure along with the analysis of its correctness. However before that we need to give a brief description of a sparse recovery procedure that we use in the main algorithm.

Sparse Recovery. Given an update stream that implicitly defines a vector $x \in \mathbb{R}^n$, we are interested in space efficient algorithm that recovers the non-zero coordinates of x. We need such an algorithm as a subroutine in our main algorithm of this section. Generally it is known that when it is guaranteed that x will have at most r non-zero coordinates, a $O(r \log n)$ space sparse recovery algorithm exists. In our case, since the updates are limited (at most 2 updates to each coordinate), we use the following result by Lipsky and Porat that gives a time and space efficient algorithm for the limited case that we are interested in.

Theorem 4.11 (Lipsky et al. [PL07])**.** *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 $\mathrm{Ham}(x,y) > r$ or not using $O(r(\log n + \log|\Sigma|))$ bits of space and $O(\log n)$ per-item time. Moreover in case $\mathrm{Ham}(x,y) \leq r$, the algorithm finds all pairs $(x[i], y[i])$ where $x[i] \neq y[i]$. The probability of error is at most n^{-1}.*

Having the above result, for positive integer k and n by n' matrix A with entries from $[m]$, we define a randomized procedure $SR_r(A)$ as follows. Given the entries of A in a column-order fashion, $SR_r(A)$ outputs all the content of the rows that contain two entries that are different. If A has more than r number of such rows, with high probability $SR_r(A)$

Let $B_i\langle 1\rangle, \ldots, B_i\langle 2\log n\rangle$ be a random partitioning of the characters in B_i into $2\log n$ equal-size sets, (note that each $B_i\langle j\rangle$ is a string with $k' = \frac{16}{\delta^2}\log n$ length.) Now for $j = 1, \ldots, 2\log n$, let

- $\alpha_{i,j} = \frac{d^2}{2\binom{t}{2}} \sum_a f_{B_i\langle j\rangle}(a)(f_{B_i\langle j\rangle}(a) - 1)$.
- $\varphi_{i,j} = \frac{1}{2d}(F(B_i\langle j\rangle^{\oplus \frac{d}{k'}})^2 + \frac{1}{2d}(\alpha_{i,j} - F_\infty^2(B_i\langle j\rangle^{\oplus \frac{d}{k'}})$

Let $\varphi(B_i)$ be the median of $\{\varphi_{i,1}, \ldots, \varphi_{i,2\log n}\}$.

Figure 4.2: Description of $\varphi(B_i)$

rejects the input. Given the above result, we can implement $SR_r(A)$ in $O(r(\log n + n' \log m))$ bits.

The sampling procedure. Our sampling procedure is described two main phases. In the first phase, we downsize the input matrix A, by picking a random subset of columns of size $k = O(\frac{1}{\delta^2}\log^2 n)$. Let B be the projection of A over the random columns. We define function $\varphi(B_i)$ which gives an estimate for $F''(A_i)$. The description of this function is given in Figure 4.5.2. Let $x^{\oplus t}$ denote the string resulted from x by repeating each character t times. We use $F((B_i)^{\oplus \frac{d}{k}})$ to estimate $F(A_i)$, where $k = |B_i|$. Now let $G_{\epsilon d} = \{i | F((B_i)^{\oplus \frac{d}{k}}) \geq \epsilon d\}$. In the second phase of our sampling procedure, we take samples from $G_{\epsilon d}$. We do this step, by devising an exact sparse recovery procedure. In the following we give a brief description.

While we take sample $i \in G_{\epsilon d}$, we also compute $\varphi(B_i)$. Finally we use these samples to estimate $\sum_{i \in G_{\epsilon d}} \varphi(B_i)$. The detailed steps of our algorithm is given in Figure 4.3.

We analyse the correctness of our algorithm in the following lemmas. In the next lemma, we show that the matrix B, with high probability, satisfies two important properties. First, the sum $\sum_{i \in G_{\epsilon d}} \varphi(B_i))$, added with $F'(A)$, gives a good estimate of $F(A)$. Second, the rows in $G_{\epsilon d}$ comprise a large enough fraction of the non-uniform rows in B. The latter fact helps us in getting an efficient sampler for $G_{\epsilon d}$.

Lemma 4.5. *Let $\delta < \frac{1}{10}\epsilon^2$. With probability at least $1/8$, the followings are the case.*

1. $|F(A) - F'(A) - \sum_{i \in G_{\epsilon d}} \varphi(B_i)| \leq 2\epsilon F(A)$.

2. *Let* $\gamma = \frac{F(H_{2\epsilon d})}{F(A)}$. *We have* $|G_0| \leq \frac{16k}{\gamma}|G_{\epsilon d}|$ *where* G_0 *is the set of non-uniform rows in* B.

Proof. First we prove for all $i \in [p]$, with high probability, $|F(A_i) - F((B_i)^{\oplus \frac{d}{k}})| \leq \delta d$ and $|F''(A_i) - \varphi(B_i)| \leq 5\delta d$. Fix an i. By Chernoff bounds, for $a \in \Sigma$, with probability at least $1 - \frac{1}{8n^3}$,

$$|f_{(B_i)^{\oplus \frac{d}{k}}}(a) - f_{A_i}(a)| \leq \delta d.$$

Consequently, using union bound, with probability at least $1 - \frac{1}{8n^2}$, we have $|F(A_i) - F((B_i)^{\oplus \frac{d}{k}})| \leq \delta d$.

Now for the second part, fix some $j \in [2 \log n]$ and consider the term $\varphi_{i,j}$. With probability at least $1 - \frac{1}{8n^2}$ we have

$$|f_{B_i\langle j\rangle^{\oplus \frac{d}{k'}}}(a) - f_{A_i}(a)| \leq \delta d.$$

Therefore, with probability at least $1 - \frac{1}{8n}$, the error of the first term in $\varphi_{j,i}$, i.e., $\frac{1}{2d}(F(B_i\langle j\rangle^{\oplus \frac{d}{k'}})^2$, is bounded by $2\delta d$. To bound the error of the second term in $\varphi_{i,j}$, we use Chebyshev bound and the variance analysis of [BYKS01] (cf. Lemma 5.3) to estimate F_2. From [BYKS01], we have $\mathbb{E}[\alpha_{i,j}] = F_2(A_i)$ and $\mathrm{Var}(\alpha_{i,j}) \leq \frac{d}{t}(F_2(A_i))^{3/2}$. Using Chebyshev's inequality, we get

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

Given that $k' = \frac{16}{\delta^2}\log n$, this probability is bounded by $1/(16 \log n)$. It follows that with probability at least $1 - 1/(16 \log n)$, the second term of $\varphi_{i,j}$ has error at most $3\delta d$. Since φ_i is the median of $2 \log n$ outcomes, with probability at least $1 - 1/(n^2 \log n) - 1/16n$, we have $|\varphi(B_i) - F''(A_i)| < 5\delta d$.

Proof of (I): Following the above argument and using union bound, with probability at least $1 - p/(8n)$,

$$H_{(\epsilon+\delta)d} \subseteq G_{\epsilon d}, \quad ([p] \setminus H_{(\epsilon-\delta)d}) \cap G_{\epsilon d} = \emptyset \tag{4.3}$$

Since $\delta < \frac{1}{10}\epsilon^2$, we get $5\delta d \leq \epsilon(\epsilon - \delta)d$ and hence for all $i \in G$, φ_i is away from $F''(A_i)$ by at most $\epsilon F(A_i)$. Putting these observations, (4.2), and (4.3) together we get the desired statement.

We pick $k = \frac{32}{\delta^2}\log^2 n$ random columns of A and let B be the projection of A over sampled columns. Run the following in parallel for $j = 1, \ldots, \frac{1}{\epsilon}\log p$. Set $l = \frac{128k}{\epsilon^{1.5}}$.

1. Repeat the following for $u = 1, \ldots, (t = \frac{128}{\epsilon^4}\log^2 n)$ in parallel. The output of the u-th thread is the pair $(z_{j,u}, v_{j,u})$.

   (a) Select a function $h(x) = (ax + b \mod q)$ by picking a and b randomly from the field F_q where q is the smallest prime $\geq p$.

   (b) Let $S_{j,u} = \{i | h(i) \leq \frac{q}{(1+\epsilon)^j}\}$.

   (c) Run the SR_l procedure over the projection of B over the rows in $S_{j,u}$. If $|S_{j,u} \cap G_0| > l$ or $S_{j,u} \cap G_{\epsilon d} = \emptyset$, then stop and output $z_{j,u} = 0$, $v_{j,u} = 0$. Otherwise let $z_{j,u} = |S_{j,u} \cap G_{\epsilon d}|$ and $v_{j,u} = \varphi(B_{i_R})$ where i_R is randomly selected from $i \in S_{j,u} \cap G_{\epsilon d}$. Output the pair $(z_{j,u}, v_{j,u})$.

2. Let $z_j = \sum_u^t z_{j,u}$. Partition the interval $[t]$ into $t_1 = \frac{t}{16 \log n}$ blocks of equal size, $T_1, \ldots, T_{t_1}$. For $c \in [t_1]$, let $\mathcal{T}_{j,c} = \{u | v_{j,u} > 0, u \in T_c\}$. Select $\bar{v}_{j,c}$ randomly from the set of $v_{j,u}$'s where $u \in \mathcal{T}_{j,c}$. If $\mathcal{T}_{j,c} = \emptyset$, then we set $\bar{v}_{j,c} = 0$. Set $v_j = \sum_c^{t_1} \bar{v}_{j,c}$. Let x_j be the number of non-empty sets $\mathcal{T}_{j,c}$ where $c \in [t_1]$. Output the triple (z_j, v_j, x_j).

