TIGHT BOUNDS FOR DATA STREAM ALGORITHMS AND COMMUNICATION PROBLEMS

by

Mert Sağlam

B.A., Sabanci University, 2008

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

APPROVAL

Name:	Mert Sağlam
Degree:	Master of Science
Title of Thesis:	Tight Bounds For Data Stream Algorithms and Communication Problems

Abstract

In this thesis, we give efficient algorithms and near-tight lower bounds for the following problems in the streaming model.

Improving on the works of Monemizadeh and Woodruff from SODA'10 and Andoni, Krauthgamer and Onak from FOCS'11, we give L_p-samplers requiring $O(\epsilon^{-p}\log^2 n)$ space for $p \in (1,2)$. Our algorithm also works for $p \in [0,1]$, taking $\tilde{O}(\epsilon^{-1}\log^2 n)$ space.

As an application of our sampler, we give an $O(\log^2 n)$ space algorithm for finding duplicates in data streams, improving the algorithms of Gopalan and Radhakrishnan from SODA'09.

Given a stream that consists of a pattern of length m and a text of length n, the pattern matching problem is to output all occurrences of the pattern. Improving on the results of Porat and Porat from FOCS'09, we give a $O(\log n \log m)$ space algorithm that works entirely in the streaming model.

Finally we show several near-tight lower bounds for the above problems through new results in communication complexity.

I dedicate this thesis to Erol Bakşi.

Acknowledgements

It was my great luck to have Gábor Tardos as a supervisor. Doing research with Gábor is an amazing experience with many privileges. Among these, a clear winner for me was to watch Gábor approach and tackle a problem resourcefully, which has been very instructive and also fun. Gábor's constant encouragement and patience played a key role throughout my life at SFU. Gábor, I cannot thank you enough for your support and generosity.

I am grateful to Valentine Kabanets for agreeing to be the examiner of this thesis and for explaining me very useful concepts on several occasions. Valentine's talent for explaining involved results with great clarity, precision and ease, say over coffee, has always amazed me. I thank Funda Ergün for being in my committee, for her support and for being a great co-author. I am grateful to Cenk Şahinalp for introducing me to great problems and topics in my first year in SFU and his insightful guidance throughout my studies. I am indebted to Hossein Jowhari for guiding me through my early days in research with patience and kindness and for countless stimulating discussions. I thank all my friends at SFU who made my years in Vancouver even more memorable. Finally I thank my family for their love and support.

Contents

Chapter 1

Introduction

In the classical model of computation, algorithms with linear space and time are considered optimal. This is well justified, as for most problems, any algorithm must inspect the whole input and the input itself takes linear space. However, starting from late 90's, the explosion of data available to computers made even the optimal algorithms simply impractical for some applications. For instance, consider a search engine that indexes a large portion of the Web. In the classical model, the entire memory can be accessed in unit time. However by todays technology it is simply not possible to have random access memory large enough to fit a web index.

More importantly, recent advances in the Internet infrastructure changed how we think of data in non-trivial ways. More data is being moved to the Cloud by the day, whether it belongs to an end user or a large Internet business, which in turn decreases the costs of cloud storage further. However, data stored remotely can no longer be accessed randomly and must be transmitted over a network in a sequential order first. Furthermore, the devices that form the Internet infrastructure often need to calculate statistics about the data traffic so as to route the data traffic more efficiently. These devices typically have very limited space.

By late 90's it was clear that such advances in technology call for new formal models of computation. In the streaming model, we assume that a sequence of data items are given one by one to the algorithm. The algorithm should compute a function of the input stream by maintaining a small memory. Here, in contrast with the classical model, the input is not counted towards the space usage of the algorithm.

There are very early examples of streaming algorithms. For instance Morris [71] in 1978 shows that one can count the number of data items seen so far approximately using $O(\log\log n)$ bits, where n is an upper bound on the number of data items. Note that the naive solution uses $O(\log n)$ bits, hence is exponentially worse. Also in 1978, Munro and Paterson gave an $O(\sqrt{n}\log n)$ space algorithm that computes the median of a stream of integers by doing two passes over the stream. Then in 1985, Flajolet and Martin [34] gave a one-pass algorithm that computes a constant approximation to the number of distinct items in a given stream using logarithmic space.

After these results, the streaming model attracted only little attention, until the celebrated paper of Alon, Matias and Szegedy [1], where the authors studied the problem of approximating frequency moments and gave strong lower bounds via communication complexity. As also touched upon above, by late 90's the need for a formal model of computation for small space algorithms was evident and a plethora of papers on the streaming model followed [1].

The results presented in chapters 4 and 5 of this thesis appeared in a PODS 2011 paper *Tight Bounds for L_p Samplers, Finding Duplicates in Streams, and Related Problems* [52], joint with Hossein Jowhari and Gábor Tardos. The results presented in Chapter 6 appeared in a RANDOM 2010 paper *Periodicity in Streams* [29], joint with Funda Ergün and Hossein Jowhari. The results of Chapter 3 are either from one of [52] and [29], or from results in preparation, joint with Gábor Tardos and Hossein Jowhari.

Chapter 2

Preliminaries

Throughout this thesis we denote by $[n]$ the set of integers $\{1, 2, \ldots, n\}$ and we denote by $[a..b]$ the set $\{a, a+1, \ldots, b\}$ where $a \leq b$ are two integers. As usual, e denotes Euler's number $2.718\ldots$. We consistently denote by $\log x$ the binary logarithm of x and by $\ln x$ the natural logarithm of x. Let x and y be two n-dimensional vectors. We define the coordinate-wise product operator $\star$ as follows.

$$x \star y = (x_1 y_1, \ldots, x_n y_n)$$

A *string* is a finite length sequence $x_1, \ldots, x_n$, where $x_i \in \Sigma$ for some set Σ, called the *alphabet*. We denote by Σ^* the set of all strings over Σ. The length of a string $x = x_1, \ldots, x_n$ is denoted by $|x|$. Let x and y be two strings. We denote the concatenation of x and y by $x \circ y$ or xy. 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 The Streaming Model

Let $f : \Sigma^* \to T$ be a function where T is an arbitrary set and Σ is an alphabet. In a *streaming problem* we are allowed to do a few (typically one) left to right passes over a sequence $s \in \Sigma^*$, called *the stream*, while maintaining a small memory, and after the last pass, we are asked to output $f(s)$. Let $m, p \geq 1$ be integers and $\delta \in [0, 1]$ be a real. A p-pass δ-error *streaming algorithm* A with space $B = B(m, \delta)$ consists of two stages defined as follows. In the first stage the algorithm is given m, the length of the stream, the desired error probability δ, and possibly other parameters. Through a randomized computation, the algorithm prepares a B bit memory state for the second stage. Here B is a function of m, δ and possibly other parameters given as input to the first stage. In the second stage, the algorithm has only B bits of working memory—initialized in the first stage—and has no access to randomness. The algorithm reads the stream at most p times left to right (but has no write access to it) and after the last pass outputs a value, denoted $A(s)$, with the property

$$\Pr[A(s) \neq f(s)] \leq \delta$$

where the probability is over the randomness used in the first stage. Sometimes we consider streaming problems of the form $f : \Sigma^* \to T^*$ where the algorithms additionally have access to a write-only stream on which the variable length output is written. In a *deterministic streaming algorithm* no randomness is used in either stage.

In an *update stream* the alphabet consists of tuples (i, u), called *updates*, 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. Typically, when a streaming problem takes an update stream as the input, the answer to the problem is only a function of x but not the exact configuration of the stream.

In the *strict turnstile model* we are guaranteed that all coordinates of x are non-negative at the end of the stream, although negative updates are still allowed. In the general model such guarantee does not exist. In the more restrictive cash register model only positive updates are allowed. Namely, for any update (i, u) we have $u \geq 0$. A notable special case is when $u = 1$ for all updates, where the input is no more called an update stream but simply referred to as an *item stream*.

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 $S = \mathrm{supp}(\mu)$. A *random variable* is, roughly speaking, a variable which equals x with probability $\mu(x)$ for each $x \in \mathrm{supp}(\mu)$, where μ is a probability distribution. We say that X is distributed according to μ and write $\mathrm{dist}(X) = \mu$. Further, we let $\mathrm{supp}(X) := \mathrm{supp}(\mathrm{dist}(X))$, i.e., the support of a random variable is defined to be the support of its distribution.

Let $\sigma : S \times T \to \mathbb{R}$ be a distribution. Define distributions μ and ν with support sets respectively S and T as follows.

$$\mu(s) = \sum_{t \in T} \sigma(s, t) \qquad\qquad \nu(t) = \sum_{s \in S} \sigma(s, t)$$

We say that μ is the *marginal distribution* of σ for S and ν is the marginal distribution of σ for T. A function $\pi : S \times T \to \mathbb{R}$ is said to be the *product of μ and ν*, and is denoted $\pi = \mu\nu$, if $\pi(s, t) = \mu(s)\nu(t)$ for all $s \in S$ and $t \in T$.

We say that random variables X and Y are *independent* if $\mathrm{dist}(X, Y) = \mathrm{dist}(X)\,\mathrm{dist}(Y)$. A collection of random variables $X_1, \ldots, X_n$ is called *k-wise independent* if for any $I = \{i_1, \ldots, i_t\} \subseteq [n]$ having at most k elements, we have

$$\mathrm{dist}(X_{i_1}, \ldots, X_{i_t}) = \prod_{j=1}^{t} \mathrm{dist}(X_{i_j}).$$

Namely, in a k-wise independent collection, every selection of at most k random variables are distributed according to the product of their marginal distributions.

In the above definitions, probability distributions and random variables are assumed to have countable support. These definitions can be extended to continuous support sets as well, however this requires more care and the reader is referred to [41] for a rigorous treatment. 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 $\mu = \mathrm{dist}(X)$. It is easy to see that for a real a and two real-valued random variables X and Y we have $\mathbb{E}[aX+Y] = a\,\mathbb{E}[X]+\mathbb{E}[Y]$. This fact, although simple, is very powerful and often referred to as the *linearity of expectation.*

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 [50])**.** *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 $\mathrm{supp}(X) = \mathrm{supp}(U)$ and $\mathrm{supp}(Y) = \mathrm{supp}(V)$ and E be an event determined by X. Set $\mu = \mathrm{dist}(X)$ and $\nu = \mathrm{dist}(U)$. The following hold.*

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

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

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

Proof. Let $S = \mathrm{supp}(X)$. Statement (i) follows by noting

$$\mathbf{D}(\mu \,\|\, \nu) = -\sum_{s \in S} \mu(s) \log \frac{\nu(s)}{\mu(s)} \geq -\log \sum_{s \in S} \nu(s) = 0$$

where the inequality follows from Lemma 2.1. To prove statement (ii) we write

$$\mathbf{D}(\mathrm{dist}(X \mid E) \,\|\, \mu) = \sum_{s \in E} \frac{\mu(s)}{\Pr[E]} \log \frac{\mu(s)}{\Pr[E] \cdot \mu(s)} = -\log \Pr[E].$$

Statement (iii) is obtained by straightforward calculation, however this takes space and we refer the reader to Theorem 2.5.3 in [24] for a proof. □

The following lemma will be used in this thesis, which is immediate from Lemma 2.2 (iii) by an induction.

Lemma 2.3. *Let X be a random variable that is supported on binary strings of length n and X_i be the ith bit of X. We have $\mathbf{D}(\mathrm{dist}(X) \,\|\, q^n) \geq \sum_{i=1}^{n} \mathbf{D}(\mathrm{dist}(X_i) \,\|\, q)$ where q^n is the product of n copies of a Bernoulli distribution q.*

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

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

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:

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

It satisfies the following useful properties, whose proofs can be found in Section 2.1 in [24].

Lemma 2.4. *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) $\mathrm{H}(X) \leq \log |\mathrm{supp}(X)|$, *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 2.5 (Fano [31])**.** *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 2.4 we have

$$\begin{aligned}
\mathrm{H}(X \mid Y) &\leq \mathrm{H}(X, F \mid Y) \\
&= \mathrm{H}(X \mid Y, F) + \mathrm{H}(F \mid Y) \\
&\leq \mathrm{H}(X \mid f(Y), F) + \mathrm{H}(F \mid Y) \qquad \text{(by Lemma 2.4 (v))} \\
&= \delta\, \mathrm{H}(X \mid f(Y), F = 1) + (1 - \delta)\, \mathrm{H}(X \mid f(Y), F = 0) + \mathrm{H}(F \mid Y)
\end{aligned}$$

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 2.4 (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 [16] and Hoeffding [43] 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 [16])**.** *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 $\Pr[X \geq \gamma n] \leq 2^{-n\, \mathbf{D}_2(\gamma \,\|\, \epsilon)}$.*

Proof. Let E be the event that $X \geq \gamma n$. Let μ be the Bernoulli distribution that equals 1 with probability ϵ.

$$\begin{aligned}
-\log \Pr[E] &= \mathbf{D}(\operatorname{dist}(X \mid E) \,\|\, \mu^n) \quad \text{(by Lemma 2.2 (ii))} \\
&\geq \sum_{i=1}^{n} \mathbf{D}(\operatorname{dist}(X_i \mid E) \,\|\, \mu) \quad \text{(by Lemma 2.3)} \\
&\geq n\, \mathbf{D}_2(\gamma \,\|\, \epsilon) \quad (\text{as } \mathbf{D}_2(\delta \,\|\, \epsilon) \geq \mathbf{D}_2(\gamma \,\|\, \epsilon) \text{ for } \epsilon \leq \gamma \leq \delta)
\end{aligned}$$

Hence, $\Pr[E] \leq 2^{-n\, \mathbf{D}_2(\gamma \,\|\, \epsilon)}$ as required. □

Furthermore, the above theorem is essentially tight as shown next. 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 [80]). *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 and communication complexity upper bounds. 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 [77], 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 [77]).

Theorem 2.3 (Schmidt et al. [77]). *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 [77].

Theorem 2.4 (Schmidt et al. [77]). *Let $X = X_1 + \cdots + X_n$, where $X_1, \ldots, X_n$ are k-wise independent real valued random variables such that $\operatorname{supp}(X_i) = [0,1]$ and $\mathbb{E}[X_i] = \epsilon$ for $i \in [n]$. Then for any $2\epsilon + ek/n \leq \gamma \leq 1$ we have $\Pr[X \geq \gamma n] \leq e^{-\lfloor k/2 \rfloor}$.*

2.5 Error Correcting Codes

Suppose that we want to colour the set of length-n strings over the alphabet $[m]$ such that if $0 < \mathrm{Ham}(x,y) \leq k$ then x and y have different colours. Clearly, such colouring can be obtained using only $\sum_{i=0}^{k} \binom{n}{i}(m-1)^i$ colours. To see this, construct a graph where each $x \in [m]^n$ is a node and there is an edge between x and y if $\mathrm{Ham}(x,y) \leq k$. This graph has maximum degree $l = \sum_{i=1}^{k} \binom{n}{i}(m-1)^i$, thus can be coloured using $l+1$ colours. By Theorem 2.1, $\log \sum_{i=0}^{k} \binom{n}{k} \leq n - n\,\mathbf{D}_2(k/n\,\|\,1/2) = n\,\mathrm{H}_2(k/n)$, hence we have the following lemma. This fact is commonly attributed to Gilbert [39] who showed it for the $m = 2$ case and with focus on a single colour class.