Let $\hat{j}$ the largest j such that $|z_j - t| \leq 2\epsilon t$ and $x_j = t_1$. The final output of this phase is $v = (1+\epsilon)^{\hat{j}} \frac{v_{\hat{j}}}{t_1}$. If there is no such j then we then output $v = 0$.

Figure 4.3: Description of the sampling procedure.

Proof of (II): We have $\mathbb{E}[F(B)] \leq \frac{k}{d}F(A)$. Since always $F(B) > |G_0|$, by Markov inequality, we have $\Pr[|G_0| > \frac{16k}{d}F(A)] < 1/16$. Assuming the event $|G_0| \leq \frac{16k}{d}F(A)$, by definition of γ and the fact that $F(H_{2\epsilon d}) \leq d|H_{2\epsilon d}|$ we have $|G_0| \leq \frac{16k}{\gamma}|H_{2\epsilon d}|$. At the other hand, from (4.3) it follows that $H_{2\epsilon d} \subseteq G_{\epsilon d}$. This implies that $|H_{2\epsilon d}| \leq |G_{\epsilon d}|$ and consequently our statement is correct. □

The following lemma implies that the outcome of the sampling procedure has small error.

Lemma 4.6. *Let $W = \sum_{i \in G_{\epsilon d}} \varphi(B_i)$. With probability at least $1 - o(\frac{1}{n})$, the following statements are the case.*

1. *$v \leq (1 + \Theta(\epsilon))W$.*
2. *If $F''(H_{2\epsilon d}) \geq \epsilon F(A)$, then $v \geq (1 - \Theta(\epsilon))W$.*

Proof. First, we observe that if $|G_{\epsilon d}| = 0$ then always $v = 0$, and hence the above statements are satisfied. Therefore in the following we assume $|G_{\epsilon d}| > 0$. To prove (I), fix $j \in [\frac{1}{\epsilon}\log p]$ and $u \in [t]$. Let $\beta_j = \frac{|G_{\epsilon d}|}{(1+\epsilon)^j}$. Consider the random variable $z_{j,u}$. Since the hash function h is uniform, $\mathbb{E}[z_{j,u}] \leq \beta_j$. By linearity of expectation, we have $\mathbb{E}[z_j] \leq \beta_j t$. Since the threads are independent, we get $\Pr(z_j > (1+\epsilon)\beta_j t) < \exp(-\frac{\epsilon^2 \beta_j t}{4|G_{\epsilon d}|}) < \exp(-\frac{\epsilon^2 t}{4}) < \frac{1}{n^2}$. Now let j' be the smallest j such that $\beta_j \geq 1$. It follows that, by applying union bound, with probability at least $1 - \frac{\log p}{\epsilon n^2}$, we have $\hat{j} \leq j' + 3$ and consequently,

$$(1+\epsilon)^{\hat{j}} < (1 + \Theta(\epsilon))|G_{\epsilon d}| \tag{4.4}$$

On the other hand, $v_{\hat{j}}$ is the sum of t_1 independent uniform samples from $\{\varphi(B_i)\}_{i \in G_{\epsilon d}}$. By Chernoff bound and the fact that $\varphi(B_i) \geq \frac{\epsilon^2}{2}d$, we get

$$\Pr(|v_{\hat{j}} - \frac{t_1}{|G_{\epsilon d}|}\sum_{i \in G_{\epsilon d}} \varphi(B_i)| > \epsilon \frac{t_1}{|G_{\epsilon d}|}\sum_{i \in G_{\epsilon d}} \varphi(B_i)) < \exp(-\frac{\epsilon^4 t_1}{8}) < \frac{1}{n^2}. \tag{4.5}$$

This and (4.4) proves the statement of (I). To prove (II), we show that, given the assumption of the statement, with high probability, we have $\hat{j} \geq j'$ and $x_{j'} = t_1$. This, combined with (4.4) and (4.5), proves our claim. Consider the random variable $r_{j',u} = |R_{j',u}|$. Since the hash function h is uniform, we have $\mathbb{E}[r_{j',u}] = \frac{|G_0|}{(1+\epsilon)^{j'}}$. Also since h

is pairwise independent, $\mathrm{Var}[r_{j',u}] = \mathbb{E}[r_{j',u}] + \frac{1}{q-1}\mathbb{E}^2[r_{j',u}]$. As result, using Chebyshev bound, we get

$$\Pr\left(|r_{j',u} - \mathbb{E}[r_{j',u}]| > s\,\mathbb{E}[r_{j',u}]\right) \leq \frac{1}{s^2}\left(\frac{(1+\epsilon)^{j'}}{|G_0|} + \frac{1}{q-1}\right). \tag{4.6}$$

Now we observe that, by part (II) in Lemma 4.5, $\mathbb{E}[r_{j',u}] \leq \frac{32k}{\epsilon}$. After plugging this in (4.6) and using the fact that $|G_0| \geq |G_{\epsilon d}|$, we get $\Pr\left(|r_{j',u} - \mathbb{E}[r_{j',u}]| > \frac{32ks}{\epsilon}\right) \leq \frac{2}{s^2}$, and consequently by setting $s = \sqrt{\frac{2}{\epsilon}}$,

$$\Pr\left(r_{j',u} > \frac{128k}{\epsilon^{1.5}}\right) \leq \epsilon.$$

Now since $l > \frac{128k}{\epsilon^{1.5}}$, by linearity of expectation, $\mathbb{E}[z_{j'}] > (1-\epsilon)t$ and therefore, by Chernoff bound, $\Pr(z_{j'} < (1-2\epsilon)t) < \exp(-\frac{\epsilon^2 t}{8})$.

It remains to show $x_{j'} = t_1$ with high probability. For $c \in [t_1]$, let X_c be the indicator variable for the event $\mathcal{T}_{j',c} \neq \emptyset$. By definition, $x_{j'} = \sum_c^{t_1} X_c$. Let $Z_c = \sum_{u \in T_c} z_{j',u}$. We have $\Pr(X_c) = \Pr(Z_c \geq 1)$. By $\mathbb{E}[Z_c] \geq \frac{t}{t_1}(1-\epsilon)$, and Chernoff bound, we get $\Pr(Z_c < 1) < \exp(-(\frac{t}{4t_1})^2)$. It follows that $\Pr(X_c) = 1 - \exp(-(\frac{t}{4t_1})^2)$. Therefore, by applying union bound, we get $Pr(x_{j'} = t) \geq 1 - t_1 \exp(-(\frac{t}{4t_1})^2) > 1 - \frac{t_1}{n^2}$. □

Putting the statement of part (I) in Lemma 4.5, Lemma 4.6, and (4.2) give us our main theorem in this section. Considering the parallel repetitions, the space usage of the algorithm is governed by $O(\frac{1}{\epsilon} lt \log n \log p)$ which amounts to $O(\frac{1}{\epsilon^{10.5}} \log^5 n)$ after plugging the values. Finally, assuming n is large enough, we get the following theorem.

Theorem 4.12. *There is a randomized 1-pass streaming algorithm that outputs a* $1 \pm \epsilon$ *approximation of* $D_{0,p}(s)$ *with probability at least* $3/4$ *using* $O(\frac{1}{\epsilon^{10.5}} \log^5 n)$ *words of space.*

Chapter 5

Finding Duplicates and L_p-Samplers

In this chapter first we present our L_p sampler algorithms and then we give our results for finding duplicates. Our duplicate detection algorithm uses the L_1 sampler as a black box. Further we show lower bounds on the space complexity of finding duplicates and L_p samplers.

5.1 The L_p Sampler

In this section, we present our L_p sampler algorithm. In the following, we assume $p \in (0, 2)$. This particular method does not seem to be applicable for the $p = 2$ case and we know of no $O(\log^2 n)$ space L_2-sampling algorithm. We treat the $p = 0$ case separately later.

5.1.1 Preliminaries and definitions

Recall that an update stream is a sequence of pairs (i, u), where $i \in [n]$ and $u \in \mathbb{F}$ for some field $\mathbb{F}$. The stream of updates implicitly define an n-dimensional vector $x \in \mathbb{F}^n$ as follows. Initially, x is the zero vector. An update of the form (i, u) adds u to the coordinate x_i of x (leaving the other coordinates unchanged). Roughly speaking, given a stream of updates (additions and subtraction) to the coordinates of an underlying vector $x \in \mathbb{R}^n$, an L_p-sampler processes the stream and outputs a sample coordinate of x where the i-th

coordinate is picked with probability proportional to $|x_i|^p$.

Definition 5. *Let $x \in \mathbb{R}^n$ be a non-zero vector. For $p > 0$ we call the L_p* distribution *corresponding to x the distribution on $[n]$ that takes i with probability*

$$\frac{|x_i|^p}{\|x\|_p^p},$$

with $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$. For $p = 0$, the L_0 distribution corresponding to x is the uniform distribution over the non-zero coordinates of x.

We call a streaming algorithm a *perfect L_p-sampler* if it outputs an index according to this distribution and fails only if x is the zero vector. An *approximate L_p-sampler* may fail but the distribution of its output should be close to the L_p distribution. In particular, we speak of an ϵ relative error L_p-sampler if, conditioned on no failure, it outputs the index i with probability $(1 \pm \epsilon)|x_i|^p/\|x\|_p^p \pm n^{-c}$, where c is an arbitrary constant. For $p = 0$ the corresponding formula is $(1 \pm \epsilon)/k \pm n^{-c}$, where k is the number of non-zero coordinates in x. Unless stated otherwise we assume that the failure probability is at most $1/2$. In the above definition one can consider c to be 2, but all existing constructions of L_p-samplers work for an arbitrary c with just a constant factor increase in the space, so we will not specify c in the following and ignore errors of probability n^{-c}. A sampler that outputs an approximation of x_i along with a sample coordinate i is called an *augmented sampler*. In specific, we say that an algorithm is ϵ relative error augmented sampler if it outputs a value $\hat{x_i}$ that is within $(1 \pm \epsilon)x_i$.