Lemma 2.6 (Gilbert [39]). *There exists a mapping $C : [m]^n \to \{0,1\}^s$, where $s = k\log(m-1) + \mathrm{H}_2(k/n)n + O(1)$, such that for $x, y \in [m]^n$ satisfying $0 < \mathrm{Ham}(x,y) \leq d$, we have $C(x) \neq C(y)$.*

In this section we will see the existence of such mappings that can further be computed efficiently using small space as x is being defined by an update stream. Let n, k be integers such that $n \geq k$ and $\mathbb{F}$ be a field with at least n elements. An n by k *Vandermonde matrix* over $\mathbb{F}$ is given by

$$V = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{k-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{k-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \dots & \alpha_n^{k-1} \end{bmatrix}$$

where $\alpha_1, \dots, \alpha_n$ are distinct elements of $\mathbb{F}$. It is well known that any square Vandermonde matrix has full rank. Observing that any subset of rows of V is also a Vandermonde matrix, we conclude that any k rows of V are linearly independent.

Theorem 2.5 (Joffe [51])**.** *Let $\mathbb{F}_q$ be a finite field and $k \leq n$ be integers. Set $S = O(k \log n + k \log q)$. There is an algorithm that given $i \in [n]$ and a seed r, outputs $X_i \in \mathbb{F}_q$ using S space, such that when r is chosen uniformly at random, the random variables $X_1, \dots, X_n$ are k-wise independent where each X_i is uniform on $\mathbb{F}_q$. Further, r can be represented using S bits.*

Proof. Let l be the smallest integer such that $q^l \geq n$. Let g be a generator of $\mathbb{F}_{q^l}$, that is, the finite field of size q^l. Let V be an n by k Vandermonde matrix over $\mathbb{F}_{q^l}$, where we set $\alpha_i = g^i$. Denote the ith row of V by V_i. The seed r is an element of $\mathbb{F}_{q^l}$. On input $i \in [n]$ and r the algorithm computes $a = V_i r$ and considering a as an element of $\mathbb{F}_q^l$ sets $X_i = a_1$, where we denote by a_1 the first coordinate of a.

Clearly, r can be represented using $S = O(k \log n + k \log q)$ bits and a can be computed using the space S efficiently. Now we show that $X_1, \dots, X_n$ are k-wise independent and each X_i is uniformly distributed. Let $I \subseteq [n]$ have exactly k elements. Let V_I be submatrix of V constructed by taking rows V_i for $i \in I$. We argue that $f(r) = V_I r$ is a bijection from $\mathbb{F}_{q^l}$ to itself. To see this, suppose $V_i r_1 = V_I r_2$. This implies $r_1 = r_2$ as V_I has full rank, as required. Thus when r is uniform on $\mathbb{F}_{q^l}$, so is $V_I r$. This completes the proof. □

The next theorem can be regarded as the computationally efficient version of Lemma 2.6 tailored for the streaming model. The following theorem and the corollary following it are folklore, although we are unable to locate a reference.

Theorem 2.6. *Let $1 \leq s \leq n$. For any field $\mathbb{F}$ having infinitely many elements, there is a linear map $L : \mathbb{F}^n \to \mathbb{F}^{2s}$ and a recovery procedure that on input $L(x)$, outputs $x' \in \mathbb{F}^n$ with the following guarantee. For any s-sparse vector x, it holds that $x' = x$.*

For any finite field $\mathbb{F}_q$, there there is random linear map $R : \mathbb{F}_q^n \to \mathbb{F}_{q^l}^{2s} \times \mathbb{F}_{q^{2sl}}$, where l is the smallest integer such that $q^l \geq n$, and a recovery procedure that on input $R(x)$ outputs DENSE or a $x' \in \mathbb{F}_q^n$. If x is s-sparse, $x' = x$ with probability 1. Otherwise, the procedure outputs DENSE with at least $1 - n^{-s}$ probability.

Corollary 2.7. *There is a streaming algorithm which, given an update stream over $\mathbb{F}_q$, takes $O(s \log n + s \log q)$ bits of space and outputs x' or DENSE such that for any s-sparse vector x the output is $x' = x$ with probability 1 and for any vector x that is not s-sparse the output is DENSE with probability at least $1 - n^{-s}$.*

Chapter 3

Communication Complexity

Communication complexity, introduced in [84] by Yao, studies the communication requirements of computing a function whose input is distributed among several parties. In the most basic and standard setting, two players, often referred to as Alice and Bob, receive inputs x and y, respectively, and are required to evaluate $f(x, y)$, where the function f is known in advance to both players. To this end, the players take turns sending each other bit strings according to a predetermined protocol and finally the last player to receive a message should find out $f(x, y)$. Note that the players agree on a protocol after seeing what f is and before receiving their input x, y. The basic question of communication complexity is the minimal number of bits required to compute f, often when there is a restriction on the number of rounds the players can take.

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. Lower bounds in communication complexity have applications in a wide range of areas including data stream algorithms, circuit complexity, data structures (via the cell probe model), Turing machines, and VLSI design. The excellent book of Kushilevitz and Nisan [62] covers the fundamentals of communication complexity as well as its applications; although the area has taken a long way since 1997, where [62] was published.

3.1 Communication Games

Let X, Y, Z be arbitrary sets and let $f : X \times Y \to Z$ be a function. In a *two player communication game*, the first player, Alice, gets $x \in X$ and the other player, Bob, gets $y \in Y$. The aim of the players is to compute $f(x, y)$ by communicating bits. In addition to their input, the players have access to an infinite length shared random string R of independent unbiased coin tosses.

The communication between Alice and Bob takes place according to a predetermined protocol. A *protocol* consists of several rounds, where in a *round* a player sends a bit string, called a *message*, to the other player. We assume that Alice always sends the first message and no player sends messages in consecutive rounds (i.e., the players alternate). The protocol terminates when a player announces the answer. For a protocol P, we denote by $P(x, y)$ the value announced by the last player, when Alice's input was x and Bob's input was y. Note that $P(x, y)$ is a random variable, in case the protocol P depends on the shared random string.

Let μ be a distribution supported on $X \times Y$. We say that P is a (μ, δ)-*distributional error protocol* for f if

$$\Pr_{R,(x,y)\sim\mu}[P(x, y) \neq f(x, y)] \leq \delta$$

where the probability is over both the input distribution μ and the shared random string R. A *deterministic protocol* does not depend on R; i.e., no position of R is queried throughout the execution of P. In a δ-*error randomized protocol*, for any $x \in X$ and $y \in Y$ we have

$$\Pr_R[P(x, y) \neq f(x, y)] \leq \delta\,.$$

In an r-*round protocol*, the number of messages sent is at most r for any x, y and R. The *total communication complexity* of a protocol is the maximum, over all x, y and R, the number of bits sent in total by two parties. The *maximum communication complexity* is the size of the longest message in a protocol for the worst x, y and R.

Let f be a communication problem. We let $D(f)$ be the minimum total communication complexity, over all deterministic protocols for f that have zero distributional error on the uniform distribution. We denote by $R_\delta(f)$ the minimum, over all δ-error randomized protocols P computing f, the total communication complexity of P. The quantities $D^r(f)$ and $R^r_\delta(f)$ are defined similarly, except we take the minimum over protocols with at most r rounds. When δ is not specified, we take it to be 1/3, i.e., $R(f) := R_{1/3}(f)$ and $R^r(f) := R^r_{1/3}(f)$.

Let P be a δ-error randomized protocol and let E be the event that the protocol makes an error, i.e., $P(x, y) \neq f(x, y)$. For any input distribution μ supported on $X \times Y$, we have

$$\begin{aligned}\mathop{\mathbb{E}}_{\mu}\left[\Pr_R[E]\right] &\leq \delta\\ \Pr_{\mu,R}[E] &\leq \delta\\ \mathop{\mathbb{E}}_{R}\left[\Pr_\mu[E]\right] &\leq \delta.\end{aligned}$$

Hence, for the outer expectation in the last line, there exists a fixing of R, which makes $\Pr_\mu[E]$ no bigger than δ. Hard-coding this fixing to P, we obtain a deterministic protocol with (μ, δ)-distributional error. This fact, known as the easy direction of Yao's lemma, is very useful in that it allows us to convert any randomized protocol to a deterministic one without increasing its error (although changing the type of error guarantee); and in that working with deterministic protocols is often much easier. The harder direction of Yao's lemma, roughly speaking, states that fixing the coins of a protocol as described above will not make our lower bounds any weaker, given that μ is chosen carefully. Namely, the lemma states that for any communication problem f, there exists a distribution μ

such that any (μ, δ)-distributional error protocol for f needs to communicate at least $R_\delta(f)$ bits. This direction of Yao's lemma will not be used in this thesis hence we refrain from stating it rigorously. Interested reader is referred to Section 3.4 in [62] (in specific, Theorem 3.20) for a detailed account on the subject.

3.2 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 [68] by Miltersen et al.[1] who showed that any one-round randomized protocol for the problem must communicate $\Omega(n)$ bits. Later on in [7], Bar-Yossef et al. gave the optimal $(1 - \mathrm{H}_2(\delta))n$ bits lower bound for one-round protocols. This problem (with binary alphabets) is also studied by [66, 13, 46] and [14] with the goal of lower bounding the information cost of any randomized protocol. Here we hand-wave the definition of information cost, however we note that it is tightly related to the mutual information between players' inputs and the messages sent in the protocol and that it lower bounds $R(\mathrm{AIND}^n_m)$. We refer the interested reader to [47] for a tutorial on the subject.

To the best of our knowledge, the problem with arbitrary alphabets was first studied in [29] by Ergün, Jowhari and Sağlam who gave an $\Omega(n \log m)$ lower bound for any one-round protocol. Later on, in [49] by Jayram and Woodruff, a closely related problem is studied. In their version of augmented indexing, Bob is additionally given an $a \in [m]$ and is required to determine whether $x_i = a$ or not. They show a tight $\Omega(\min\{n \log(1/\delta), n \log m\})$ bits lower bound for any one-round δ-error randomized protocol.

In this section we present the lower bound of Ergün et al. [29] and an almost matching upper bound for the augmented indexing problem, which together show that

$$\begin{aligned} R^1_\delta(\mathrm{AIND}^{\mathrm{n}}_{\mathrm{m}}) &= n \log m - \delta n \log(m-1) \\ &\quad - \mathrm{H}_2(\delta)n + O(\log n + \log\log(\log m/\delta)). \end{aligned} \tag{3.1}$$

Observe that our upper and lower bounds match up to an additive logarithmic factor. However, in the following chapters the above precise formula will seldom be needed hence we state the following more compact lower bound. The proof of Theorem 3.1 assumes the correctness of Eq. (3.1), which we establish by the two lemmas following Theorem 3.1.

Theorem 3.1. *For $m \geq 3$ and $0 \leq \delta < 1 - e/m^{1-\epsilon}$ for some $\epsilon > 0$, we have $R^1_\delta(\mathrm{AIND}^{\mathrm{n}}_{\mathrm{m}}) = \Omega((1-\delta)n \log m)$. For $m = 2$, we have $R^1_\delta(\mathrm{AIND}^{\mathrm{n}}_{\mathrm{m}}) \geq (1 - \mathrm{H}_2(\delta))n$.*

[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.

Proof. By Eq. (3.1), we have $\mathrm{R}_\delta(\mathrm{AIND}^n_m) \geq (1-\delta)n\log m - \mathrm{H}_2(\delta)n$. This is $\Omega((1-\delta)n\log m)$ as long as $(1-\delta)\log m(1-\epsilon) > \mathrm{H}_2(\delta)$ for some $\epsilon > 0$. We bound

$$\begin{aligned}\mathrm{H}_2(\delta) &= (1-\delta)\log\frac{1}{1-\delta} + \delta\log\frac{1}{\delta}\\ &\leq (1-\delta)\log\frac{1}{1-\delta} + (1-\delta)/\ln 2 \quad (\text{by } x \geq \ln(1+x))\end{aligned}$$

Hence we require $(1-\delta)\log m^{1-\epsilon} > (1-\delta)(\log\frac{1}{1-\delta} + 1/\ln 2)$. This happens when $\delta < 1 - e/m^{1-\epsilon}$ for some $\epsilon > 0$, as required. The second claim is obtained by substituting $m = 2$ in Eq. (3.1). □

Most of our lower bounds for streaming problems in Chapters 4, 5 and 6 are obtained by reductions from Theorem 3.1. In each of these bounds one can, on the surface, replace Theorem 3.1 with the well known results on binary augmented indexing. However, doing the reduction from Theorem 3.1 instead makes our lower bounds hold even for protocols with sub-constant success probability. This does not seem possible by reductions from the binary augmented indexing problem as this problem admits a 0-communication protocol with 1/2 success probability. Equation (3.1) follows directly from the following two lemmas. Let us start with the upper bound.

Lemma 3.1. *Let $m \geq 2$. There is a 1-way protocol for $\mathrm{AIND}^{\mathrm{n}}_{\mathrm{m}}$ that communicates*

$$n\log m - \delta n\log(m-1) - \mathrm{H}_2(\delta)n + O(\log n + \log\log(\log m/\delta))$$

bits and outputs the correct answer with probability at least $1-\delta$.