In this chapter we present our upper bounds as linear maps $L : \mathbb{F}^n \to \mathbb{F}^m$ where we set $\mathbb{F} = \mathbb{R}$ for the highest generality. Such linear maps can be converted to streaming algorithms assuming that all the updates are integers ($u \in \mathbb{Z}$) and the coordinates of the vector x throughout the stream remain bounded by some value $M = \text{poly}(n)$. Under these assumptions, we can set $\mathbb{F} = \mathbb{Z}_p$ for some $p > 2M$ and map the integers $\{-M, \ldots, M\}$ to elements of $\mathbb{Z}_p$. This way, maintaining $L(x)$ requires updating m counters over $\mathbb{Z}_p$ and takes $O(m \log n)$ bits with fast update time (especially since the matrices we consider are sparse). This discretization step is standard and thus we omit most details.

We say an event happens with *low probability* if the probability can be made less than n^{-c}. Here $c > 0$ is an arbitrary constant, for example one can set $c = 2$. The actual value of

c has limited effect on the space of our algorithm: it changes only the unspecified constants hidden in the O notation. We will routinely ignore low probability events, sometimes even $O(n)$ of them, which is not a problem as we leave c unspecified. Events complementary to low probability events are referred to as *high probability* events.

For $0 \leq m \leq n$ we call the vector $x \in \mathbb{R}^n$ *m-sparse* if all but at most m coordinates of x are zero. We define $\mathrm{Err}_2^m(x) = \min \|x - \hat{x}\|_2$, where $\hat{x} \in \mathbb{R}^n$ ranges over all the m-sparse vectors.

5.1.2 The sampler algorithm

We start by stating the properties of the two streaming algorithms we are going to use. Both are based on maintaining $L(x)$ for a well chosen random linear map $L : \mathbb{R}^n \to \mathbb{R}^{n'}$ with $n' < n$.

The *count-sketch* algorithm [CCFC04] is so simple we cannot resist the temptation to define it here. For parameter m, the count-sketch algorithm works as follows. It selects independent samples $h_j : [n] \to [6m]$ and $g_j : [n] \to \{1, -1\}$ from pairwise independent uniform hash families for $j \in [l]$ and $l = O(\log n)$. It computes the following linear function of x for $j \in [l]$ and $k \in [6m]$: $y_{k,j} = \sum_{i \in [n], h_j(i) = k} g_j(i) x_i$. Finally it outputs $x^* \in \mathbb{R}^n$ as an approximation of x with

$$x_i^* = \operatorname*{median}_{j \in [l]} g_j(i) y_{h(i),j}$$

for $i \in [n]$. The performance guarantee of the count-sketch algorithm is as follows. (For a compact proof see a recent survey by Gilbert and Indyk [GI10].)

Lemma 5.1 (Charikar et al. [CCFC04])**.** *For any $x \in \mathbb{R}^n$ and $m \geq 1$ we have $|x_i - x_i^*| \leq \mathrm{Err}_2^m(x)/m^{1/2}$ for all $i \in [n]$ with high probability, where x^* is the output of the count-sketch algorithm with parameter m. As a consequence we also have*

$$\mathrm{Err}_2^m(x) \leq \|x - \hat{x}\|_2 \leq 3\,\mathrm{Err}_2^m(x)$$

with high probability, where $\hat{x}$ is the m-sparse vector best approximating x^* (i.e., $\hat{x}_i = x_i^*$ for the m coordinates i with $|x_i^*|$ highest and is $\hat{x}_i = 0$ for the remaining $n - m$ coordinates).

We will also need the following result for the estimation of L_p norms.

Lemma 5.2 (Kane et al. [KNW10a]). *For any $p \in (0, 2]$ there is a streaming algorithm based on a random linear map $L : \mathbb{R}^n \to \mathbb{R}^l$ with $l = O(\log n)$ that outputs a value r computed solely from $L(x)$ that satisfies $\|x\|_p \le r \le 2\|x\|_p$ with high probability.*

Our streaming algorithm on Figure 5.1.2 makes use of a single count-sketch and two norm estimation algorithms. The count-sketch is for the randomly scaled version z of the vector x. One of the norm approximation algorithms is for $\|x\|_p$, the other one approximates $\mathrm{Err}_2^m(z)$ through the almost equal value $\|z - \hat{z}\|_2$. A standard L_2 approximation for z works if we modify z by subtracting $\hat{z}$ in the recovery stage. One can get arbitrary good approximations of $\mathrm{Err}_2^m(x)$ this way.

First we estimate the probability that the algorithm aborts at the step 5 of the recovery stage because $s > \beta m^{1/2} r$. This depends on the scaling that resulted in z and it will be important for us that the bound holds even after conditioning on any one scaling factor.

Lemma 5.3. *Conditioned on an arbitrary fixed value t of t_i for a single index $i \in [n]$ we have $\Pr[s > \beta m^{1/2} r \mid t_i = t] = \epsilon + n^{-c}$.*

Proof. First note that by Lemma 5.2 we have $r \ge \|x\|_p$ and $s \le 2\|z - \hat{z}\|_2$ with high probability. By Lemma 5.1 we have $\|z - \hat{z}\| \le 3\,\mathrm{Err}_2^m(z)$ also with high probability. We may therefore assume that all of these inequalities hold, and in particular $r \ge \|x\|_p$ and $s \le 6\,\mathrm{Err}_2^m(z)$. It is therefore enough to bound the probability that $6\,\mathrm{Err}_2^m(z) > \beta m^{1/2}\|x\|_p$.

For simplicity (and without loss of generality) we assume that the fixed scalar is $t_n = t$ and will freely use i for indexes in $[n-1]$.

Let $T = \beta\|x\|_p$. For each $i \in [n-1]$ we define two variables z_i' and z_i'' determined by z_i as follows. The indicator variable $z_i' = 1$ if $|z_i| > T$ and 0 otherwise. We set $z_i'' = z_i^2(1 - z_i')/T^2 \in [0, 1]$. Let $S' = \sum_{i\in[n-1]} z_i'$ and $S'' = \sum_{i\in[n-1]} z_i''$. Note that $T^2 S'' = \|z - w\|_2^2$, where w is defined by $w_i = z_i z_i'$ for $i \in [n-1]$ and $w_n = z_n$. Here w is $(S'+1)$-sparse, so we have $\mathrm{Err}_2^m(z) \le TS''^{1/2}$ unless $S' \ge m$. It is therefore enough to bound the probabilities of the events $S' \ge m$ and $S'' > m\beta^2\|x\|_p^2/(6T)^2 = m/36$, each with $\epsilon/2$.

We have $\mathbb{E}[z_i'] = |x_i|^p/T^p$, $\mathbb{E}[S'] \le \beta^{-p} = \epsilon^{1-p}$. Since $z_1, \ldots, z_{i-1}$ are k-wise independent,

by concentration bounds of Schmidt et al. (see Theorem 2.3) we have

$$\Pr[S' \geq m] < (e\epsilon^{1-p}/m)^k < \epsilon/2$$

$k = 2\lceil \log(2/\epsilon) \rceil$. The calculation for S'' is similar. We have

$$\mathbb{E}[z_i''] < \int_{|x_i|^p/T^p}^{\infty} x_i^2 t^{-2/p} T^{-2} dt = \frac{p}{2-p} |x_i|^p T^{-p}.$$

Thus $\mathbb{E}[S''] \leq \frac{p}{2-p}\|x\|_p^p T^{-p} = \frac{p}{2-p}\epsilon^{1-p}$. Note that the z_i'' are k-wise independent random variables from $[0,1]$ and hence by Theorem 2.4 we get

$$\Pr[S'' > m/36] < e^{-\lfloor k \rfloor/2} \leq \epsilon/2$$

by our choice of k. This completes the proof of the lemma. □

The fact that our algorithm is an approximate L_p-sampler with both relative error and success probability ϵ follows from the following lemma.

Lemma 5.4. *The probability that the algorithm of Figure 5.1.2 outputs the index $i \in [n]$ conditioned on a fixed value for $r \geq \|x\|_p$ is $(\epsilon \pm \epsilon^2)|x_i|^p/r^p \pm n^{-c}$. The relative error of the estimate for x_i is at most ϵ with high probability.*

Proof. Optimally, we would output $i \in [n]$ if $|z_i| > \epsilon^{-1/p} r$. This happens if $t_i < \epsilon |x_i|^p/r^p$ and has probability exactly $\epsilon|x_i|^p/r^p$. We have to estimate the probability that something goes wrong and the algorithm outputs i when this simple condition is not met or vice versa.

Three things can go wrong. First, if $s > m^{1/2}\beta r$ the algorithm fails. This is only a problem for our calculation if it should, in fact, output the index i. Lemma 5.3 bounds the conditional probability of this happening.

Having dealt with the $s > \beta m^{1/2} r$ case we may assume now $s \leq \beta m^{1/2} r$. We also make the assumptions (high probability by Lemma 5.2) that $\|z - \hat{z}\|_2 \leq s$ and thus $\mathrm{Err}_2^m(z) \leq \|z - \hat{z}\|_2 \leq s \leq \beta m^{1/2} r$. Finally, we also assume $|z_i^* - z_i| \leq \mathrm{Err}_2^m(z)/m^{1/2} \leq \beta r$ for all $i \in [n]$. This is satisfied with high probability by Lemma 5.1.

A second source of error comes from this βr possible difference between z_i^* and z_i. This can only make a problem if t_i is close to the threshold, namely $(\epsilon^{-1/p} + \beta)^{-p}|x_i|^p/r^p \leq t_i \leq (\epsilon^{-1/p} - \beta)^{-p}|x_i|^p/r^p$. The probability of selecting t_i from this interval is $O(\beta/\epsilon^{1+1/p}|x_i|^p/r^p) = O(\epsilon^2|x_i|^p/r^p)$ as required.