Proof. Let $\delta' = \delta(1 - 1/(n\log m))$ and let r be a string chosen uniformly at random from $[m]^n$. We have

$$\Pr_r[\mathrm{Ham}(x,r) \leq \delta' n] = \frac{\sum_{i=0}^{\delta' n}\binom{n}{i}(m-1)^i}{m^n} \geq \frac{1}{n}2^{\mathrm{H}_2(\delta')n}\frac{(m-1)^{\delta' n}}{m^n}$$

where the last inequality follows from standard lower bounds for the binomial coefficient (cf. Theorem 2.2). Let $p = n^{-1}2^{\mathrm{H}_2(\delta')n}(m-1)^{\delta' n}/m^n$. The players pick $\ln(n\log m/\delta)/p$ length-n strings $r_1, r_2, \ldots$, each uniformly at random from $[m]^n$ and independently. If there exists a j such that $\mathrm{Ham}(x, r_j) \leq \delta' n$ then Alice sends j to Bob. Otherwise the protocol fails. Upon receiving j, Bob outputs $r_j[i]$. By symmetry, the value Bob outputs is equal to x_i with probability at least $1-\delta'$. Further, the probability that the protocol fails can be bounded by

$$(1-p)^{\ln(n\log m/\delta)/p} < \frac{\delta}{n\log m}.$$

Hence with probability at least $1-\delta$ Bob outputs a value and it is correct. The number of bits Alice sends is

$$\begin{aligned}\lceil\log(\ln(n\log m/\delta)/p)\rceil &= n\log m - \delta n\log(m-1)\\ &\quad - \mathrm{H}_2(\delta)n + O(\log n + \log\log(\log m/\delta)).\end{aligned}$$

as desired. □

Now we proceed with the lower bound.

Lemma 3.2. *For $m \geq 2$ and $0 \leq \delta < 1/2$, we have $R^1_\delta(\mathrm{AIND^n_m}) \geq n \log m - \delta n \log(m - 1) - \mathrm{H}_2(\delta) n$.*

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 2.5)

$$\begin{aligned}
\mathrm{H}_2(\delta) + \delta \log(m-1) &\geq \mathrm{H}(X_I \mid M, R, X_1 X_2 \dots X_{I-1}, I) && (3.2)\\
&= \frac{1}{n} \sum_{i=1}^{n} \mathrm{H}(X_I \mid M, R, X_1 X_2 \dots X_{I-1}, I = i) && (3.3)\\
&= \frac{1}{n} \sum_{i=1}^{n} \mathrm{H}(X_i \mid M, R, X_1 X_2 \dots X_{i-1}) && (3.4)\\
&= \frac{1}{n} \mathrm{H}(X \mid M, R) && (3.5)\\
&\geq \frac{1}{n} \left(\mathrm{H}(X \mid R) - \mathrm{H}(M)\right). && (3.6)
\end{aligned}$$

In steps (3.5) and (3.6) we have used the chain rule for entropy (cf. Lemma 2.4 (iii)). Since X is an element of $[m]^n$ chosen uniformly at random and independently from R, we have $\mathrm{H}(X \mid R) = n \log m$. Arranging, we obtain

$$R^1_\delta(\mathrm{AIND}^n) \geq \mathrm{H}(M) \geq n \log m - \delta n \log(m-1) - \mathrm{H}_2(\delta) n \qquad (3.7)$$

as desired. □

3.3 Sparse Indexing Problem

Historically, in randomized communication complexity, one takes a problem and shows that any constant-error randomized protocol for this problem must communicate at least as much as the deterministic protocols, up to constant factors. In other words, one shows that the constant error and zero error complexities of the problem are asymptotically the same. This is the case with classical problems, including inner product [17, 5], disjointness [53, 76], as well as the augmented indexing problem of Section 3.2. In contrast, there are several problems for which the best protocols we know have an $O(\log(1/\delta))$ term in their communication requirements. Namely, these protocols communicate more as the success probability approaches 1. However there are only a handful of communication lower bounds which get sharper as the error decreases and obtaining such bounds recently attracted considerable attention due to its applications in streaming algorithms.

In this section we investigate a variant of the indexing problem where the optimal communication complexity depends on the desired error guarantee. Consider the following communication problem which is parameterized by integers k and n. Alice gets a

k-subset $S \subset [n]$ and Bob gets an integer $i \in [n]$, where we require that $2k \le n$. The players take turns exchanging messages, where the first message is sent by Alice, and the last player to receive a message should decide whether $i \in S$ or not. We denote this problem by IND_k^n. In this section, we show that $R_\delta^1(\mathrm{IND}_k^n) = \Theta(\min\{k \log \frac{1}{\delta}, k \log \frac{n}{k}\})$. The following upper bound can be derived from the protocol of Håstad and Wigderson [42] and is stated explicitly for $\delta = 1/3$ in [68].

Lemma 3.3 (Miltersen et al. [68])**.** *We have* $R_\delta^1(\mathrm{IND}_k^n) = O(\min\{k \log \frac{1}{\delta}, k \log \frac{n}{k}\})$.

Proof. Recall that Alice is given a k-subset S with $2k \le n$ and Bob is given an integer $i \in [n]$. If $\delta < k/n$, Alice sends the entire description of S to Bob. This takes $\log \binom{n}{k} \le k \log(n/k) + O(k)$ bits.

If on the other hand $\delta \ge k/n$, then using the shared randomness, players pick $L = \lceil (2/\delta)^k \ln(2/\delta) \rceil$ random sets $T_1, T_2, \ldots$ independently as follows. Each T_l, where $l \in [L]$, is identically distributed and generated by including each $j \in [n]$ in T_l with probability $\delta/2$, leaving j out with probability $1 - \delta/2$, independently.

If there is an l such that $S \subseteq T_l$, then Alice sends the integer l to Bob. Otherwise Alice announces $i \in S$. Upon receiving l, if $i \in T_l$, Bob announces that $i \in S$ otherwise he announces that $i \notin S$.

Clearly, $\log((2/\delta)^k \ln(2/\delta)) = O(k \log(1/\delta))$ bits are sent. Further, if $i \in S$, the protocol reports this with probability one. Now suppose $i \notin S$. If Alice cannot find an l such that $S \subseteq T_l$, then Alice erroneously announces that $i \in S$. This has probability

$$(1 + 1/L)^{L \ln(2/\delta)} \le \delta/2.$$

Secondly, the protocol announces $i \in S$ if $i \in T_l$. By construction, this has probability $\delta/2$. Hence by the union bound, the protocol announces $i \notin S$ with at least $1 - \delta$ probability. This completes the proof. $\square$

Now we present our lower bound which shows that the protocol above is tight up to constant factors.

Theorem 3.2. *For any* $\delta \le 1/3$ *we have* $R_\delta^1(\mathrm{IND}_k^n) = \Omega(\min\{k \log \frac{1}{\delta}, k \log \frac{n}{k}\})$.

Proof. For any $1/16 > \delta > k/n$ we prove an $\Omega(k \log \delta^{-1})$ lower bound. Clearly, this implies the claim of the theorem. Our hard distribution is as follows. We give Alice a uniformly random k-subset $S \subset [n]$. With probability half we give Bob an integer from $[n] \setminus S$ uniformly at random, with probability half we give an integer from S uniformly at random. We consider inputs of Alice as length-n binary strings with exactly k ones.

Assume there is a δ-error randomized protocol that communicates s bits. We fix the coins of the protocol so that it makes at most δ error on our hard distribution. By Markov's inequality, there exists a message m such that for at least $2^{-s}/2$ fraction of the input strings Alice sends m and conditioned on m Bob makes at most 2δ error. Let X be a string chosen uniformly at random among the strings for which Alice sends m. Let q be a Bernoulli distribution which equals 1 with probability k/n. Finally, let Y be a random variable distributed according to q^n. We argue that the distribution of the inputs we give to Alice and Y are very close to each other. By standard Chernoff tail estimates (see Theorem 2.2)

$$\Pr[|Y| = k] > \frac{1}{8\sqrt{k}}\left(1 - \Pr\left[\big||Y| - k\big| > 4\sqrt{k}\right]\right) = \Omega(k^{-1/2}).$$

Hence by Lemma 2.2 (ii) and the chain rule for the divergence (cf. Lemma 2.3) we have

$$s + \frac{1}{2}\log k + O(1) \geq \mathbf{D}(\operatorname{dist}(X) \,\|\, q^n) \geq \sum_{i=1}^{n} \mathbf{D}(\operatorname{dist}(X_i) \,\|\, q). \tag{3.8}$$

Define $p_i = \Pr[X_i = 1]$ for $i \in [n]$. Recall that X is chosen randomly among strings for which Alice sends m. Assume that Alice sent the message m and it is Bob's turn to announce the answer. If he outputs $i \in S$ on input i he makes $(1 - p_i)/(2(n-k))$ error, otherwise he outputs $i \notin S$ and makes $p_i/(2k)$ error. Therefore, the optimal strategy for Bob is to output $i \in S$ if $p_i > k/n$, output $i \notin S$ otherwise. Let B be the set of indices for which $p_i > k/n$. Since the total error is at most 2δ when we condition on message m, we have

$$\frac{1}{2k}\sum_{i \notin B} p_i + \frac{1}{2(n-k)}\sum_{i \in B}(1 - p_i) \leq 2\delta \tag{3.9}$$

We relax the above equation and bound either term separately by 2δ. Observe also that each string given to Alice has exactly k ones thus we have

$$\sum_{i=1}^{n} p_i = k. \tag{3.10}$$

Combining the fact that $\sum_{i \in B} p_i \leq k$ and $\delta > k/n$ with a 2δ upper bound on the second term of (3.9) we obtain that $|B| < 5\delta n$. Let $\alpha = \sum_{i \in B} p_i$. Combining a 2δ upper bound on the first term of (3.9) with (3.10) we get that

$$\alpha \geq (1 - 4\delta)k. \tag{3.11}$$

Since the divergence is non-negative, we lower bound the total divergence as follows.

$$\begin{aligned}\mathbf{D}(\operatorname{dist}(X) \,\|\, q^n) &\geq \sum_{i \in B} \mathbf{D}_2(p_i \,\|\, k/n) \\ &= \sum_{i \in B} p_i \log \frac{p_i}{k/n} - \sum_{i \in B}(1 - p_i)\log\frac{1 - k/n}{1 - p_i}\end{aligned}$$

In what follows, we lower bound the first term by $\Omega(k \log \frac{1}{\delta})$ and upper bound the second term by $O(k)$.

$$\begin{aligned}
-\sum_{i\in B} p_i \log \frac{k}{np_i} &= -\alpha \sum_{i\in B} \frac{p_i}{\alpha} \log \frac{k}{np_i} \\
&\geq -\alpha \log \sum_{i\in B} \frac{k}{\alpha n} \quad \text{(by Jensen's inequality, cf. Lemma 2.1)} \\
&= \alpha \log \frac{\alpha n}{|B|k} \\
&\geq (1-4\delta)k \log \frac{(1-4\delta)}{5\delta} \quad \text{(by (3.11) and } |B| < 5\delta n\text{)} \\
&= \Omega(k \log \frac{1}{\delta})
\end{aligned}$$

Lastly, we have

$$\begin{aligned}
\sum_{i\in B} (1-p_i) \log \frac{1-k/n}{1-p_i} &= \sum_{i\in B} (1-p_i) \log(1 + \frac{p_i - k/n}{1-p_i}) \\
&\leq \frac{1}{\ln 2} \sum_{i\in B} (p_i - k/n) \quad \text{(by } \ln(x+1) \leq x\text{)} \\
&= O(k) \quad \text{(by (3.10))}
\end{aligned}$$

Hence we see that $s = \Omega(k \log \frac{1}{\delta})$ by (3.8) as desired. □

3.4 Universal Relation

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 UR^{n}.

This relation has been studied in detail for deterministic communication, as it naturally arises in the context of Karchmer-Wigderson games [58, 57]. 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 [81] 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 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 UR^{n} is due to Tardos and Zwick [81]. Improving a previous result by Karchmer [57], 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.

In this section, we study the randomized complexity of the universal relation problem and give almost matching upper and lower bounds. In particular, Theorem 3.3 gives several efficient randomized protocols for this relation and theorems 3.4 and 3.5 show that these protocols are best possible up to $O(\log(1/\delta))$ terms.

In the next theorem, Statement (iii) is due to Feige et al. [32] and statements (i) and (ii), to the best of our knowledge, first appeared in [52] by Jowhari, Sağlam and Tardos.

Theorem 3.3. *Let $\delta \leq 1/2$ and $n > 2$ be arbitrary. The following statements hold.*

(i) $R^1_\delta(\mathrm{UR^n}) = O(\log^2 n \log \frac{1}{\delta})$

(ii) $R^2_\delta(\mathrm{UR^n}) = O(\log n \log \frac{1}{\delta})$.

(iii) $R_\delta(\mathrm{UR^n}) = O(\log n + \log \frac{1}{\delta})$

Proof. Statement (iii) follows directly from a result of Feige et al. [32] which appears as Theorem 3.10 in this thesis.

Let us prove statement (i). Recall that each of Alice and Bob has a binary string, called x and y respectively. Using the shared randomness, Alice and Bob pick $t = \lceil \log n \rceil + 1$ random binary strings $r_0, \ldots, r_{t-1}$, each of length n, according to the following procedure. The string r_i is obtained by setting each coordinate independently to 1 with probability 2^{-i} and to zero with probability $1 - 2^{-i}$. Let $k = 4e \ln(2/\delta)$. By Lemma 2.6, there is a mapping $C : \{0,1\}^n \to \{0,1\}^s$ where $s = k \log(n/k) + O(k)$ such that for any $u, v \in \{0,1\}^n$ satisfying $0 < \mathrm{Ham}(u,v) \leq k$ we have $C(u) \neq C(v)$.

Recall that $\star$ is the coordinate-wise multiplication operator. Alice sends to Bob the vectors $C(x \star r_i)$ for $i = 0, \ldots, t-1$. Upon receiving these vectors, Bob finds the greatest integer $j < t$ such that $C(x \star r_j) \neq C(y \star r_j)$. Then, Bob finds a vector $\hat{x}$ with minimal Hamming distance to y for which $C(\hat{x} \star r_j) = C(x \star r_j)$ and outputs an arbitrary mismatch between $\hat{x}$ and y.

Clearly, the protocol communicates $O(\log^2 n \log(1/\delta))$ bits. Let us show that the index Bob outputs is a mismatch with probability at least $1 - \delta$. Let $h = \mathrm{Ham}(x,y)$. We argue that there exists a $0 \leq j_0 < t$ such that

$$\frac{k}{4e} < \mathbb{E}[\mathrm{Ham}(x \star r_{j_0}, y \star r_{j_0})] \leq \frac{k}{2e}.$$

This is true, as otherwise there is an i such that $h2^{-i} \leq k/(4e)$ and $k/(2e) < h2^{-i+1}$, which is a contradiction. Let E_1 be the event that there is an $i \in [j_0..t-1]$ such that $\mathrm{Ham}(x \star r_i, y \star r_i) > k$ and let E_2 be the event that $\mathrm{Ham}(x \star r_{j_0}, y \star r_{j_0}) = 0$. Note that if none of these events happen, Bob is able to find a j such that $C(x \star r_j) \neq C(y \star r_j)$; further the vector $\hat{x}$ Bob finds satisfies $\hat{x} \star r_j = x \star r_j$. The probability of E_2 is easy to bound: There are h positions i for which $x_i \neq y_i$ and each such position is set to 1 in r_{j_0} with probability $2^{-j_0} > k/(4e)$. Hence the probability that none of the mismatches are set to 1 in r_{j_0} is $(1-k/(4e))^h < e^{-k/(4e)} \le \delta/2$. By Chernoff bounds (see Theorem 2.1), the probability of E_1 can be bounded by

$$\Pr[E_1] \le \sum_{i=0}^{\infty} 2^{-\mathbf{D}_2(k/h\,\|\,2^{-i}\cdot k/(2eh))\cdot h}$$
$$\le \sum_{i=0}^{\infty} 2^{-k(\log(2^i 2e)-\log e)} \le 2^{-k+1} < \delta/2$$

where the second inequality follows by $e^x \ge x+1$. Hence, either event happens with probability at most $\delta/2$ and by the union bound with probability at least $1-\delta$ Bob outputs a mismatch correctly.