Initialization Stage:

1. Set $k = 2\lceil \log(2/\epsilon) \rceil$.
2. For $p = 1$, set $m = O(\log 1/\epsilon)$ and for $p \neq 1$, set $m = O(\epsilon^{-\max(0,p-1)})$ with large enough constants.
3. Set $\beta = \epsilon^{1-1/p}$ and $l = O(\log n)$ with a large enough constant factor.
4. Select k-wise independent uniform scaling factors $t_i \in [0, 1]$ for $i \in [n]$.
5. Select the appropriate random linear functions for the execution of the count-sketch algorithm and L and L' for the norm estimations in the processing stage.

Processing Stage:

1. Use count-sketch with parameter m for the scaled vector $z \in \mathbb{R}^n$ with $z_i = x_i/t_i^{1/p}$.
2. Maintain a linear sketch $L(x)$ as needed for the L_p norm approximation of x.
3. Maintain a linear sketch $L'(z)$ as needed for the L_2 norm estimation of z.

Recovery Stage:

1. Compute the output z^* of the count-sketch and its best m-sparse approximation $\hat{z}$.
2. Based on $L(x)$ compute a real r with $\|x\|_p \leq r \leq 2\|x\|_p$.
3. Based on $L'(z - \hat{z})$ compute a real s with $\|z - \hat{z}\|_2 \leq s \leq 2\|z - \hat{z}\|_2$.
4. Find i with $|z_i^*|$ maximal.
5. If $s > \beta m^{1/2} r$ or $|z_i^*| < \epsilon^{-1/p} r$ output FAIL.
6. Output i as the sample and $z_i^* t_i^{1/p}$ as an approximation for x_i.

Figure 5.1: Our L_p-sampler with both success probability and relative error $\Theta(\epsilon)$

Finally, the third source of error comes from the possibility that i should be output based on $|z_i| > \epsilon^{-1/p} r$, yet we output another index $i' \neq i$ because $z_{i'}^* \geq z_i^*$. In this case we must have $t_{i'} < (\epsilon^{-1/p} - \beta)^{-p} |x_i|^p / r^p$. This has probability $O(\epsilon |x_{i'}|^p / r^p)$. By the union bound the probability that such an index i' exists is $O(\epsilon \|x\|_p^p / r^p) = O(\epsilon)$. Pairwise independence is enough to conclude that the same bound holds after conditioning on $|z_i| > \epsilon^{-1/p} r$. This finishes the proof of the first statement of the lemma.

The algorithm only outputs an index i if $s \leq \beta m^{1/2} r$ and $|z_i^*| \leq \epsilon^{-1/p} r$. The first implies that the absolute approximation error for z_i is at most βr, while the second lower bounds the absolute value of the approximation itself by $\epsilon^{-1/p} r$, thus ensuring a $\beta \epsilon^{1/p} = \epsilon$ relative error approximation. Our approximation for $x_i = z_i t_i^{1/p}$ is $z_i^* t^{1/p}$, so the relative error is the same. Note that the inverse polynomial error probability comes from the various low probability events we neglected. The same is true for the additive error term in the distribution. □

The next theorem describes our final L_p-sampling algorithm as well as its space and error bounds.

Theorem 5.1. *For $\delta > 0$ and $\epsilon > 0$, $0 < p < 2$ there is an $O(\epsilon)$ relative error one pass augmented L_p-sampling algorithm with failing probability at most δ and having low probability that the relative error of the estimate for the selected coordinate is more than ϵ. The algorithm uses $O_p(\epsilon^{-\max(1,p)} \log^2 n \log(1/\delta))$ space for $p \neq 1$ while for $p = 1$ the space is $O(\epsilon^{-1} \log(1/\epsilon) \log^2 n \log(1/\delta))$.*

Proof. Using Lemma 5.4 and the fact that $\|x\|_p \leq r \leq 2\|x\|_p$ with high probability one obtains that the failure probability of the algorithm in Figure 5.1.2 is at most $1 - \epsilon/2^p + n^{-c}$. Conditioning on obtaining an output, returning i has probability $(1 + O(\epsilon))|x_i|^p/\|x\|_p^p + n^{-c}$. Clearly, the latter statement remains true for any number of repetitions and the failure probability is raised to power v for v repetitions. Thus using $v = O(\log(1/\delta)/\epsilon)$ repetitions (taking the first non-failing output), the algorithm is an $O(\epsilon)$ relative error δ failure probability L_p-sampling algorithm. Here we assume $v < n$ as otherwise recording the entire vector x is more efficient.

The low probability of more than ϵ relative error in estimating x_i also follows from Lemma 5.4. In one round, the algorithm on Figure 5.1.2 uses $O(m \log n)$ counters for the count-sketch and this dominates the counters for the norm estimators. Using standard discretization this can be turned into an $O(m \log^2 n)$ bit algorithm. For the discretization we also have to keep our scaling factors polynomial in n. Recall that in the continuous model these factors $t_i^{-1/p}$ were unbounded. But we can safely declare failure if $t_i^{-1} > n^c$ for some $i \in [n]$ as this has low probability n^{1-c}. We have to do the v repetitions of the

algorithm in parallel to obtain a single pass streaming algorithm. This increases the space to $O(vm \log^2 n)$ which is the same as the one claimed in the theorem. □

Note that the hidden constant in the space bound of the theorem depends on p, especially that $1/(2-p)$, $1/p$ and $1/|1-p|$ factors come in. The last can always be replaced by a $\log(1/\epsilon)$ factor but the former ones are harder to handle. For $p = 2$ an extra $\log n$ factor seems to be necessary for an algorithm along these lines, see [AKO10].

As we will see in Theorem 5.7, our space bound is tight for ϵ and δ constants. Note that the lower bound holds even if we only require the overall distribution of the L_p-sampler to be close to the L_p distribution as opposed to the much more strict definition of ϵ relative error sampling.

5.2 The L_0 Sampler

For p near zero, the method of precision sampling becomes intractable. This is because our scaling factors are $t_i^{-1/p}$ which clearly rules out $p = 0$. In the following we present a L_0-sampler using a different approach.

Theorem 5.2. *There exists a zero relative error L_0 sampler which uses $O(\log^2 n \log(1/\delta))$ bits and outputs a coordinate $i \in [n]$ with probability at least $1 - \delta$.*

Proof. We first present our algorithm assuming a random oracle, and then we remove this assumption through the use of the pseudo-random generator of Nisan [Nis90]. Let I_k for $k = 1, \ldots, \lfloor \log n \rfloor$ be subsets of $[n]$ of size 2^k chosen uniformly at random and $I_0 = [n]$. For each k we run the sparse recovery procedure of Theorem 4.11 on the vector x restricted to the coordinates in I_k with s set to $\lceil 4 \log(1/\delta) \rceil$. We return a uniform random non-zero coordinate from the first recovery that gives a non-zero s-sparse vector. The algorithm fails if each recovery algorithm returns zero or DENSE.

Let J be the set of coordinates i with $x_i \neq 0$ (the support of x). Disregarding the low probability error of the procedure in Theorem **??** this procedure returns each index $i \in J$ with equal probability and never returns an index outside J. To bound the failure probability we observe that for $|J| \leq s$ failure is not possible, while for $|J| > s$ one has $k \in [\lfloor \log n \rfloor]$ such that $\mathbb{E}[|I_k \cap J|] = 2^k |J|/n$ is between $s/3$ and $2s/3$. For this k alone

$1 \leq |I_k \cap J| \leq s$ is satisfied with probability at least $1 - \delta$ by the Chernoff bound limiting failure probability by δ.

To get rid of the random oracle we use Nisan's generator [Nis90] that produces the random bits for the algorithm (including the ones describing I_k and the ones for the eventual random choice from $I_k \cap J$) from an $O(\log^2 n)$ length seed. It fools every logspace tester including the one that tests for a fixed set $J \subseteq [n]$ and $i \in [n]$ if the algorithm (assuming correct reconstruction) would return i. Thus this version of the algorithm, now using $O(\log^2 n)$ random bits and $O(\log^2 \log(1/\delta))$ total space, is also a zero relative error L_0-sampler with failure probability bounded by $\delta + O(n^{-c})$. □

As we shall see in Section 5.4, this space bound is also tight for δ a constant and better sampling is not possible even if we allow constant relative error or a small overall distance of the output from the L_0 distribution.

5.3 Algorithms for finding a duplicate

Recall that the `DUPLICATE` problem asks to report a repeating item in the array $x_1, \ldots, x_{n+1}$ where all $x_i \in [n]$. A duplicate can be found using $O(\log n)$ bits of space in $O(\log n)$ passes deterministically as follows. Let $h = \lceil n/2 \rceil$ and in the first pass count the number of i such that $a_i \in [h]$. This can be done using $O(\log n)$ space simply by incrementing a counter whenever the new item is no bigger than h. If the final count is bigger than h, by the pigeon-hole principle, there exists a duplicate in the array which is between 1 and h. Hence in the subsequent passes, we can safely discard elements bigger than h. Otherwise, there are more than $n - h$ items in $[h + 1, n]$ and in the subsequent passes we disregard elements that are in $[h]$. Bisecting the alphabet $[n]$ in each pass similarly gives us a duplicate after $O(\log n)$ rounds.

This algorithm can be generalized to work in only p passes while taking $O(n^{1/p} \log^{1-1/p} n)$ space as follows. In each of the first $p-1$ passes, divide the alphabet into $K = \lceil (n/\log n)^{1/p} \rceil$ blocks and count how many items appear in the stream from each of these blocks. By the pigeon-hole principle, there exists a block b which has more than $K + 1$ appearances in the stream. In the next pass define the alphabet to be the items in block b. It can be verified

that after pass $p-1$, the size of the alphabet has been reduced to $O(K \log n)$. In the last pass, we can simply keep a bit vector of size $O(K \log n)$ to find a duplicate. Overall, we use $O(n^{1/p} \log^{1-1/p} n)$ bits in each pass as desired. In the following we show that randomization can help to get exponentially better bounds for detecting a duplicate.