Finally, we show statement (ii). It is well known that there is a 1-way communication protocol that computes an $(1\pm\epsilon)$ approximation to $\mathrm{Ham}(x,y)$ with probability $1-\delta$ using $O(\epsilon^{-2}\log n\log(1/\delta))$ bits [63, 23]. Using this protocol, in the first round the players compute a $(1\pm 1/3)$ approximation h' to $\mathrm{Ham}(x,y)$ with $1-\delta/2$ probability. Set $k = 4e\ln(2/\delta)$ as before. Using the shared randomness, players choose a binary random string r of length n by setting each coordinate to 1 with probability $k/(3eh')$. Bob sends to Alice the vector $C(y\star r)$ and Alice outputs a coordinate exactly as Bob does in the proof of statement (i). Clearly, $O(\log n\log(1/\delta))$ bits are sent in total. The proof of correctness for statement (i) holds here mutatis mutandis. □

We start with an easy theorem which shows an $\Omega(\log n)$ communication bound for any UR^{n} protocol.

Theorem 3.4. *For any $\delta < 1$ we have $R_\delta(\mathrm{UR}^{\mathrm{n}}) = \Omega((1-\delta)\log n)$.*

Proof. Let P be a δ-error protocol for UR^{n} and let μ be the uniform distribution on binary string pairs (x,y) of Hamming distance 1. Let p be the probability, when the inputs are drawn from μ, that the protocol terminates on Alice's side with a correct answer. Let q be the probability, when the inputs are drawn from μ, that the protocol terminates on Bob's side with a correct answer. Here the probabilities are over both the input distribution and the shared random string R. Since the protocol has error probability at most δ, we have $p+q \ge 1-\delta$. In particular, either p or q is greater than $(1-\delta)/2$.

If $q \ge (1-\delta)/2$, we will use P to transmit a $\log n$ bit integer from Alice to Bob, otherwise we will use P to transmit such integer from Bob to Alice. Assume by symmetry $q \ge (1-\delta)/2$. Given an integer $j \in [n]$, Alice sets $x_j = 1$ and $x_i = 0$ for $i \ne j$. Bob sets $y_i = 0$ for $i \in [n]$. Then using the shared randomness players pick a length n permutation π and permute their strings according to it. Further they pick a length n binary string r uniformly at random and XOR the resulting strings with r. They run protocol P on the final strings. Since the final strings constructed by the players are distributed according to μ, with probability q, the protocol terminates on Bob's side with a correct answer. When this happens the output of the protocol is $\pi(i)$, as the strings constructed by the players differ only in this position. Hence, with probability q Bob learns $\pi(i)$ and he can infer i as he knows π.

However, it is a simple fact that to transmit an integer from $[n]$ with q probability, one needs to send $\Omega(q \log n)$ bits. To see this give Alice an integer J chosen uniformly at random from $[n]$ and let Π be the transcript of the protocol, i.e., the collection of all messages sent in each round. By Fano's inequality (Lemma 2.5),

$$q \log n - \mathrm{H}_2(q) \leq \mathrm{I}(J : \Pi \,|\, R) \leq R_\delta(\mathrm{UR^n})$$

hence, $R_\delta(\mathrm{UR^n}) \geq \frac{1}{2}(1-\delta)\log n - 1$ as desired. □

In the following theorem, we show that the 1-way protocol given in Theorem 3.3 (i) is tight up to constant factors and the $\log \delta^{-1}$ term.

Theorem 3.5. *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 $R^1_\delta(\mathrm{UR^n})$-length δ-error one-round protocol for $\mathrm{UR^n}$. 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 $\mathrm{UR^n}$ protocol does not err Bob learns the correct value of z_i. This yields a $R^1_\delta(\mathrm{UR^n})$-length one-way protocol for the augmented indexing problem with error probability $(1+\delta)/2$. By Theorem 3.1 we have $R^1_\delta(\mathrm{UR^n}) = \Omega((1-\delta)st)$. Choosing $s = t$ proves the theorem. □

3.5 Finding a Common Element

Consider the following communication problem which is parameterized by integers s and m. Alice gets an m-subset $A \subset [2m+s]$ and Bob gets an m-subset $B \subset [2m+s]$ with the guarantee that $|A \cap B| = 1$. The players take turns exchanging messages and the last player to receive a message should output the element in $A \cap B$. We denote this problem by FCE^m_s.

Along the lines of Theorem 3.4, it can be shown that any protocol for FCE can be used to transmit an integer from $[m]$ between the two parties, therefore the communication complexity of FCE is at least $\Omega(\log m)$. Furthermore, it was observed in [40] that the classical disjointness problem [53, 76] of size s reduces to FCE which, combined with the previous observation, implies that $R(\mathrm{FCE}^m_s) = \Omega(s + \log m)$. Note that this bound holds for protocols with arbitrary number of rounds. In the following theorem we give a protocol that almost achieves this bound in two rounds only by communicating $O(s \log\log s + \log m)$ bits.

Theorem 3.6. *The following statements hold.*

(i) $D^1(\mathrm{FCE}_s^m) = O(s \log(m/s))$

(ii) $R_\delta^2(\mathrm{FCE}_s^m) = O(s \log\log(s/\delta)\log(1/\delta) + \log m)$.

Proof. Recall that the players are given m-subsets A and B, respectively, with the guarantee $|A \cap B| = 1$. Let x and y be the characteristic vectors of respectively sets A and B.

First we prove statement (i). By Lemma 2.6, there exists a mapping $C : \{0,1\}^n \to \{0,1\}^k$ where $k = O(s \log(m/s))$ such that for any $u, v \in \{0,1\}^n$ satisfying $0 < \mathrm{Ham}(u,v) \le 2s+4$ we have $C(u) \ne C(v)$. Let x' be the string obtained by flipping every bit in x. Alice sends $C(x')$ to Bob and Bob finds the closest string $\hat{x}$ to y such that $C(x') = C(\hat{x})$. Then Bob outputs the index i for which $\hat{x}_i = 0$ and $y_i = 1$. Observe that $\mathrm{Ham}(x', y) = s+2$, therefore $x' = \hat{x}$ and Bob finds the common element correctly.

Let us prove (ii) now. Assume $s \ge 16$ as there is a $O(\log m)$ bits protocol when s is smaller by (i). Using the shared randomness, players pick $t = \lceil 3s \ln(2/\delta) \rceil$ random sets $S_1, \ldots, S_t \subseteq [2m+s]$, chosen according to the following procedure. Each $j \in [2m+s]$ is included in S_i with probability $1/s$ and left out with probability $1 - 1/s$ independently. This is repeated for each $i = 1, \ldots, t$ using fresh randomness. For each i we have

$$\Pr\left[|A \cap B \cap S_i| = 1 \text{ and } |\overline{A} \cap \overline{B} \cap S_i| = 0\right] = \frac{1}{s}(1 - 1/s)^{s+1} > \frac{1}{3s}$$

for $s \ge 16$. Hence with probability all but at most $(1 - \frac{1}{3s})^{3s \ln(2/\delta)} < \delta/2$, there exists an i such that S_i contains the common element and each item in S_i is contained in at least one of A and B. Further, letting $K_i = (A \cap B \cap S_i) \cup (\overline{A} \cap \overline{B} \cap S_i)$, we have

$$\mathbb{E}\left[|K_i|\right] = \frac{s+2}{s} < 2.$$

Let $M = \lceil 12 \log(s/\delta) \rceil$. By Chernoff bounds (see Theorem 2.1) and the fact that each $j \in [2m+s]$ is included in S_i independently, we have $|K_i| < M$ for all $i = 1, \ldots, t$ simultaneously with probability at least $1 - \delta/2$.

Alice sends t integers $c_1, \ldots, c_t$ to Bob, where $c_i = -|\overline{A} \cap S_i| \pmod{M}$. Bob adds in mod M the value $|B \cap S_i|$ to counter c_i for each $i = 1, \ldots, t$. After this addition, if no counter equals 1 (mod M), the protocol fails. Otherwise, let i_0 be an integer such that $c_{i_0} = 1 \pmod{M}$. Now Bob sends i_0 and the integer $\sum_{j \in S_{i_0} \cap B} j$ to Alice, who adds to the latter integer $\sum_{j \in S_i \cap \overline{A}} -j$. It can be verified that if $K_{i_0} = A \cap B$, the result of this addition is the only element of $A \cap B$, i.e., the common element.

In total, Alice sends $t = \lceil 3s \ln(2/\delta) \rceil$ integers each of which takes $O(\log\log(s/\delta))$ bits. Bob replies with a single integer no bigger than m^2 hence the total communication is $O(s \log\log(s/\delta)\log(1/\delta) + \log m)$ bits. The protocol fails if Bob cannot find an i_0 such that $c_{i_0} = 1 \pmod{M}$. This happens with at most $\delta/2$ probability. Further, conditioned on $|K_i| < M$ for all $i = 1, \ldots, t$, $c_i = 1 \pmod{M}$ if and only if $|K_i| = 1$. Hence, by the union bound, Alice outputs the common element with at least $1 - \delta$ probability. □

Now we show a lower bound for the one-way communication complexity of FCE. Let us focus on the $s = 1$ case first. In what follows, A and B correspond to m-subsets of $[2m+1]$. Let μ_0 be an input distribution which gives Alice and Bob only disjoint sets A and B and picks each such disjoint pair with equal probability. Our second distribution μ_1 is the uniform distribution on m-subset pairs A, B having exactly one common element.

We show that in any protocol in which only Alice can toss coins, if the error is polynomially small under μ_1, then Alice must reveal $\Omega(\log m)$ bits of information about her set even under μ_0. Formally, let Π be the message Alice sends and E be the event that the protocol makes an error. Here, Π is a random variable and the randomness is over the input distribution and the private coins of Alice. Bob's answer depends solely on Π and B, as Bob has no access to coins and hence is deterministic. We prove the following.

Lemma 3.4. *If we have* $\mathrm{I}_{\mu_0}(A : \Pi \mid B) < \frac{1}{4}\log m$, *then* $\Pr_{\mu_1}[E] \geq \frac{1}{16m}$.

Proof. Assume for contradiction that $\mathrm{I}_{\mu_0}(A : \Pi \mid B) < (\log m)/4$ and $\Pr_{\mu_1}[E] < 1/(16m)$. Observe that $\mathrm{H}_{\mu_0}(A \mid B) = \log(m+1)$ and $\mathrm{I}_{\mu_0}(A : \Pi \mid B) = \log(m+1) - \mathrm{H}_{\mu_0}(A \mid \Pi, B)$ by the definition of mutual information. By Markov's inequality, there exists a message π such that

$$\log(m+1) - \underset{\mu_0}{\mathrm{H}}(A \mid B, \Pi = \pi) < \frac{1}{2}\log m \quad \text{and} \tag{3.12}$$

$$\underset{\mu_1}{\Pr}[E \mid \Pi = \pi] < \frac{1}{8m} \tag{3.13}$$

hold simultaneously. From now on we fix this message π. Let ν be the distribution of A conditioned on $\Pi = \pi$. Namely, $\nu(a) = \Pr[A = a \mid \Pi = \pi]$, where the probability is over either μ_0 or μ_1. This choice does not make a difference as both distributions have the same marginal for Alice, that is, the uniform distribution on m-subsets of $[2m+1]$. Let ν_0 be the distribution μ_0 conditioned on $\Pi = \pi$ and ν_1 be μ_1 conditioned on $\Pi = \pi$. Note that for two m-subsets $a, b \subset [2m+1]$ and $h \in \{0,1\}$ we have

$$\nu_h(a,b) = \mu_h(a,b)\binom{2m+1}{m}\nu(a). \tag{3.14}$$

Substituting ν_0 in (3.12) and expanding the conditional entropy we get

$$\sum_b \underset{\nu_0}{\Pr}[B = b] \underset{\nu_0}{\mathrm{H}}(A \mid B = b) > \frac{1}{2}\log m. \tag{3.15}$$

Now we argue that (3.15) forces the protocol to make many errors on distribution μ_1. Let a_1, a_2, b be distinct m-subsets of $[2m+1]$ such that $a_1 \cap b = \emptyset$ and $a_2 \cap b = \emptyset$. Let $c = \overline{(a_1 \cup b)} \cup \overline{(a_2 \cup b)} \cup d$, where d is an arbitrary $(m-2)$-subset of b. Observe that $|c| = m$ and c shares exactly one element with both a_1 and a_2. The protocol outputs the same answer on inputs (a_1, c) and (a_2, c), as once Alice's message is fixed, Bob's answer depends solely on his set, namely c. Since $a_1 \cap c \neq a_2 \cap c$, the protocol makes $\min\{\nu_1(a_1,c), \nu_1(a_2,c)\}$ error, regardless of its answer on c. Moreover, we can relate the ν_0 measures of (a_h, b) to ν_1 measures of (a_h, c) for $h \in \{0,1\}$ as follows

$$\nu_1(a_h, c) = \nu_0(a_h, b)\frac{2}{m^2}. \tag{3.16}$$

To see this observe that $\mu_0(a_h, b) = (\binom{2m+1}{m}(m+1))^{-1}$ and $\mu_1(a_h, c) = (\binom{2m+1}{m}\binom{m+1}{2}m)^{-1}$, and recall μ and ν measures are proportional in the sense of (3.14). Now we calculate the total error. For each input b of Bob, we construct a graph G_b such that each input of Alice is a node. Further, between every two node a_1, a_2 that satisfies $|a_1 \cap a_2| = m - 1$, $|a_1 \cap b| = |a_2 \cap b| = 1$ and $a_1 \cap b \neq a_2 \cap b$, we put an edge of weight $\min\{\nu_1(a_1, b), \nu_1(a_2, b)\}$. Observe that each node a in G_b has degree $m - 1$, since each neighbour of a intersects with b at a different element. Note that the weight of any matching in G_b lower bounds the error Bob makes on input b. Further, we claim that given any G_b, we can find a matching of weight $\nu_1(G_b)/(2m - 3)$, where $\nu_1(G_b)$ denotes the total weight of all edges in the graph. Indeed, sorting the edges according to their weight and greedily adding the edges to matching starting from the heaviest edge gives such a guarantee, as we remove $2m - 4$ edges from G_b for every edge we add to the matching. Hence the total error on ν_1 is at least

$$\frac{1}{2m-3}\sum_b \nu_1(G_b) \tag{3.17}$$

$$= \frac{1}{2m-3}\sum_b \sum_{a_1 \neq a_2} \sum_d \min\{\nu_1(a_1, c), \nu_1(a_2, c)\} \tag{3.18}$$

where b is an m subset of $[2m+1]$, a_1, a_2 range over all m-subsets of $[2m+1] \setminus b$, d ranges over $(m-2)$-subsets of b and $c = \overline{(a_1 \cup b)} \cup \overline{(a_2 \cup b)} \cup d$. By (3.16), we can write (3.18) as

$$\frac{2\binom{m}{2}}{m^2(2m-3)}\sum_b \sum_{a_1 \neq a_2} \min\{\nu_0(a_1, b), \nu_0(a_2, b)\} \tag{3.19}$$

where a_1, a_2 are m-subsets of $[2m+1] \setminus b$.

Let x be a vector such that $\sum_{i=1}^n x_i = 1$ and $0 \leq x_1 \leq \ldots \leq x_n \leq 1$. We claim that $1 - x_n \geq \mathrm{H}(x)/\log n - 1/4$ for large n. Recall that $\mathrm{H}(x) \leq (1 - x_n)\log(n-1) + 1$ and hence

$$(1 - x_n) + 1/4 \geq (1 - x_n) + 1/\log n \geq \mathrm{H}(x)/\log n$$

whenever $n > 16$. Also note that $\sum_{i<j} \min\{x_i, x_j\} \geq 1 - x_n$. Therefore,

$$\begin{aligned}
&\sum_b \sum_{a_1 \neq a_2} \min\{\nu_0(a_1, b), \nu_0(a_2, b)\} \\
\geq &\sum_b \Pr_{\nu_0}[B = b]\left(\frac{\mathrm{H}_{\nu_0}(A \mid B = b)}{\log(m+1)} - 1/4\right) \\
= &\frac{1}{\log(m+1)}\sum_b \Pr_{\nu_0}[B = b]\,\mathrm{H}_{\nu_0}(A \mid B = b) - \frac{1}{4}\sum_b \Pr[B = b] \\
\geq &1/2 - 1/4 = 1/4
\end{aligned}$$

where the last inequality follows from (3.15). Hence the protocol makes at least $1/(8m)$ error on μ_1, and this contradicts with (3.13), proving our claim. □

In the next theorem, through a direct-sum argument, we extend our results to all s.

Theorem 3.7. *Let $n > s$ be integers. It holds that $R^1_{s/n}(\mathrm{FCE}^n_s) = \Omega(s \log(n/s))$.*

Proof. Recall that in FCE^n_s the players receive n-subsets A, B of $[2n+s]$. Assume that $n = sm$ for some integer m and consider the universe $[2n+s]$ as partitioned into s blocks of size $2m+1$. Each of Alice and Bob will get exactly m elements from each block. We denote Alice's subset in block i by A_i. Similarly, B_i denotes the elements Bob get in ith block. Then, $A = \bigcup_{i=1}^s A_i$ and $B = \bigcup_{i=1}^s B_i$. By b_{-i} we denote a certain fixing of Bob's input for blocks $j \neq i$. In other words, b_{-i} specifies an m-subset for Bob for each block except i. For each such fixing, we define a distribution $D(b_{-i})$ as follows: In $D(b_{-i})$, Bob's elements in blocks $j \neq i$ is fixed to values given by b_{-i}. In blocks $j \neq i$, Alice gets a random m-subset that does not intersect with Bob's set. In block i we pick sets according to μ_1 (i.e., uniquely intersecting). Our hard distribution D is the average of $D(b_{-i})$ for all i and all fixings b_{-i}, that is, the uniform distribution on pairs of sets intersecting in a single element and containing equal number of elements from each block.

Assume there is a randomized one-way protocol for FCE^n_s which communicates $s\log(n/s)/16$ bits and outputs a correct answer with high probability. We fix the coins of this protocol in a way that it makes at most $1/(32n)$ error on D. Denote the messages of the deterministic protocol we obtain by Π. By Markov's inequality, the protocol makes at most $1/(16n)$ error on $D(b_{-i})$ for at least half of all fixings b_{-i}, $i \in [s]$. Further

$$\mathop{\mathrm{I}}_{\mu_0^s}(A : \Pi \mid B) \leq s\log(n/s)/16 \tag{3.20}$$

and hence, there is an integer $k \in [s]$ and fixing b_{-k} such that

$$\Pr_{D(b_{-k})}[E] < 1/(16m) \quad \text{and} \tag{3.21}$$

$$\mathop{\mathrm{I}}_{\mu_0^s}(A_k : \Pi \mid B_k, B_{-k} = b_{-k}) \leq \log(n/s)/8 \tag{3.22}$$

Now we give a protocol for FCE^m_1 in which only Alice tosses coins. Alice and Bob are given m-subsets S and T. Bob, in block $j \neq k$ places the sets given in b_{-k}. In block k, he places the set T. For each $j \neq k$, Alice picks a random set that does not intersect the set given in b_{-k} (Recall that b_{-k} is fixed and Alice knows it). In block k Alice places the set S. The players run the protocol with these inputs. If S and T come from μ_1, the error is small by (3.21). If S and T come from μ_0, then Alice reveals little by (3.22). This contradicts with Lemma 3.4. $\square$

3.6 The Greater Than Problem

In the *greater-than problem*, two players are given integers, respectively $0 \leq x, y < 2^n$, with the guarantee that $x \neq y$ and are asked to determine whether $x < y$ or the other way around. In [81], a harder variant of the universal relation problem was also studied, where two players are given length-n binary strings x, y and are required to output the leftmost position in which x and y differ. It is easy to see that any protocol for finding the leftmost difference is also a protocol for the greater-than problem. For finding the leftmost difference, Tardos and Zwick [81] give a deterministic protocol that communicates $n + \log^* n$ bits in $\log^* n + 1$ rounds. In fact, their protocol is tunable and for any $1 \le r \le \log^* n$, the protocol runs in r rounds using $n + \log^{(r)} n + r$ bits of total communication. Furthermore, they show that any r-round deterministic protocol that finds the leftmost difference must communicate at least $n + \log^{(r)} n - 2$ bits. Contrasting this lower bound with the 2-round $n - \lfloor \log n \rfloor + 2$ bits protocol [81] for universal relation, we see that finding the leftmost difference is harder than finding an arbitrary difference in deterministic communication complexity. The gap between between the two is much sharper in randomized communication as we shall see in this section.

The communication complexity of the greater-than problem was first studied by Smirnov in [79]. In [32], Feige et al. give a randomized protocol for finding the leftmost difference (hence for the greater-than problem too) which communicates $O(\log n)$ bits in total, taking $O(\log n)$ rounds. This result is given as Theorem 3.10 in this section. In [68], Miltersen et al. show through a round elimination argument that any r-round randomized protocol must have a message of size $\Omega(n^{1/r}/2^{O(r)})$. They also give an r-round protocol which sends $O(n^{1/r} \log n)$ bits per message. Finally, in [78] the round elimination lemma of [68] was strengthened by Sen and Venkatesh to obtain the following result.

Theorem 3.8 (Sen et al. [78]). *In any r-round 1/3-error randomized protocol for the greater than problem, there is a message of size at least $\Omega(n^{1/r}/r^2)$ bits.*

In particular, this bound shows that the communication complexity of finding the leftmost difference is polynomial for any constant r, whereas, by results of Section 3.4, an arbitrary mismatch can be found using $O(\log n)$ bits in two rounds only. Hence there is an exponential gap between finding the leftmost and an arbitrary difference in randomized communication complexity. An $O(n^{1/r} \log n)$ bits per message upper bound was also mentioned in [78]. In the next theorem, we improve this upper bound slightly and give an r-round protocol with $O(n^{1/r})$ bits per round.

Theorem 3.9. *Suppose Alice and Bob are given binary strings, respectively x and y, of length n. There is an r-round protocol with $O(n^{1/r})$ bits messages that finds the leftmost difference of x and y except with probability at most $2^{-r-2} + r2^{-n^{1/(2r)}}$. The error probability can be simplified to 2^{-r-1} for $r < \log n / \log\log n$.*

Proof. If $r = 1$, then Alice sends her entire input to Bob and Bob can solve the problem with probability 1. Hence the theorem holds for $r = 1$. Let us assume $r \ge 2$.

Let $B = 2^6 \lceil n^{1/r} \rceil$. Players consider x and y as partitioned into $B+1$ non-intersecting blocks of length precisely $\lfloor n/B \rfloor$, except for the last block, which can be shorter. We denote by X_i the concatenation of the first i blocks of x. Define Y_i similarly for Bob's string.

In the first round, Alice sends one bit of information about each X_i to Bob as follows. Using the shared randomness, the players pick $B+1$ random strings $r_1, r_2 \ldots, r_{B+1}$ where r_i is of length $|X_i|$. Each string is chosen uniformly at random and independently. Alice, for each $i = 1, \ldots, B+1$, sends the dot product $r_i \cdot X_i$. Here, this dot product and all dot products in this proof are computed in modulo 2. Bob compares these values with $r_i \cdot Y_i$ for $i \in [B+1]$. Clearly, if $X_i = Y_i$ then the the dot products are equal. On the other hand, if $X_i \neq Y_i$ then the dot products will differ with probability half. Let j be the smallest integer such that $r_j \cdot X_j \neq r_j \cdot Y_j \pmod 2$. If there is no such integer take j to be $B+1$.

Now for each $i = 1, \ldots, j$, Bob partitions block i further into $B2^{i-j}$ sub-blocks. We denote the concatenation of the first k sub-blocks of block i by $Y_{i,k}$. Define $X_{i,k}$ similarly. Bob sends to Alice a 1-bit hash about $Y_{i,k}$ for each $i < j$ such that $B2^{i-j} > 1$ and for each $k < 2^{j-i}$. Note that in total less than $2B$ bits will be sent. These hash values are obtained as described above, i.e., by multiplying the sub-blocks in mod 2 with properly sized random strings. On top of these hash values, for block j and for the first $\sqrt{B}$ blocks preceding j, Bob sends a $\sqrt{B}$-bits equality hash. Again, these has values are obtained by multiplying the respective block with properly sized $\sqrt{B}$ different (independently chosen) random strings. Here, the number of bits sent is $\sqrt{B} \cdot \sqrt{B} = B$. Further, Bob sends the integer j to Alice.

Upon receiving Bob's message, Alice first inspects the $\sqrt{B}$ bit long hashes and locates the first block preceding j (or block j itself) for which her hashes and Bob's hashes do not agree. Let this be block i_0. Note that the probability that the first mismatch is to the left of first $\sqrt{B}$ blocks preceding j is $2^{-\sqrt{B}}$. Further, these hashes are $\sqrt{B}$ bits long, hence the probability that there is a mismatch to the left of block i_0 is $O(2^{-n^{1/(2r)}})$. Note that players need to locate the left most mismatch block at the beginning of each round, hence by the union bound, except with probability $O(r2^{-n^{2/r}})$, we may assume such comparisons yield the correct answer.

After Alice locates i_0, she zooms to block i_0 discarding all other blocks. Then she inspects the 1-bits hashes for $X_{i_0,k}$ where $k \leq 2^{j-i_0}$. Let k_0 be the smallest integer such that the hash values for X_{i_0,k_0} and Y_{i_0,k_0} do not agree. Similarly, for $k < k_0$, she partitions the sub-block k further into $B2^{k-k_0}$ pieces and sends a 1-bit hash about each prefix of this block. Similar to the previous round, she sends $\sqrt{B}$ bit hashes for the $\sqrt{B}$ sub-blocks preceding k_0. The protocol continues in the same manner until the end of the $(r-1)$th round. In the last round, the player to speak sends its entire remaining input to the other player.

Recall that in the first round only $B+1$ bits are sent and for rounds 2 to $r-1$ players send $O(B)$ bits per round. Now we analyze the probability the input size exceeds B at the end of the $(r-1)$th round. Imagine that we are tossing coins one after another, while keeping track of how many heads and tails appeared so far. The probability of having seen $6r$ heads at the point we reach our $(r-1)$th tail can be calculated as

$$\binom{7r}{r} 2^{-6r} \leq (7e)^r / 2^{6r} < 2^{-r-2}.$$

Let H be the size of the remaining input at the end of the $(r-1)$th round. Now we relate the probability that $H > B$ with the above experiment. Associate the event that a 1-bit hash correctly detects a mismatch with the event that we see a tails and associate the event that a 1-bit hash fails with the event that we see a heads. Note that each time a 1-bit hash fails, H is multiplied by two and each time a mismatch is detected we proceed to the next round and H is divided by $B = 2^6 \lceil n^{1/r} \rceil$. Hence, if at the time we reach our $(r-1)$th tail (i.e., when we are at the last round) we have seen no more than $6r$ heads, we have $H < n^{1/r} < B$ as desired. By the union bound, this happens with probability at least $1 - 2^{-r-2} + r2^{-n^{1/(2r)}}$. This completes the proof □

The error guarantee of Theorem 3.9 is meaningful only when $r = O(\log n / \log\log\log n)$ and in particular the second error term increases above 1 if we push r further than this threshold. At the extremum point, we obtain a $\log n / \log\log\log n$ rounds protocol with $O(\log n \frac{\log\log n}{\log\log\log n})$ total communication and $1/\log n$ error probability (further, each message is at most $O(\log\log n)$ bits). It is an intriguing question at this point whether the total communication can be brought down to $O(\log n)$ bits. An early result due to Feige et al. shows that, in fact, this is possible by further increasing the number of rounds to above $\log n$. For completeness, we include the proof of this theorem here.