5.3.1 A $\Theta(\log^2 n)$ space bound for DUPLICATE

In this section we show a $O(\log^2 n)$ space algorithm via a direct application of our L_1 sampler for the duplicate problem.

Let x be an n-dimensional vector, initially zero at each coordinate. We run the L_1-sampler of Theorem 5.1 on x, with both relative error and failure probability set to $1/2$. Before we start processing the stream, we subtract 1 from each coordinate of x; i.e., we feed the updates $(i, -1)$ for $i = 1, \ldots n$ to the L_1 sampling algorithm. When a stream item $i \in [n]$ comes, we increase x_i by 1; i.e., we generate the update $(i, 1)$.

Observe that when the stream is exhausted, we have $x_i \geq 1$ for items i that have at least two occurrences in the stream, $x_i = 0$ for items that appear once, and $x_i = -1$ for items that do not appear. Note that our L_1-sampler, if it does not fail, outputs an index i and an approximation x^* of x_i. If x^* is positive, we output i, if it is negative or the L_1-sampler fails, we output FAIL. We have $\sum_{i=1}^{n} x_i = 1$, hence a perfect L_1 sample from x is positive with more than half probability. Taking into account that our L_1-sampler has $1/2$ relative error and failure probability (and neglecting for a second the chance that x^* has different sign from x_i) we conclude that we output a duplicate with probability at least $1/4$. The event that x^* does not have the same sign as x_i (and thus the relative error is at least 1) has low probability. This low probability can increase the failure probability and/or introduce error when we output non-duplicate items.

Repeating the algorithm $O(\log(1/\delta))$ times in parallel and accepting the first non-failing output reduces the failure rate to δ but keeps the error rate low.

Theorem 5.3. *For any $\delta > 0$ there is a $O(\log^2 n \log(1/\delta))$ space one-pass algorithm which, given a stream of length $n+1$ over the alphabet $[n]$, outputs an $i \in [n]$ or FAIL, such that the probability of outputting FAIL is at most δ and the algorithm outputs a letter $i \in [n]$ that is no duplicate with low probability.*

This space bound is best possible for $\delta < 1$ a constant, as shown in the following theorem.

Theorem 5.4. *Any one-pass streaming algorithm that outputs a duplicate with constant probability uses $\Omega(\log^2 n)$ space. This remains true even if the stream is not allowed to have an element repeated more than twice.*

Proof. We show our claim by a reduction from the universal relation problem (see Section 5.4.2). Each of Alice and Bob is given a binary string of length n, respectively x and y. Further, the players are guaranteed that $x \neq y$. Alice sends a message to Bob, after which Bob must output an index $i \in [n]$ such that $x_i \neq y_i$. By Theorem 5.10, solving this problem with any constant error probability requires $\Omega(\log^2 n)$ bits for one-way communication. The players solve the given instance of universal relation problem using a small space finding duplicates algorithm as follows. Alice constructs the set $S = \{2i - 1 + x_i \mid i \in [n]\} \subseteq [2n]$ and Bob constructs $T = \{2i - y_i \mid i \in [n]\} \subseteq [2n]$. Observe that $|S| = |T| = n$ and $x_i \neq y_i$ if and only if either $2i$ or $2i - 1$ is in both S and T.

Next, using the shared randomness, players pick a random subset P of $[2n]$ of size n. We have

$$\Pr[|S \cap P| + |T \cap P| \geq n + 1] > 1/8.$$

To see this, let $i \in S \cap T$ and $j \in [2n] \setminus (S \cap T)$. We have $|P \cap \{i, j\}| = 1$ with probability more than $1/2$. The sets P satisfying this can be partitioned into classes of size four by putting $Q \cup \{i\}$, $Q \cup \{j\}$ and their complements in the same class for any $Q \subseteq [2n] \setminus \{i, j\}$, $|Q| = n-1$. Clearly, at least one of the four sets P in each class satisfies $|S \cap P| + |T \cap P| > n$.

Given a streaming algorithm A for finding duplicates, Alice feeds the elements of $S \cap P$ to A and sends the memory contents over to Bob, along with the integer $|S \cap P|$. If $|S \cap P| + |T \cap P| < n + 1$, Bob outputs FAIL. Otherwise, feeds arbitrary $n + 1 - |S \cap P|$ elements of $T \cap P$ to A. Note that no element repeats more than twice.

On the other hand $|P| = n$ and we always give $n + 1$ elements of P to the algorithm. Also with constant probability, Bob finds an $a \in S \cap T$, which in turn reveals an i such that $x_i \neq y_i$. Therefore by Theorem 5.10, any algorithm for finding duplicates must use $\Omega(\log^2 n)$ bits. □

5.3.2 Finding duplicates in short streams

Now we turn our attention to finding duplicates under weaker guarantees than above. Assume that we have a stream of length $n-s \leq n$ over the alphabet $[n]$ and we want to output a duplicate, if one exists. For this problem, Gopalan et al. [GR09] gave a $O(s \log^3 n)$ space algorithm which finds a duplicate if one exists or reports that none exists with constant error probability. Further, they showed that any such algorithm must use $\Omega(s)$ bits of space. When $s = \Omega(n)$ the problem requires $\Omega(n)$ bits by the former bound and a matching upper bound can be achieved by recording the entire stream in memory. Hence from now on, for simplicity, we assume $s \leq n^{1-\epsilon}$ for some $\epsilon > 0$.

In this section, we give an algorithm for this problem which uses $O(s \log n + \log^2 n)$ space and finds a duplicate, if one exists, with constant probability. If there is no duplicate, our algorithm reports so with probability 1. Moreover, we prove in Theorem 5.6 that the space bound of our algorithm is best possible.

Theorem 5.5. *For any $\delta > 0$ there is an $O(s \log n + \log^2 n \log(1/\delta))$ space one-pass algorithm which, given a stream of length $n-s$ over the alphabet $[n]$, outputs NO-DUPLICATE with probability 1 if the input sequence has no duplicates, otherwise it outputs $i \in [n]$ or reports FAIL. The returned number is a duplicate with high probability while the probability of returning FAIL is at most δ.*

Proof. Let x be an n-dimensional vector updated according to the description in the proof of Theorem 5.3; i.e., x_i is one less than the number of times i appears in the stream. In parallel, we run the exact recovery procedure of Theorem 4.11 with parameter $5s$ and the $1/2$ relative error L_1-sampler of Theorem 5.1 with failure rate $1/2$, both on the vector x. If the recovery algorithm returns a vector (as opposed to DENSE) we proceed and output a positive coordinate of the vector assuming the sparse recovery algorithm did not err. On the other hand, if the sparse recovery algorithm outputs DENSE, we consider the output of the sampling algorithm. If it is (i, x^*) with $x^* > 0$ we report i as a duplicate otherwise (if $x^* \leq 0$ or the sampling algorithm fails) we output FAIL. Define

$$\|x\|_1^+ = \sum_{i:x_i>0} |x_i| \qquad \text{and} \qquad \|x\|_1^- = \sum_{i:x_i<0} |x_i|.$$

Note that $\|x\|_1^+ - \|x\|_1^- = \sum_{i=1}^n x_i = -s$. If $\|x\|_1^+ + \|x\|_1^- \leq 5s$, then x is $5s$-sparse, thus the sparse recovery procedure outputs x and the algorithm makes no error. Note that the no repetition case falls into this category. If, however, $\|x\|_1^+ + \|x\|_1^- > 5s$, then the probability that a perfect L_1 sample from x is positive is $\|x\|_1^+/\|x\|_1 > 2/5$. Taking into account the relative error and failing probability (but ignoring the low probability event of the sampler outputting a wrong sign or sparse recovery algorithm reporting a vector), we conclude that with probability at least $1/10$ we get a positive sample and a correct output, otherwise we output FAIL. The failure probability can be decreased to δ by $O(\log(1/\delta))$ independent repetitions of the sampler. Note that the sparse recovery does not have to be repeated as it has low error probability.

The sparse recovery procedure takes $O(s \log n)$ bits by Theorem 4.11 for $s > 0$ (it takes $O(\log n)$ bits for $s = 0$) and each instance of the L_1-sampler requires $O(\log^2 n)$ bits by Theorem 5.4, totalling $O(s \log n + \log^2 n \log(1/\delta))$ bits. □

We remark the upper bounds given in the above theorem and Theorem 5.3 can be stated in a bit more general form. Instead of considering repetitions in data streams one can consider the problem of finding an index i with $x_i > 0$ for a vector $x \in \mathbb{Z}^n$ given by an update stream. Let $s = -\sum_{i=1}^n x_i$. If $s < 0$, then a positive coordinate exists and the algorithm of Theorem 5.3 finds one using $O(\log^2 n \log(1/\delta))$ space with low error and at most δ failure probability. If $s \geq 0$ a positive coordinate does not necessarily exist, but the algorithm of Theorem 5.1 finds one, report none exists or fails, with the error and failure bounds claimed there using $O(s \log n + \log^2 n \log(1/\delta))$ bits.

In the next theorem we show that our algorithm for length $n-s$ streams is best possible.

Theorem 5.6. *Any one-pass streaming algorithm that outputs a duplicate in a length $n-s$ stream uses $\Omega(s \log n + \log^2 n)$ space. This remains true even if the stream is not allowed to have an element repeated more than twice.*

Proof. Observe that an instance of size $n - s - 1$ of the original duplicates problem is an instance of the duplicate problem with length $n-s$ streams. Therefore, Theorem 5.4 implies a $\Omega(\log^2(n-s))$ bound, which is $\Omega(\log^2 n)$ by our assumption $s \leq n^{1-\epsilon}$.