Theorem 3.10 (Feige et al. [32])**.** *There is a protocol that solves the greater than problem with $1 - \delta$ probability in $O(\log n + \log \frac{1}{\delta})$ rounds by communicating $O(\log n + \log \frac{1}{\delta})$ bits in total. Further, the protocol outputs the location of the first mismatch.*

Proof. Without loss of generality, we assume that $n = 2^t$ for some integer t. If this is not the case, the players can pad their inputs with sufficiently many zeros. We construct a labelled tree of depth $4\log(n/\delta) = 4t + 4\log(1/\delta)$ on top of the input as follows. The root of the tree is labelled by the interval $[1, 2^t]$. For $i \in [0..t-1]$, every node at depth i having label $[a, b]$ has two children, such that the left one is labelled by $[a, m]$ and the right one is labelled by $[m + 1, b]$ where $m = (b - a - 1)/2$. Hence, at depth t there are 2^t nodes, each labelled by $[a, a]$ for distinct $a \in [2^t]$. Every node at depth greater than t has exactly one child, labelled the same as the parent.

We imagine that that a chip is placed at the root of the tree and in each round one player moves the chip to an adjacent position. Intuitively, if the chip is located at a node with label $[a, b]$, this means that the players expect the leftmost mismatch to be between a and b. In particular, the players aim at maintaining that $x[a, b] \neq y[a, b]$ and they will backtrack as soon as they detect that this condition is violated.

The protocol proceeds as follows. Let the chip be located at node v with label $[a, b]$. If v has two children, the label of the left child is $[a, m]$ and the label of the right is $[m + 1, b]$, where $m = (b - a - 1)/2$. Suppose it is Alice's turn to speak. She sends 4-bit equality hashes for the substrings $x[a, b]$ and $x[a, m]$. Bob compares these hashes with the hashes for $y[a, b]$ and $y[a, m]$, respectively. If the hashes for the interval $[a, b]$ match, he moves the chip to its parent. Otherwise, if the hashes for interval $[a, m]$ match, he moves the chip to the right child of v. In the final case, the chip is moved to the left child of v. Let v' be the node the chip is moved to.

After this, now it is Bob's turn to speak, who first lets Alice know v' (this takes 2 bits) and similarly sends hashes for v' and the left child of v'. The players repeat this exchanging messages for exactly $4t + 4\log(1/\delta)$ rounds. After the final round, if the chip is on a node with a label of the form $[a, a]$, then a is announced as the answer. Otherwise, the protocol fails.

Clearly, the protocol takes $O(\log(n/\delta))$ rounds and 10 bits are sent in each round. Now we analyze the probability of success. Let i be the first mismatch between x and y and u be the deepest node with label $[i, i]$. Orient every edge in the tree so that starting from any node, following the edges leads us to u. Note that, at each round with probability 7/8, equality hashes give the correct answer and the chip is moved to a node which is closer to u. Initially the chip is at distance $4t + 4\log(1/\delta)$ to u, hence the expected distance of the chip to u after the final round is $t + \log(1/\delta)$. Moreover, the correct answer is announced if the final distance is at most $3t + 4\log(1/\delta)$. By Chernoff bounds, the final distance exceeds $3t + 4\log(1/\delta)$ with probability less than δ. This completes the proof. $\square$

Chapter 4

L_p-Sampling from Update Streams

Sampling has become an indispensable tool in analyzing massive data sets, and particularly in processing data streams. In the past decade, several sampling techniques have been proposed and studied for the data stream model [6, 27, 11, 22, 70, 3]. In this chapter, we study L_p*-samplers*, a new variant of space efficient samplers for data streams that was introduced by Monemizadeh and Woodruff in [70]. 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 $|x_i|^p$.

In [70], it was observed that L_p-samplers lead to alternative algorithms for many known streaming problems, including heavy hitters and frequency moment estimation. Here in this thesis, we focus on a specific application, namely finding duplicates in long streams; although our L_p samplers work and often give better space performance for all applications listed in [70]. We refer the reader to [70] and [3] for further applications of L_p-samplers.

Observe that we allow both negative and positive updates to the coordinates of the underlying vector. In the case where only positive updates are allowed and $p = 1$, the problem is well understood. For instance the classical reservoir sampling [60] from the 60's (attributed to Alan G. Waterman) gives a simple solution as follows. Given a pair (i, u), indicating an addition of u to the i-th coordinate of the underlying vector x, the sampler having maintained s, the sum of the updates seen so far, replaces the current sample with i with probability u/s, otherwise does nothing and moves to the next update. It is easy to verify that this is a perfect L_1-sampler and the space usage is only $O(1)$ words.

With the presence of negative updates, sampling becomes a non-trivial problem. In this case, it is not clear at all how to maintain samples without keeping track of the updates to the individual coordinates. In fact, the question regarding the mere existence of such samplers was raised few years ago by Cormode, Muthukrishnan, and Rozenbaum in [21]. Last year in SODA 2010, Monemizadeh and Woodruff [70] answered this question affirmatively, however in an approximate sense. Before stating their results we give a formal definition of L_p-samplers.

Definition 1. *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 $(1 \pm \epsilon)x_i$.

Previous work A zero relative error L_0-sampler which uses $O(\log^3 n)$ bits was shown in [35]. In [70], the authors gave an ϵ relative error L_p-sampler for $p \in [0, 2]$ which takes $\text{poly}(\epsilon^{-1}, \log n)$ space. They also showed a 2-pass $O(\text{polylog}\, n)$ space zero relative error L_p-sampler for any $p \in [0, 2]$. In addition to these, they demonstrated that L_p-samplers can be used as a black-box to obtain streaming algorithms for other problems such as L_p estimation (for $p > 2$), heavy hitters, and cascaded norms [48]. Unfortunately, due to the large exponents in their bounds, the L_p-samplers given there do not lead to efficient solutions for the aforementioned applications.

Very recently, Andoni, Krauthgamer and Onak in [3] improved the results of [70] considerably. Through the adaptation of a generic and simple method, named *precision sampling*, they managed to bring down the space upper bounds to $O(\frac{1}{\epsilon^p} \log^4 n)$ bits for ϵ relative error L_p-samplers for $p \in [1, 2]$. Roughly speaking, the idea of precision sampling 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. In more detail, the input vector $(x_1, \ldots, x_n)$ is scaled by random coefficients $(t_1^{-1}, \ldots, t_n^{-1})$, where each t_i is picked uniformly at random from $[0, 1]$. Let $z = (x_1 t_1^{-1}, \ldots, x_n t_n^{-1})$ be the scaled vector. Here the important observation is $\Pr[t_i^{-1} \geq t] = 1/t$ and hence, for instance, by replacing t with $\|x\|_1/|x_i|$, we get $\Pr[|z_i| \geq \|x\|_1] = |x_i|/\|x\|_1$. (In the same manner, one can scale x_i by $t_i^{-1/p}$ instead of t_i^{-1} and get a similar result for general p.) It turns out, we only need to we have a constant approximation to $\|x\|_1$ and look for a coordinate in

z that has reached a limit of $\Omega(\|x\|_1)$. On the other hand it is shown that the heaviest coordinate in z has a weight of $\Omega(\log^{-1} n)\|z\|_1$ (with constant probability), and thus a small-space heavy hitter computation can be used to find the maximum. In particular, the L_p-sampler of [3] adapts the popular *count-sketch* scheme [15] for this purpose.

Our contributions In this chapter, we give L_p-samplers requiring only $O(\epsilon^{-p}\log^2 n)$ space for $p \in (1,2)$. For $p \in (0,1)$, our 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. In essence, our sampler follows the basic structure of the precision sampling method explained above. However compared to [3], we do a sharper analysis of the error terms in the count-sketch, and through additional ideas, we manage to get rid of a log factor and preserve the previous dependence on ϵ. Roughly speaking, we use the fact that the error term in the count-sketch is bounded by the L_2 norm of the tail distribution of z (the heavy coordinates do not contribute). On the other hand, taking the distribution of the random coefficients into account, we bound this by $O(\|x\|_p)$, which enables us to save a log factor. Additionally, to preserve the dependence on ϵ, we have to use a slightly more powerful source of randomness for choosing the scaling factors (in contrast with the pairwise-independence of [3]), and take care of some subtle issues regarding the conditioning on the error terms which are not handled in the previous work (Lemma 4.3).

As p approaches zero, precision sampling becomes inefficient, as the random coefficients $t_i^{-1/p}$ tend to infinity. For the $p = 0$ case, we present a zero relative error sampler through a completely different approach. Briefly, our L_0-sampler tries to detect a non-zero coordinate by picking random subsets of $[n]$. The non-zero coordinates are found by an exact sparse recovery procedure and Nisan's PRG [73] is applied to decrease the randomness involved. Our $O(\log^2 n)$ space bound compares favourably to the previous algorithms, which use respectively $O(\log^3 n)$ space [35] and $\mathrm{poly}(\log n, \epsilon^{-1})$ space [70] (the latter one gives only ϵ-relative error sampling).

In Section 4.3, we prove that sampling from $0, \pm 1$ vectors requires $\Omega(\log^2 n)$ space, by a reduction from the universal relation communication problem. In this special case p is not relevant for L_p-sampling, hence this shows that our 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.

Finally, we prove lower bounds for the problem of finding heavy hitters in update streams, which is closely related to the L_p-sampling problem. This lower bound is obtained by a direct reduction from the augment indexing and proves that any L_p heavy hitters algorithm (defined in Section 4.3) must use $\Omega(\frac{1}{\phi^p}\log^2 n)$ space, even in the strict turnstile model. Our lower bound essentially matches the known upper bounds [20, 15, 54] which work in the general update model.

Related work In [6, 11], the authors have studied sampling from sliding windows, and the recent paper of Cormode et al. [22] generalizes the classical reservoir sampling to distributed streams. These works only consider insertion streams. The basic idea of random scaling used in [3] and in our paper has appeared earlier in the priority sampling technique [27, 18], where the focus is to estimate the weight of certain subsets of a vector, defined by a sequence of positive updates.

Heavy hitter algorithms have been studied extensively. The work of Berinde et al. [9] gives tight lower bounds for heavy hitters under insertion only streams. We are not aware of similar works on general update streams, although the recent works of [4, 83], where the authors show lower bounds for respectively approximate sparse recovery, and Johnson-Lindenstrauss transforms (via augmented indexing) is closely related.

Notation Recall that an update stream is a sequence of tuples (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). 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.

In the standard model for randomized streaming algorithms the random bits used (to generate the random linear map L, for example) are part of the space bound (see Section 2.1 for a precise definition of the model). In contrast, our lower bounds do not make any assumption on the working of the streaming algorithm and allow for the *random oracle model*, where the algorithm is allowed free access to a random string at any time. All lower bounds are proved through reductions from communication problems given in Chapter 3.

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 $\text{Err}_2^m(x) = \min \|x - \hat{x}\|_2$, where $\hat{x} \in \mathbb{R}^n$ ranges over all the m-sparse vectors.

4.1 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.

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 $n' < n$.

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

The *count-sketch* algorithm [15] 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 [38].)

Lemma 4.1 (Charikar et al. [15]). *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

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

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 4.2 (Kane et al. [55]). *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 4.1 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 4.3. *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 4.2 we have $r \ge \|x\|_p$ and $s \le 2\|z - \hat{z}\|_2$ with high probability. By Lemma 4.1 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 $\mathrm{Err}_2^m(z) \le T S''^{1/2}$ 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

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

$k = 2\lceil \log(2/\epsilon) \rceil$. 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 4.4. *The probability that the algorithm of Figure 4.1 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 4.3 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 4.2) 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 4.1.

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.

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 4.1. *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 4.4 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 4.1 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 4.4. In one round, the algorithm on Figure 4.1 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 [3].

As we will see in Theorem 4.3, 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.

4.2 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 4.2. *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 [73]. 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 2.6 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 2.6 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 [73] 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 4.3, 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.

4.3 Lower bounds

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.

Theorem 4.3. *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. Recall that 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 5.2, $S = \Omega(\log^2 n)$, 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 4.4. *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 3.1, $S = \Omega(st)$. Setting $s = \log_b n$ and $t = \epsilon^{-p}$ completes the proof, as p is constant. □

Theorem 4.5. *Let $p > 0$ and $\phi \in (0, 1)$ be a reals. Any one pass heavy hitter algorithm in the strict turnstile model uses $\Omega(\phi^{-p} \log^2 n)$.*

Proof. Suppose there is a one pass heavy hitter algorithm for parameters p and ϕ. We allow for a random oracle and assume the updates are polynomially bounded in n and integers. We can also restrict the number of updates to be $O(\phi^{-p} \log n)$ and assume all coordinates of the final vector are positive (strict turnstile model). We turn this streaming algorithm into a protocol for augmented indexing in a similar way as we transformed the protocol for UR^n to a protocol for augmented indexing in the proof of Theorem 3.5. The exponential growth is now achieved not by repetition but by multiplying the coordinates with a growing factor.

Suppose Alice and Bob wants to solve the augmented indexing problem and Alice receives $y \in [2^t]^s$ and Bob gets $i \in [s]$ and y_j for $j < i$. Let them construct real vectors u and v as follows. Let $b = (1 - (2\phi)^p)^{-1/p}$ and let $e_q \in \mathbb{R}^{2^t}$ be the standard unit vector in the direction of coordinate q. Alice obtains u by concatenating the vectors $\lceil b^{s-j} \rceil e_{y_j}$ for $j \in [s]$. The dimension of u is $n' = s2^t$. Bob obtains v by concatenating the same vectors for $j \in [i-1]$ and then concatenating enough zeros, namely $(s - i + 1)2^t$, to reach the same dimension n'. Now Alice and Bob perform the heavy hitter algorithm for the vector

$x = u - v$ as follows. Alice generates the necessary updates to increase the initially zero vector $x \in \mathbb{Z}^n$ to reach $x = u$, maintains the memory content throughout these updates and sends the final content to Bob. Now Bob generates the necessary updates to decrease $x = u$ to its final value $x = u - v$ and maintains the memory throughout. Finally Bob learns the heavy hitter set S the streaming algorithm produces and outputs $z \in [2^t]$ if the smallest index in S is $(i-1)2^t + z$.

We claim that the above protocol errs only if the streaming algorithm makes an error. Notice that all coordinates of x_l of $x = u - v$ are zero except the ones of the form $x_{l_j} = \lceil b^{s-j} \rceil$ for $l_j = (j-1)2^t + y_j$, where $i \leq j \leq s$. Thus x_{l_i} is the first non-zero coordinate. So the claim is true if $x_{l_i} \geq \phi \|x\|_p$. Using $\lceil v \rceil < 2v$ for $v \geq 1$ we get exactly this:

$$\begin{aligned}
\phi^p \|x\|_p^p &= \phi^p \sum_{j=i}^{s} \lceil b^{s-j} \rceil^p \\
&< (2\phi)^p b^{p(s-i+1)}/(b^p - 1) \\
&= b^{p(s-i)} \qquad \text{(since } b^p = 1/(1-(2\phi)^p)\text{)} \\
&\leq x_{l_i}^p
\end{aligned}$$

Let us now choose $s = \lceil (2\phi)^{-p} \log n \rceil$ and $t = \lceil \log n/2 \rceil$. For large enough n this gives $n' = s2^t < n$ and all coordinates of x throughout the procedure remain under n. Still if the streaming algorithm works with probability over 1/2, then by Theorem 3.1 the message size of the devised protocol is $\Omega(st) = \Omega(\phi^{-p} \log^2 n)$. This proves the theorem as the message size of the protocol is the same as the memory size of the streaming algorithm. $\square$

Chapter 5

Finding Duplicates and Odd-frequency Items in Streams

5.1 Finding Duplicates

Suppose we are given an array $a_1, a_2, \ldots, a_{n+1}$ of length $n+1$ where each $a_i \in [n]$. In the finding duplicates problem we are asked to output an item that appears at least twice in the array. Observe that by the pigeonhole principle, existence of such item is guaranteed. Here our goal is to design algorithms for finding a duplicate that use small space and do a few passes over the data. In [40] a one-pass randomized algorithm with $O(\log^3 n)$ space was presented which outputs a duplicate with constant probability. In this section, we give another such algorithm that takes only $O(\log^2 n)$ space. Further, we show that our result is optimal; namely, any one pass algorithm for this problem requires $\Omega(\log^2 n)$ space.

Before restricting our attention to streaming algorithms, let us see what happens when we have random access to the array. We will assume a RAM model with words of size $O(\log n)$ bits, where we can access any position in the array in constant time. In this case, the following simple and elegant linear time algorithm, attributed to Robert W. Floyd, finds an answer using 2 words of space. Consider the array a as a directed graph on $n+1$ vertices that are named by integers $1, \ldots, n+1$. We imagine that there is an

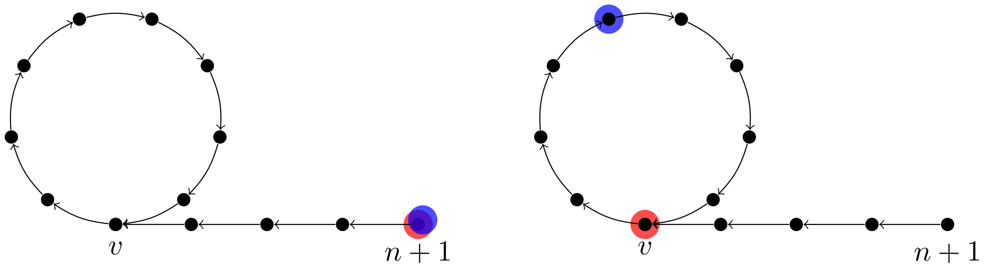

Figure 5.1: An example with $l = 4$ and $k = 9$. The left picture shows the initial placement of the chips and the right picture shows the chips when the red chip reaches v.

edge from node i to node a_i for each $i \in [n+1]$. The algorithm goes as follows. We place two distinct chips on node $n+1$, say red and blue, and in each time step we advance the red chip by following one outgoing arrow and advance the blue chip by following two outgoing arrows in a row. Eventually, two chips will meet again, at which point we move the blue chip back to node $n+1$. After this, in each time step, we advance both the red and the blue chip by one arrow, until they meet again. To see why the chips will meet again, let v be the first node that the red chip visits for the second time (i.e., where the cycle begins), let l be the number of arrows that the red chip took to reach v for the first time, and finally let k be the number of arrows the red chip takes between every subsequent visit to v. Observe that when the red chip reaches v for the first time, the blue chip is $l \pmod{k}$ arrows away from v. Hence, after $k - l \pmod{k}$ more moves the two chips will meet somewhere in the cycle. Notice that following l more links from the meeting point goes back to v. Therefore, after we move the blue chip back to node $n+1$, both chips are $l \pmod{k}$ arrows away from node v and hence the final encounter of the two chips happens at v, whose label corresponds to a number that appears at least twice in the array as desired (since there are two incoming edges). This algorithm runs in $O(n)$ time using $O(\log n)$ bits of space.

5.1.1 Deterministic Algorithms

Now let us return back to the streaming model where we are only allowed left to right passes over the array. All deterministic algorithms presented here are folklore. 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 pigeonhole 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 pigeonhole 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.

Finding duplicates in streams was first considered in the context of fraud detection in click streams [67]. Muthukrishnan in [72] 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 [82], 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.

5.1.2 Randomized Algorithms

Finally in [40], Gopalan and Radhakrishnan gave a one-pass $O(\log^3 n)$ bits randomized algorithm with constant failure rate. The authors of [40] also conjectured that there is a $O(\log n)$ bits algorithm. Here we settle the one-pass complexity of the finding duplicates problem by giving an $O(\log^2 n)$ space algorithm via a direct application of our L_1 sampler from Chapter 4, and by giving an $\Omega(\log^2 n)$ lower bound afterwards.

Theorem 5.1. *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.*

Proof. Let x be an n-dimensional vector, initially zero at each coordinate. We run the L_1-sampler of Theorem 4.1 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 $i = 1, \dots n$ 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. □

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

Theorem 5.2. *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 3.4). 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 3.5, 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 3.5, any algorithm for finding duplicates must use $\Omega(\log^2 n)$ bits. □

5.1.3 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. [40] 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 5.4 that the space bound of our algorithm is best possible.

Theorem 5.3. *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 5.1; 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 2.6 with parameter $5s$ and the $1/2$ relative error L_1-sampler of Theorem 4.1 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 2.6 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 4.4, totalling $O(s \log n + \log^2 n \log(1/\delta))$ bits. □

We remark the upper bounds given in the above theorem and Theorem 5.1 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 5.1 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 4.1 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 5.4. *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 5.2 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 $2m + s = n$. Recall that 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 by Theorem 3.7, $R^1(\mathrm{FCE}_s^m) = \Omega(s \log(m/s))$ 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. □

5.1.4 Finding Duplicates in Longer Streams

Let us consider finally the version of the duplicates problem where we have a stream of length $n + s > n$ over the alphabet $[n]$. Our lower and upper bounds are even farther in this case. A duplicate can be found using $O(\min\{\log^2 n, (n/s) \log n\})$ bits of memory in one pass with constant probability as follows. If we sample a random item from the stream, it will appear again unless that was the last appearance of the letter. As there are at most n last appearances in the stream of length $n+s$, the probability for a uniform random sample to repeat later is at least $s/(n+s)$. Therefore, if $n/s < \log n$, we can sample $4\lceil n/s \rceil$ items from the stream uniformly at random and check if one of them appears again to obtain a constant error algorithm for finding duplicates. If on the other hand $n/s \geq \log n$, we use the algorithm in Theorem 5.1.

Combining our lower bound for the original version of the duplicates problem with the $\Omega(\log n)$ lower bound that follows from Theorem 3.4, we conclude that any streaming algorithm that finds a duplicate in length $n + s$ streams must use $\Omega(\log^2(n/s) + \log n)$ bits.

5.2 Finding Odd-frequency Items

In [66], the authors studied the problem of detecting whether a given sequence of parentheses is well-formed in the streaming model. It is assumed that the given sequence is over an alphabet that consists of several pairs of parentheses, say $(), [], \{\}$ etc., where each pair includes one left and one right parenthesis. A sequence is said to be well-formed if the parentheses in the sequence can be matched in pairs so that in each match-up the the first symbol is a left parenthesis and the second one is the corresponding right parenthesis; further, the substring between every match-up is also a well-formed sequence. For instance, the sequence $[()]()([])$ is well-formed whereas $[(]), (()$ or $][$ are not.

When the alphabet consists of a single pair of parentheses, say $()$, then the problem can be trivially solved in one-pass and $O(\log n)$ bits of space as follows. We keep a counter, initialized to zero, and we increment the counter when a left parenthesis arrives and decrement it when a right parenthesis arrives. If the counter becomes negative at some point, we immediately declare that the sequence is malformed. Otherwise, if the counter is zero when the stream is exhausted, we say that the input is well-formed. Clearly, this algorithm recognizes well-formed strings (and only those) using $O(\log n)$ bits.

By contrast, if there are at least two sets of parentheses in the alphabet, it is easy to see that any deterministic p-pass algorithm must use $\Omega(n/p)$ space, by a reduction from the two-player equality problem. Namely, given a bit string x, Alice replaces each 1 with a (and each zero with a [and Bob replaces each 1 with a) and each 0 with a].

One sees that x and y are equal if and only if the concatenation of the strings Alice and Bob construct is well-formed. In [66], Maginez et al. give an $O(\sqrt{n \log n})$ bits one-pass randomized algorithm that works for any number of parenthesis and decides whether the input is well-formed or not with high probability. Moreover, they prove an elegant lower bound of $\Omega(\sqrt{n \log n})$ bits for any one-pass algorithm that outputs the answer with high probability. Improving on a series of work including [13, 46], it is proven in [14] that in fact for any p-pass algorithm with constant success probability, an $\Omega(\sqrt{n}/p)$ space bound holds as long as all the passes are performed in the same direction.

Detecting whether a sequence of parentheses is well-formed is a fundamental problem that is solved on a daily basis by every web browser, XML parser, code compiler and interpreter. In fact, a very large fraction of the whole data available on the Internet is XML marked and checking the integrity of such data as it is being received or sent over a network is of very high importance. However, this requires $\Omega(\sqrt{n})$ space [66], which can be too costly for small memory devices like routers. Here we consider a relaxation of the above problem. Given a sequence, if there exists an item that appears odd number of times, we want to output one such item. Note that if a certain parenthesis type appears an odd number of times, the sequence cannot be well-formed.

We define the problem formally as follows. Suppose we are given an array $a_1, a_2, \ldots, a_m$ over the alphabet $[n]$, where m is an odd number. In the *odd frequency item problem* we are asked to output a symbol from the alphabet $[n]$ that appears odd number of times in the array. Observe that such symbol is guaranteed to exist, since otherwise every item would repeat even number of times and the array would be of even length.

5.2.1 Deterministic Algorithms

Let us present the following deterministic algorithms, which are all folklore. Similar to the finding duplicates problem, an odd-frequency item can be found in $O(\log n)$ passes using $O(\log n)$ space as follows. Let $h = \lceil n/2 \rceil$ and in the first pass compute the parity of number of i such that $a_i \in [h]$. This can be done by remembering h and keeping a single bit and flipping it whenever the new item is no bigger than h. If the final parity is odd, there exists an odd-frequency item in the array which is between 1 and h. Hence in the subsequent passes we can safely discard elements bigger than h. Otherwise, the parity of the array elements that are between $h+1$ and n is odd (since the array is of odd length) and in the subsequent passes we disregard elements that are in $[h]$. Bisecting the universe $[n]$ in each pass similarly gives us an odd-frequency item after $O(\log n)$ passes. Note that the counter takes only one bit however we still need to remember the upper and lower bounds of the universe we are considering, therefore we use $O(\log n)$ bits of space in each pass.

The above algorithm can be generalized to work in p-passes using $O(n^{1/p})$ space for any $1 \leq p \leq \lceil \log n \rceil$ as follows. In each pass, we divide the alphabet into $\Theta(n^{1/p})$ blocks and count in modulo two how many items appear in the stream from each block. At the end of the pass, there is necessarily a block with odd parity. In the next pass, we take this block as the universe. This is repeated for every pass. Note that in pass p the blocks are singletons, hence at the end of this pass, we locate an odd frequency item.

Next, we observe that the above algorithm is close to being optimal. Namely, we show a $\Omega(n^{1/(2p-1)})$ lower bound for any deterministic p-pass algorithm, through a reduction from the Karchmer-Wigderson communication game for parity. In this game, the first player, Alice, gets a length-n binary string x which has odd number of ones. Bob gets a binary string y with even number of ones. The goal of the players is to agree on a coordinate i such that $x_i \neq y_i$. It is well known [44] that in any r-round protocol for this game there must be message of size $\Omega(n^{1/r})$.

Theorem 5.5. *Any p-pass deterministic algorithm for odd-frequency problem uses $\Omega(n^{1/(2p-1)})$ bits of space.*

Proof. Suppose there is a p-pass s-bits streaming algorithm for odd-frequency problem. Using this algorithm, Alice and Bob can solve the parity communication game in $2p-1$ rounds with messages of size at most s as follows. Alice feeds to the streaming algorithm the positions in which her string is one. Namely, she inputs the set $\{i \mid x_i = 1\}$ to the algorithm. Then, she sends the memory contents of the algorithm to Bob, who similarly inputs the set $\{i \mid y_i = 1\}$ to the algorithm. They repeat this p times, inputting the sets in the same order, until the algorithm is given the data p times. Clearly, this can be done $2p-1$ rounds.

At the end, the algorithm outputs an odd-frequency item i which corresponds to a position where $x_i \neq y_i$ as desired. Hence any p-pass algorithm that finds an odd-frequency item must use $\Omega(n^{1/(2p-1)})$ bits of memory. □

5.2.2 Randomized Algorithms

Next we show that with help of randomization, above results can be improved significantly. Namely, the next theorem gives an $O(\log^2 n)$ space upper bound by appealing to an isolation technique similar to that of the L_0-sampler of Theorem 4.2.

Theorem 5.6. *Let x be an n-dimensional vector defined by an update stream with integer updates. There is a streaming algorithm that, given such update stream, uses $O(\log^2 n \log(1/\delta))$ bits of space and outputs an integer $i \in [n]$, FAIL or EVEN. If all coordinates of x are even, the output is EVEN with probability 1. If the algorithm outputs an integer i, then x_i is even with probability at most $\delta^{\log n} \ll \delta$. The algorithm outputs FAIL with at most δ probability.*

Proof. Let $X_1, \ldots, X_n$ be k-wise independent random variables for $k = \lceil 4e \ln(2/\delta) \rceil$, such that each X_i is uniformly distributed in $\mathbb{F}_{2^t}$. Here, $\mathbb{F}_{2^t}$ is the finite field of size 2^t and $t = \lceil \log n \rceil + 1$. By Theorem 2.5, such variables can be efficiently generated in the streaming model using $O(k \log n)$ bits of space. Consider a field element $\alpha \in \mathbb{F}_{2^t}$ as a binary string of length t. We define vectors $r_0, \ldots, r_{t-1} \in \mathbb{F}_2^n$ as follows. We set the jth coordinate of r_i to one if and only if first i positions of $X_j \in \mathbb{F}_{2^t}$ are all one. Otherwise the coordinate is set to zero. We have $\Pr\left[r_i[j] = 1\right] = 2^{-i}$ for all $0 \leq i < t$ and $j \in [n]$.