We obtain the $\Omega(s \log n)$ bound by a reduction from the FCE_s^m problem, where we set $2m + s = n$. In FCE_s^m, two players, Alice and Bob are given m-subsets $S, T \in [2m + s]$.

A streaming algorithm A for the duplicates problem at hand can be used to solve FCE as follows. Alice feeds the elements of S to A and passes memory contents to Bob. Bob feeds the elements of T to A which, in turn, outputs duplicate. Therefore Bob learns an element in $S \cap T$ and since $R^1(\text{FCE}_s^m) = \Omega(s \log(m/s))$ (cf. [Mer]) and hence the algorithm A must use $\Omega(s \log n)$ bits of space, by our assumption $s \leq n^{1-\epsilon}$. Observe that no item is given more than twice to the algorithm. This completes the proof. □

5.4 Lower bounds for L_p samplers

Our first result in this section is a $\Omega(\log^2 n)$ lower bound for L_p-sampling for all p. This result implies that our L_0 sampler is optimal (up to constant factors) when $\delta < 1$ a constant and that our L_p sampler is optimal for $\delta, \epsilon < 1$ constants. Our lower bounds are obtained via reduction from two known 2-player communication games: Augmented Indexing and Universal relations. For more elaborate discussions and corresponding upper bounds on these problems, we refer the reader to the thesis [Mer].

Theorem 5.7. *Any one pass L_p-sampler with an output distribution, whose variation distance from the L_p distribution corresponding to x is at most $1/3$, requires $\Omega(\log^2 n)$ bits of memory. This holds even when all the coordinates of x are guaranteed to be -1, 0 or 1.*

For constants $\delta < 1$ and $\epsilon < 1$ the same lower bound holds for any ϵ relative error L_p-sampler with failure probability δ.

Proof. We establish the correctness of the claim by a reduction from the Universal Relation problem. In this problem two players Alice and Bob are given bit strings $u, v \in \{0,1\}$ and are required to output an i such that $u_i \neq v_i$.

Given a one-pass L_p-sampler with space S, the players can solve the universal relation by communicating S bits in one-round as follows. For each $j \in [n]$ such that $u_j = 1$, Alice feeds the update $(j, 1)$ to the streaming algorithm. Then she sends the memory contents of the algorithm to Bob. Bob resumes the algorithm with the memory Alice sent and for each $j \in [n]$ for which $v_j = 1$, he feeds the update $(j, -1)$. Let x be the vector implicitly defined by the update stream. Treating u, v as vectors in $\mathbb{Z}^n$, we have $x = u - v$. Note that the L_p distribution for x puts weight only on coordinates where u and v differ. Hence, if the

streaming algorithm at hand is an ϵ-relative error L_p-sampler or has an output distribution with small variation distance to L_p-distribution of x, Bob learns an i such that $u_i \neq v_i$ with constant probability. Hence by Theorem 5.4, $S = \Omega(\log^2 n)$, as required. □

The next theorem shows that the $1/\epsilon^p$ factor in the space usage of our augmented L_p sampler is unavoidable.

Theorem 5.8. *Any one pass augmented L_p-sampler with ϵ relative error requires $\Omega(\epsilon^{-p} \log n)$ space.*

Proof. We show the claim by a reduction from the binary augmented indexing problem. Assume Alice is given a 0-1 vector u of length n and Bob is given an integer $i \in [n]$ and a 0-1 vector v such that $v_j = u_j$ for $j < i$ and $v_j = 0$ for $j \geq i$. The goal of Bob is to find out u_i.

Suppose we have a one-pass ϵ relative error augmented L_p-sampling algorithm which uses S bits of space. Using this algorithm we give a one-round S bits protocol for the augmented indexing problem. Let $n = st$ and consider the coordinates of u and v as partitioned into s blocks of size t. Set $b = \lceil 2^{1/p} \rceil$. For each $j = 1, \ldots, s$, Alice multiplies the coordinates of u in block j by b^{s-j} and Bob multiplies the coordinates of v in block j by b^{s-j}. Then Alice she generates the updates (j, u_j) for $j \in [n]$ and sends the memory contents of the algorithm to Bob. Bob generates the updates $(j, -v_j)$ for $j \in [n]$ so as to make first $i - 1$ coordinates of x zero. Then Bob generates the update $(i, 3b^{s-\lceil i/t \rceil} t^{1/p})$. Let x be the vector defined by the updates. By construction, with constant probability, the ith coordinate will be sampled. Furthermore, if the algorithm returns a $(1 \pm t^{-1/p})$ approximation to x_i, Bob can recover the initial value of u_i. Hence by Theorem 5.9, $S = \Omega(st)$. Setting $s = \log_b n$ and $t = \epsilon^{-p}$ completes the proof, as p is constant. □

5.4.1 Augmented Indexing problem

Consider the following two-player communication game. The first player, Alice, is given a string $x \in [m]^n$ and the second player, Bob, is given an integer $i \in [n]$ and the copies of x_j for $j < i$. The players exchange messages, with Alice sending the first message, and the last player to receive a message should output x_i. We refer to this problem as the *augmented indexing* and denote it by $\mathrm{AIND}^{\mathrm{n}}_{\mathrm{m}}$.

The augmented indexing problem is well studied for binary alphabets, i.e., for the $m = 2$ case. This case was first investigated in [MNSW95] by Miltersen et al.[1] who showed that any one-round randomized protocol for the problem must communicate $\Omega(n)$ bits. Later on in [BYJKK04], Bar-Yossef et al. gave the optimal $(1 - \mathrm{H}_2(\delta))n$ bits lower bound for one-round protocols.

Here we show a lower bound of $\Omega((1-\delta)n \log m)$ on the δ-error randomized communication complexity of $\mathrm{AIND}^{\mathrm{n}}_{\mathrm{m}}$.

Theorem 5.9. *For $m \geq 3$ and $0 \leq \delta < 1 - e/m^{1-\epsilon}$ for some $\epsilon > 0$, we have $R^1_\delta(\mathrm{AIND}^{\mathrm{n}}_{\mathrm{m}}) = \Omega((1-\delta)n \log m)$.*

Proof. Our hard distribution is as follows. We give Alice a string X chosen uniformly at random from $[m]^n$. We give Bob a uniformly random integer I from $[n]$ and the prefix of X of length $I - 1$. Assume there is an δ-error randomized protocol for this problem. Let M denote the message Alice sends when the inputs are drawn from our hard distribution. Note that M is a random variable and the randomness is over both the input distribution and the shared random string R. By Fano's inequality (see Lemma 2.4)

$$\begin{aligned}
\mathrm{H}_2(\delta) + \delta \log(m-1) &\geq \mathrm{H}(X_I \,|\, M, R, X_1 X_2 \ldots X_{I-1}, I) && (5.1)\\
&= \frac{1}{n} \sum_{i=1}^{n} \mathrm{H}(X_I \,|\, M, R, X_1 X_2 \ldots X_{I-1}, I = i) && (5.2)\\
&= \frac{1}{n} \sum_{i=1}^{n} \mathrm{H}(X_i \,|\, M, R, X_1 X_2 \ldots X_{i-1}) && (5.3)\\
&= \frac{1}{n} \mathrm{H}(X \,|\, M, R) && (5.4)\\
&\geq \frac{1}{n} \left(\mathrm{H}(X \,|\, R) - \mathrm{H}(M) \right). && (5.5)
\end{aligned}$$

In steps (5.4) and (5.5) we have used the chain rule for entropy (cf. Lemma 2.3 (iii)). Since X is an element of $[m]^n$ chosen uniformly at random and independently from R, we have $\mathrm{H}(X \,|\, R) = n \log m$. Arranging, we obtain

$$R^1_\delta(\mathrm{AIND}^n) \geq \mathrm{H}(M) \geq n \log m - \delta n \log(m-1) - \mathrm{H}_2(\delta) n \qquad (5.6)$$

as desired. □

[1] We note that they study the much more powerful round-elimination concept, however their results imply non-trivial lower bounds for augmented indexing when m is a constant only.

5.4.2 Universal Relation problem

Consider the following two player communication game. Alice gets a string $x \in \{0,1\}^n$, and Bob gets $y \in \{0,1\}^n$ with the promise that $x \neq y$. The players exchange messages and the last player to receive a message should output an index $i \in [n]$ such that $x_i \neq y_i$. We call this the *universal relation communication problem* and denote it by $\mathrm{UR^n}$.

This relation has been studied in detail for deterministic communication, as it naturally arises in the context of Karchmer-Wigderson games [KW88, Kar89]. We note however that our definition is slightly unusual: in most settings both players must obtain the same index i such that $x_i \neq y_i$, whereas we are satisfied with the last player to receive a message learning such an i. Clearly, the stronger requirement can be met in $\lceil \log n \rceil$ additional bits and one additional round. The additional bits are needed in the deterministic case [TZ97] but we are not concerned with $O(\log n)$ terms for our bounds, so the two models are almost equivalent up to the shift of one in the number of rounds.

Note that, we can trivially obtain a $n + \lceil \log n \rceil$ bits protocol for $\mathrm{UR^n}$ in which both players learn a difference. Alice simply sends her entire input to Bob and Bob replies with the index of a difference. It turns out, however, one can do significantly better. The best deterministic protocol for $\mathrm{UR^n}$ is due to Tardos and Zwick [TZ97]. Improving a previous result by Karchmer [Kar89], they gave a 3 round deterministic protocol using $n + 2$ bits of communication with both players learning the same index i and showed that $n + 1$ bits is necessary for such a protocol. They also gave an $n - \lfloor \log n \rfloor + 2$ bit 2 round deterministic protocol for our weaker version of the problem, which is also tight except for the $+2$ term. They also gave an $n - \lfloor \log n \rfloor + 4$ bit 4 round protocol, where both players find an index where x and y differ—but not necessarily the same index. It follows that finding the same difference is harder for deterministic communication.

We conclude this section with a lower bound on the randomized complexity of the universal relation problem. For an almost matching lower bounds see [Mer].

Theorem 5.10. *For any $\delta < 1$ we have $R^1_\delta(\mathrm{UR^n}) = \Omega((1-\delta)\log^2 n)$.*

Proof. Suppose Alice and Bob want to solve the Augmented Indexing problem with Alice receiving $z \in [2^t]^s$ and Bob getting $i \in [s]$ and z_j for $j < i$.

Let them construct real vectors u and v as follows. Let $e_q \in \mathbb{R}^{2^t}$ be the standard unit vector in the direction of coordinate $1 \leq q \leq 2^t$. Alice forms the vectors w_j by concatenating 2^{s-j} copies of e_{z_j}, then she forms u by concatenating these vectors w_j for $j \in [s]$. The dimension of u is $n = (2^s - 1)2^t$. Bob obtains v by concatenating the same vectors w_j for $j \in [i-1]$ (these are known to him) and then concatenating enough zeros to reach the same dimension n.

Then, using the shared randomness, the players pick a length n permutation π uniformly at random and permute the coordinates of their vectors according to π. Now Alice and Bob perform the $R^1_\delta(\mathrm{UR}^{\mathrm{n}})$-length δ-error one-round protocol for UR^{n}. Suppose the protocol does not err and it returns the coordinate r. By construction, $\pi^{-1}(r)$ is a uniform random index where u and v differ. Note that each such index reveals one coordinate $z_j \in [2^t]$ to Bob for $j \geq i$. As z_j is revealed by 2^{s-j} such indices, more than half the time when the UR^{n} protocol does not err Bob learns the correct value of z_i. This yields a $R^1_\delta(\mathrm{UR}^{\mathrm{n}})$-length one-way protocol for the augmented indexing problem with error probability $(1+\delta)/2$. By Theorem 5.9 we have $R^1_\delta(\mathrm{UR}^{\mathrm{n}}) = \Omega((1-\delta)st)$. Choosing $s = t$ proves the theorem. $\square$

Bibliography

[ACCL07] Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu, *Estimating the distance to a monotone function*, Random Struct. Algorithms **31** (2007), no. 3, 371–383.

[AD99] David Aldous and Persi Diaconis, *Longest increasing subsequences: from patience sorting to the baik-deift-johansson theorem*, Bull. Amer. Math. Soc. **36** (1999), no. 3, 413–432.

[AELL11] Amihood Amir, Estrella Eisenberg, Avivit Levy, and Noa Lewenstein, *Closest periodic vectors in l p spaces*, ISAAC, 2011, pp. 714–723.

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor, *Analyzing graph structure via linear measurements*, SODA, 2012, pp. 459–467.

[AJKS02] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar, *Approximate counting of inversions in a data stream*, STOC, 2002, pp. 370–379.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak, *Polylogarithmic approximation for edit distance and the asymmetric query complexity*, FOCS, 2010, pp. 377–386.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*, Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '96, ACM, 1996, pp. 20–29.

[BC09] Joshua Brody and Amit Chakrabarti, *A multi-round communication lower bound for gap hamming and some consequences*, IEEE Conference on Computational Complexity, 2009, pp. 358–368.

[BDM02] Brian Babcock, Mayur Datar, and Rajeev Motwani, *Sampling from a moving window over streaming data*, SODA, 2002, pp. 633–634.

[BG11] Dany Breslauer and Zvi Galil, *Real-time streaming string-matching*, CPM, 2011, pp. 162–172.

[BKMT03] Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang, *Bounds for frequency estimation of packet streams*, SIROCCO, 2003, pp. 33–42.

[BO07] Vladimir Braverman and Rafail Ostrovsky, *Smooth histograms for sliding windows*, FOCS, 2007, pp. 283–293.

[BOZ09] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo, *Optimal sampling from sliding windows*, PODS, 2009, pp. 147–156.

[BYJKK04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar, *The sketching complexity of pattern matching*, APPROX-RANDOM, 2004, pp. 261–272.

[BYJKS02] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar, *An information statistics approach to data stream and communication complexity*, FOCS, 2002, pp. 209–218.

[BYKS01] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar, *Sampling algorithms: lower bounds and applications*, Proceedings of the thirty-third annual ACM symposium on Theory of computing (New York, NY, USA), STOC '01, ACM, 2001, pp. 266–275.

[CCFC04] Moses Charikar, Kevin Chen, and Martin Farach-Colton, *Finding frequent items in data streams*, Theor. Comput. Sci. **312** (2004), no. 1, 3–15.

[CCM08] Amit Chakrabarti, Graham Cormode, and Andrew McGregor, *Robust lower bounds for communication and stream computation*, STOC, 2008, pp. 641–650.

[CDK+09] Edith Cohen, Nick G. Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup, *Stream sampling for variance-optimal estimation of subset sums*, SODA, 2009, pp. 1255–1264.

[CG00] Artur Czumaj and Leszek Gasieniec, *On the complexity of determining the period of a string*, CPM, 2000, pp. 412–422.

[CH10] Graham Cormode and Marios Hadjieleftheriou, *Methods for finding frequent items in data streams*, VLDB J. **19** (2010), no. 1, 3–20.

[Che52] Herman Chernoff, *A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations*, Annals of Mathematical Statistics **23** (1952), 409–507.

[CKS03] Amit Chakrabarti, Subhash Khot, and Xiaodong Sun, *Near-optimal lower bounds on the multi-party communication complexity of set disjointness*, IEEE Conference on Computational Complexity, 2003, pp. 107–117.

[CM02] Graham Cormode and S. Muthukrishnan, *The string edit distance matching problem with moves*, SODA, 2002, pp. 667–676.

[CM11] Michael S. Crouch and Andrew McGregor, *Periodicity and cyclic shifts via linear sketches*, APPROX-RANDOM, 2011, pp. 158–170.

[CMS01] Graham Cormode, S. Muthukrishnan, and Süleyman Cenk Sahinalp, *Permutation editing and matching via embeddings*, ICALP, 2001, pp. 481–492.

[CMY08] Graham Cormode, S. Muthukrishnan, and Ke Yi, *Algorithms for distributed functional monitoring*, SODA, 2008, pp. 1076–1085.

[CMYZ10] Graham Cormode, S. Muthukrishnan, Ke Yi, and Qin Zhang, *Optimal sampling from distributed streams*, PODS, 2010, pp. 77–86.

[COP03] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy, *Better streaming algorithms for clustering problems*, STOC, 2003, pp. 30–39.

[CP10] Timothy M. Chan and Mihai Patrascu, *Counting inversions, offline orthogonal range counting, and related problems*, SODA, 2010, pp. 161–173.

[CR11] Amit Chakrabarti and Oded Regev, *An optimal lower bound on the communication complexity of gap-hamming-distance*, in Fortnow and Vadhan [FV11], pp. 51–60.

[CT06] Thomas M. Cover and Joy A. Thomas, *Elements of information theory (2. ed.)*, Wiley, 2006.

[DG77] Persi Diaconis and Ron Graham, *Spearman's footrule distance as a measure of disarray*, J. of Royal Statistical Society **39** (1977), no. 2, 262–268.

[DGIM02] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani, *Maintaining stream statistics over sliding windows (extended abstract)*, SODA, 2002, pp. 635–644.

[Die89] Paul F. Dietz, *Optimal algorithms for list indexing and subset rank*, WADS, 1989, pp. 39–46.

[DLT07] Nick G. Duffield, Carsten Lund, and Mikkel Thorup, *Priority sampling for estimation of arbitrary subset sums*, J. ACM **54** (2007), no. 6.

[EAE06] Mohamed G. Elfeky, Walid G. Aref, and Ahmed K. Elmagarmid, *Stagger: Periodicity mining of data streams using expanding sliding windows*, Proceedings of the Sixth International Conference on Data Mining (Washington, DC, USA), ICDM '06, IEEE Computer Society, 2006, pp. 188–199.

[ECW92] Vladimir Estivill-Castro and Derick Wood, *A survey of adaptive sorting algorithms*, ACM Comput. Surv. **24** (1992), no. 4, 441–476.

[EJ08] Funda Ergün and Hossein Jowhari, *On distance to monotonicity and longest increasing subsequence of a data stream*, SODA, 2008, pp. 730–736.

[EJS10] Funda Ergün, Hossein Jowhari, and Mert Saglam, *Periodicity in streams*, APPROX-RANDOM, 2010, pp. 545–559.

[EKK+00] Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan, *Spot-checkers*, J. Comput. Syst. Sci. **60** (2000), no. 3, 717–751.

[EMS04] Funda Ergün, S. Muthukrishnan, and Süleyman Cenk Sahinalp, *Sublinear methods for detecting periodic trends in data streams*, LATIN, 2004, pp. 16–28.

[Fan61] Robert M. Fano, *Transmission of information: A statistical theory of communication*, MIT Press, March 1961.

[FIS05] Gereon Frahling, Piotr Indyk, and Christian Sohler, *Sampling in dynamic data streams and applications*, Symposium on Computational Geometry, 2005, pp. 142–149.

[FKM+08] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang, *Graph distances in the data-stream model*, SIAM J. Comput. **38** (2008), no. 5, 1709–1727.

[FKSV02] Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan, *An approximate l1-difference algorithm for massive data streams*, SIAM J. Comput. **32** (2002), no. 1, 131–151.

[FKZ04] Joan Feigenbaum, Sampath Kannan, and Jian Zhang, *Computing diameter in the streaming and sliding-window models*, Algorithmica **41** (2004), no. 1, 25–41.

[FS05] Gereon Frahling and Christian Sohler, *Coresets in dynamic geometric data streams*, STOC, 2005, pp. 209–217.

[FV11] Lance Fortnow and Salil P. Vadhan (eds.), *Proceedings of the 43rd acm symposium on theory of computing, stoc 2011, san jose, ca, usa, 6-8 june 2011*, ACM, 2011.

[FW65] N. J. Fine and H. S. Wilf, *Uniqueness theorems for periodic functions*, Proceedings of The American Mathematical Society **16** (1965), 109–109.

[Gan11] Sumit Ganguly, *Polynomial estimators for high frequency moments*, CoRR **abs/1104.4552** (2011).

[GG07] Anna Gál and Parikshit Gopalan, *Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence*, FOCS, 2007, pp. 294–304.

[GGI+02] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss, *Fast, small-space algorithms for approximate histogram maintenance*, STOC, 2002, pp. 389–398.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron, *Property testing and its connection to learning and approximation*, J. ACM **45** (1998), no. 4, 653–750.

[GI10] Anna Gilbert and Piotr Indyk, *Sparse recovery using sparse matrices*, Proceeding of IEEE, 2010.

[GJKK07] Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar, *Estimating the sortedness of a data stream*, SODA, 2007, pp. 318–327.

[GK04] Michael Greenwald and Sanjeev Khanna, *Power-conserving computation of order-statistics over sensor networks*, PODS, 2004, pp. 275–285.

[GM06] Sudipto Guha and Andrew McGregor, *Approximate quantiles and the order of the stream*, PODS, 2006, pp. 273–279.

[GM09] ———, *Stream order and order statistics: Quantile estimation in random-order streams*, SIAM J. Comput. **38** (2009), no. 5, 2044–2059.

[GMMO00] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan, *Clustering data streams*, FOCS, 2000, pp. 359–366.

[GR09] Parikshit Gopalan and Jaikumar Radhakrishnan, *Finding duplicates in a data stream*, Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (Philadelphia, PA, USA), SODA '09, Society for Industrial and Applied Mathematics, 2009, pp. 402–411.

[GT01] Phillip B. Gibbons and Srikanta Tirthapura, *Estimating simple functions on the union of data streams*, SPAA, 2001, pp. 281–291.

[GZ03] Anupam Gupta and Francis Zane, *Counting inversions in lists*, SODA, 2003, pp. 253–254.

[Hoe63] Wassily Hoeffding, *Probability inequalities for sums of bounded random variables*, Journal of the American Statistical Association **58** (1963), no. 301, 13–30.

[IKM00] Piotr Indyk, Nick Koudas, and S. Muthukrishnan, *Identifying representative trends in massive time series data sets using sketches*, Proceedings of the 26th International Conference on Very Large Data Bases (San Francisco, CA, USA), VLDB '00, Morgan Kaufmann Publishers Inc., 2000, pp. 363–372.

[Ind00] Piotr Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, FOCS, 2000, pp. 189–197.

[IR08] Piotr Indyk and Milan Ruzic, *Near-optimal sparse recovery in the l1 norm*, FOCS, 2008, pp. 199–207.

[IW05] Piotr Indyk and David P. Woodruff, *Optimal approximations of the frequency moments of data streams*, STOC, 2005, pp. 202–208.

[Jen06] J. Jensen, *Sur les fonctions convexes et les ingalits entre les valeurs moyennes*, Acta Mathematica **30** (1906), 175–193, 10.1007/BF02418571.

[JMMV07] T. S. Jayram, Andrew McGregor, S. Muthukrishnan, and Erik Vee, *Estimating statistical aggregates on probabilistic data streams*, PODS, 2007, pp. 243–252.

[JST11] Hossein Jowhari, Mert Saglam, and Gábor Tardos, *Tight bounds for lp samplers, finding duplicates in streams, and related problems*, PODS, 2011, pp. 49–58.

[Kar89] Mauricio Karchmer, *A new approach to circuit depth*, Ph.D. thesis, MIT, 1989.

[KN97] Eyal Kushilevitz and Noam Nisan, *Communication complexity*, Cambridge University Press, 1997.

[KNPW11] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff, *Fast moment estimation in data streams in optimal space*, in Fortnow and Vadhan [FV11], pp. 745–754.

[KNW10a] Daniel M. Kane, Jelani Nelson, and David P. Woodruff, *On the exact space complexity of sketching and streaming small norms*, SODA, 2010, pp. 1161–1178.

[KNW10b] Daniel M. Kane, Jelani Nelson, and David P. Woodruff, *An optimal algorithm for the distinct elements problem*, Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems of data (New York, NY, USA), PODS '10, ACM, 2010, pp. 41–52.

[KR87] Richard M. Karp and Michael O. Rabin, *Efficient randomized pattern-matching algorithms*, IBM J. Res. Dev. **31** (1987), 249–260.

[KSP03] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou, *A simple algorithm for finding frequent elements in streams and bags*, ACM Trans. Database Syst. **28** (2003), 51–55.

[KW88] Mauricio Karchmer and Avi Wigderson, *Monotone circuits for connectivity require super-logarithmic depth*, Proceedings of the twentieth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '88, ACM, 1988, pp. 539–550.

[LLXY04] Xuemin Lin, Hongjun Lu, Jian Xu, and Jeffrey Xu Yu, *Continuously maintaining quantile summaries of the most recent n elements over a data stream*, ICDE, 2004, pp. 362–373.

[LN05] Oded Lachish and Ilan Newman, *Testing periodicity*, APPROX-RANDOM, 2005, pp. 366–377.

[LNVZ05] David Liben-Nowell, Erik Vee, and An Zhu, *Finding longest increasing and common subsequences in streaming data*, COCOON, 2005, pp. 263–272.

[LS62] R. C. Lyndon and M. P. Schutzenberger, *The equation am=bncp in a free group.*

[LS93] Gad M. Landau and Jeanette P. Schmidt, *An algorithm for approximate tandem repeats*, CPM, 1993, pp. 120–133.

[MAA05] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi, *Duplicate detection in click streams*, WWW, 2005, pp. 12–21.

[McG05] Andrew McGregor, *Finding graph matchings in data streams*, APPROX-RANDOM, 2005, pp. 170–181.

[Mer] Sağlam Mert, Master's thesis.

[MG82] Jayadev Misra and David Gries, *Finding repeated elements*, Tech. report, Ithaca, NY, USA, 1982.

[MMNW11] Darakhshan J. Mir, S. Muthukrishnan, Aleksandar Nikolov, and Rebecca N. Wright, *Pan-private algorithms via statistics on sketches*, PODS, 2011, pp. 37–48.

[MNSW95] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson, *On data structures and asymmetric communication complexity*, Proceedings of the twenty-seventh annual ACM symposium on Theory of computing (New York, NY, USA), STOC '95, ACM, 1995, pp. 103–111.

[MS03] S. Muthukrishnan and Martin Strauss, *Rangesum histograms*, SODA, 2003, pp. 233–242.

[Mut] S. Muthukrishnan, *Data streams: Algorithms and applications.*

[Mut05] S. Muthukrishnan, *Data streams: Algorithms and applications*, Foundations and Trends in Theoretical Computer Science **1** (2005), no. 2.

[MW10] Morteza Monemizadeh and David P. Woodruff, *1-pass relative-error lp-sampling with applications*, Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (Philadelphia, PA, USA), SODA '10, Society for Industrial and Applied Mathematics, 2010, pp. 1143–1160.

[Nis90] N. Nisan, *Pseudorandom generators for space-bounded computations*, Proceedings of the twenty-second annual ACM symposium on Theory of computing (New York, NY, USA), STOC '90, ACM, 1990, pp. 204–212.

[PL07] Ely Porat and Ohad Lipsky, *Improved sketching of hamming distance with error correcting*, CPM, 2007, pp. 173–182.

[PP09] Benny Porat and Ely Porat, *Exact and approximate pattern matching in the streaming model*, Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '09, IEEE Computer Society, 2009, pp. 315–323.

[RLG+10] Chotirat Ann Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn J. Keogh, Michail Vlachos, and Gautam Das, *Mining time series data*, Data Mining and Knowledge Discovery Handbook, 2010, pp. 1049–1077.

[RS11] Ronitt Rubinfeld and Asaf Shapira, *Sublinear time algorithms*, Electronic Colloquium on Computational Complexity (ECCC) **18** (2011), 13.

[SSS95] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan, *Chernoff-hoeffding bounds for applications with limited independence*, SIAM J. Discret. Math. **8** (1995), 223–250.

[Sta01] Pantelimon Stanica, *Good lower and upper bounds on binomial coefficients*, J. Inequalities in Pure and Applied Math. **2** (2001), no. 3.

[Tar07] Jun Tarui, *Finding a duplicate and a missing item in a stream*, Theory and Applications of Models of Computation (Jin-Yi Cai, S. Cooper, and Hong Zhu, eds.), Lecture Notes in Computer Science, vol. 4484, Springer Berlin / Heidelberg, 2007, pp. 128–135.

[TZ97] Gabor Tardos and Uri Zwick, *The communication complexity of the universal relation*, Proceedings of the 12th Annual IEEE Conference on Computational Complexity (Washington, DC, USA), IEEE Computer Society, 1997, pp. 247–.

[Woo11] David P. Woodruff, *Near-optimal private approximation protocols via a black box transformation*, in Fortnow and Vadhan [FV11], pp. 735–744.

[Yao79] Andrew Chi-Chih Yao, *Some complexity questions related to distributive computing(preliminary report)*, Proceedings of the eleventh annual ACM symposium on Theory of computing (New York, NY, USA), STOC '79, ACM, 1979, pp. 209–213.