Recall that x is the vector defined by the update stream. We consider x as a member of $\mathbb{F}_2^n$, i.e., we perform all updates in mod 2. By Theorem 2.6, there is a random linear map $L : \mathbb{F}_2^n \to \mathbb{F}_2^{4k \log n}$, and a recovery procedure with the following guarantee. On input $L(y)$ the recovery procedure outputs y if y is k-sparse. Otherwise it outputs DENSE with probability at least $1 - n^{-k}$.

Given the update stream, our algorithm simply computes the linear maps $L(x \star r_i)$ for $i = 0, \ldots, t-1$. If recovery procedure outputs the zero vector on all $L(x \star r_i)$, we output EVEN. Otherwise let i_0 be the greatest integer such that the recovery procedure returns a non-zero vector y_{i_0} on $L(x \star r_{i_0})$. If there is such an i_0, our algorithm outputs a non-zero coordinate of y_{i_0}. Otherwise the algorithm outputs FAIL.

Each linear map $L(x \star r_i)$ takes $O(k \log n)$ bits by Theorem 2.6 and we have $O(\log n)$ such maps. To generate r_i for $i = 0, \ldots, t-1$ we need only $O(k \log n)$ bits altogether. Hence the space usage is $O(\log^2 n \log(1/\delta))$ bits. If each item appears even number of times, x is the zero vector hence the recovery procedure returns 0 in each invocation. In this case we output EVEN with probability 1 as desired. Suppose that the algorithm found an i_0 such that the recovery procedure returns a non-zero vector y_{i_0} on $L(x \star r_{i_0})$. Let j be a positive coordinate of y_{i_0}. Observe that x_j is even only if $x \star r_{i_0}$ is not k-sparse and the recovery procedure fails to detect this. This has probability at most $\delta^{\log n}$ by Theorem 2.6 and our choice of k.

Let us finally bound the probability that our algorithm outputs FAIL. Let $h = \mathrm{wt}(x)$. We argue that there exists a $0 \le i_1 < t$ such that

$$\frac{k}{4e} < \mathbb{E}[\mathrm{wt}(x \star r_{i_1})] \le \frac{k}{2e}.$$

This is true, as otherwise there is an i such that $h2^{-i} \le k/(4e)$ and $k/(2e) < h2^{-i+1}$, which is a contradiction. Let E_1 be the event that there is an $i \in [i_1 + 1..t-1]$ such that $\mathrm{wt}(x \star r_i) > k$ and the sparse recovery procedure fails to detect this. Let E_2 be the event that $\mathrm{wt}(x \star r_{i_1}) = 0$ or $\mathrm{wt}(x \star r_{i_1}) > k$. Note that if none of these events happen, $i_0 = i_1$ and $x \star r_{i_1}$ is k-sparse hence the algorithm outputs a correct answer. Probability of E_1 is easy to bound: the sparse recovery procedure fails to detect a vector that is not k-sparse only with $\delta^{\log n}$ probability. By a union bound over all $i \in [i_1 + 1..t-1]$, we see that $\Pr[E_1] < \delta/2$. Now we bound the probability of event E_2. Since the coordinates of r_{i_1} are k-wise independent, by the concentration bounds of Schmidt et al. (Theorem 2.3 (i)) we have

$$\Pr[E_2] \le 2^{-h\,\mathbf{D}_2(k/h \,\|\, k/(2eh))} + 2^{-h\,\mathbf{D}_2(0 \,\|\, k/(4eh))} \le \delta/2$$

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

Further, Theorem 5.6 is essentially tight as shown next.

Theorem 5.7. *Any 1-pass streaming algorithm that solves the odd frequency item problem with constant error probability must use $\Omega(\log^2 n)$ bits of space.*

Proof. We show the claim by a reduction from the universal relation (see Section 3.4). In the universal relation problem, two players, Alice and Bob are given strings, respectively x and y with the guarantee that $x \ne y$. Their goal is to find an i such that $x_i \ne y_i$.

Suppose there is an s-bit streaming algorithm A for finding an odd-frequency item. We give an $s+1$ bit one-way protocol for the universal relation problem. Using the shared randomness, players pick a uniformly at random binary string r. Alice feeds to A the set $S = \{i \mid x_i r_i = 1\}$ and sends memory contents of A to Bob, along with the single bit integer $|S| \mod 2$. Let $T = \{i \mid y_i r_i = 1\}$. If $|S| + |T|$ is an even number, the protocol fails. This has probability exactly 1/2. Otherwise Bob feeds T to A. The algorithm, if it does not err, outputs an odd frequency item i, which corresponds to a mismatch between x and y. However by Theorem 3.5 any one-way protocol for the universal relation with constant success probability requires $\Omega(\log^2 n)$ bits and hence $s = \Omega(\log^2 n)$. This completes the proof. $\square$

Chapter 6

Pattern Matching and Detecting Periodicity in Item Streams

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

The study of periodic sequences and patterns has been important in many fields such as algorithms, data mining, and computational biology. Applications involving weather patterns, stock market data mining, intrusion detection, etc. (e.g., see [28]) aim to identify self-similar trends in large data in almost real time. Periodicity also generated fundamental algorithmic tools for solving problems on sequences/strings. For instance, the textbook Knuth-Morris-Pratt algorithm [61] computes the periods of all prefixes of the pattern in its preprocessing stage. Periodicity remained central to many pattern matching algorithms to this day [19, 2].

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*.

Given the intimate relationship between periodicity and pattern matching, we first investigate space efficient solutions for finding patterns. Recently Porat and Porat, in a breakthrough result, presented a polylogarithmic space randomized algorithm for pattern matching that does not require the storage of the entire pattern [74]. Briefly, given a pattern u of length m, in an off-line step, they preprocess u and build $O(\log n)$ bits sketches for $\log m$ prefixes of u (of geometrically increasing sizes) and use them to find occurrences of the pattern in the length n stream. It is tempting to use the above result to determine the exact period of the stream by searching various prefixes of the stream in the remainder of it, however due the offline processing stage, the algorithm of [74] does not lend itself to an efficient method to find periods. Before giving our periodicity algorithm, we first develop a simple and more streaming-friendly algorithm for pattern matching. While our solution utilizes ideas similar in essence to those used by [74], it does not require an offline preprocessing stage. In fact we show that taking only the Rabin-Karp fingerprints (defined in Section 6.1) of $u[1,2], u[1,4], \dots, u[1,2^i], \dots$ is sufficient to get the same $O(\log n \log m)$ bit space bound. Moreover, our pattern matching algorithm enjoys a very clean and simple description.

Armed with a pattern matching algorithm that works fully in the streaming model, next we develop a randomized streaming algorithm for computing the period of s. Our algorithm makes a single pass over s and uses $O(\log^2 n)$ space to find the period of s granted that s is periodic, otherwise it reports that s is aperiodic. The limitation in computing the period for aperiodic sequences turns out to be necessary as we later prove that computing the period in 1-pass for aperiodic sequences requires linear space. On the other hand we show that an additional pass will give us a $O(\log^2 n)$ space solution for periods of any length.

In addition to periodicity, our pattern matching algorithm enables us to get sublinear solutions for frequency moments defined over substrings. In real-world applications, periodic trends may be hidden or mixed with noise; thus, where exact periodicity is hard to capture, one is likely to encounter instances where a stream is close to periodic. Hence, measures that capture approximate periodicity are natural investigate. In this direction we study distance to periodicity under Hamming distance: we define the distance of s to p-periodicity as the minimum number of character substitutions required to make s p-periodic.

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

It turns out that $D_p(s)$ can be expressed as a product-sum of a certain function defined over rows of a matrix $A_{p\times d}$ where $n/p = d$. The problem then is to compute $L_1 \circ F_1^{res(1)}(A) = \sum_{i=1}^{p} F_1^{res(1)}(A_i)$ where A_i is the ith row of A and $F_1^{res(1)}(s)$, known as the *residual tail* of sequence s, equals $|s| - F_\infty(s)$. In general $F_k^{res(r)}(s) = \sum_{i>r}^{m} f_i^k$, where $f_1, \dots, f_m$ are the character frequencies in decreasing order. Note that when $r = 0$ this is the same as F_k, the kth frequency moment of s.

While there are space efficient algorithms for approximating $F_1^{res(1)}$ and $F_2^{res(r)}$ [15, 37, 9], aggregate computation of $F_1^{res(1)}$ over multiple streams has a different nature and is a new challenge. For this problem, we present two 1-pass randomized algorithms. The first algorithm is obtained by reducing the problem to computing L_0 difference of two vectors that are generated on the fly. This algorithm approximates $L_1 \circ F_1^{res(1)}(A)$ within $2 + \epsilon$ factor using $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon} \log n)$ bits of space.

Our second algorithm returns a $1 + \epsilon$ approximation of $L_1 \circ F_1^{res(1)}(A)$ using $\tilde{O}(\frac{1}{\epsilon^{10.5}})$ space. To get such a result, we apply a combination of the first algorithm, sampling, and exact sparse recovery. Briefly, the idea is that while the first algorithm computes an almost accurate estimation of the contribution of the light-weight rows (i.e., rows with $F_1^{res(1)}(A_i) \leq \epsilon d$), approximation guarantee of our estimator degrades to 2 for heavy weight rows (i.e., rows with $F_1^{res(1)}(A_i) > \epsilon d$). On the other hand using sampling and a sparse recovery procedure we can get a better estimate of the heavy weight rows and thus we will be able to compute a finer approximation of the total weight.

Related Work Aside from the implicit implications of [74], at the time of publication, [29] by Ergün, Jowhari and Sağlam was the first to investigate periodicity in the streaming model, to the best of our knowledge. Subsequent to [29], Breslauer et al. [12] gave 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, our algorithm takes $O(\log m)$ time per item, hence is slower. Also subsequently, Crouch et al. studied the distance periodicity problem [25]; 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 $O(\epsilon^{-2}\,\mathrm{polylog}\, n)$ space.

In a related direction, Ergün et al. [30] 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 [64] showed a lower bound of $\Omega(\sqrt{n})$ for testing periodicity in the query model. With a focus on time complexity, Czumaj and Gasieniec [26] presented an average case analysis for computing the exact period.

Bar-Yossef et al. [7] studied the sketching complexity of pattern matching. The work of Indyk et al. [45] 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 [28]), typically with quite different space considerations than in our model. Streaming complexity of cascaded norms $L_k \circ L_p$ over matrices is investigated in depth by Jayram and Woodruff in [48]; see also [20, 70].

6.1 Preliminaries

We assume the input stream is a sequence of length n over the alphabet $\Sigma = \{0, 1, \ldots, L\}$. Recall that 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.

Periodicity. A sequence is called 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 called *the* period of s and is denoted $\mathrm{per}(s)$. The following lemma is well-known and a proof of it can be found in [65, 33].

Lemma 6.1 (Lyndon et al. [65]). *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 6.2. *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 6.1, 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 6.1 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 6.1 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 6.2 we use Rabin-Karp fingerprints [59], 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 [59, 74, 12] for the proofs.

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

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

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

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

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 $\phi(s)$.

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 result of Section 6.5. 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 (see Section 2.5). 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 6.1 (Lipsky et al. [75]). *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 $SR_r(A)$ rejects the input. Given the above result, we can implement $SR_r(A)$ in $O(r(\log n + n' \log m))$ bits.

6.2 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 6.3. *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 6.1, 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 6.2, we should have $|a_{l+1} - a_l| = \mathrm{per}(u)$ for all $l \in [h-1]$. As a result, when we reach the position $a_2 + m - 1$, we have obtained the value of $\mathrm{per}(u)$. Now let $x = u[1, \mathrm{per}(u)]$. We show that it is possible to compute $\phi(x)$ when we reach $a_3 + m - 1$. To this end, when we are in $a_1 + m - 1$, starting from the next character we build a fingerprint until we reach $a_2 + m - 1$. This gives us $\phi(s[a_1 + m, a_2 + m - 1])$. Note that if $\mathrm{per}(u)$ divides 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 $r = m \pmod{\mathrm{per}(u)}$. Therefore

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

Likewise, we have $s[a_2 + m, a_3 + m - 1] = x[r+1, \mathrm{per}(u)] \circ x[1, r]$. Therefore, at location $a_2{+}m$, we know the value of r and $\mathrm{per}(u)$, and consequently using this information, we can build the fingerprints $\phi(x[r{+}1, \mathrm{per}(u)])$ and $\phi(x[1, r])$ when we go over $s[a_2{+}m, a_3{+}m{-}1]$. Note that here we have used the properties (P3) and (P4) from Section 6.1. 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 $\mathrm{per}(u)$ from the left-end of it and adding $s[a_{l-1} + k, a_l - 1]$ to the right-end. Therefore we can generate $\phi(s[a_l, a_l + k])$ by having $\phi(s[a_{l-1}, a_{l-1} + k])$, $\phi(s[a_{l-1} + k, a_l - 1])$, and $\phi(x)$. This proves our claim.

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

Our pattern matching algorithm is the result of a recursive application of Lemma 6.3. 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 6.3, 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 6.1 and the observation that we make at most $O(n \log |u|)$ fingerprint comparisons.

Theorem 6.2. *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 6.4. In addition to $M_s(u)$, our algorithm generates $M_s(u[1, 2^i])$ for each $i = 1, \dots, \log m$, which leads to further space economy in our periodicity algorithms in the next section. By contrast, we show in the following theorem that any pattern matching algorithm that generates $M_s(u[1, 2^i])$ for $\Omega(\log |u|)$ prefixes of u of geometrically increasing sizes must use $\Omega(\log |u| \log n)$ bits of space.

Theorem 6.3. *Any one-pass streaming algorithm that generates $M_s(u[1, 2^i])$ for $i = 1, \dots, \lfloor \log m \rfloor$ uses $\Omega(\log m \log(n/m))$ space.*

Proof. Suppose Alice and Bob are given an instance of the AIND_k^h problem, where $h = \lfloor \log m \rfloor$ and $k = \lceil n/m \rceil$. Recall that Alice gets a string $s \in [h]^k$ and Bob gets an integer i and s_j for $j < i$. Bob is required to output s_i.

Alice creates a pattern x by appending 2^{j-1} copies of s_j next to each other for each $j = 1, \dots, h$. Namely,

$$x = s_1 \circ s_2^2 \circ \dots \circ s_h^{m/2}.$$

Bob creates $k = \lceil n/m \rceil$ strings $y_1, \dots, y_k$, each of length at most m as follows.

$$y_l = t_1 \circ t_2^2 \circ \dots \circ t_2^{2^{i-2}} \circ l^{2^{i-1}}$$

Bob's text is the concatenation of all y_l for $l = 1, \dots, k$. Given a pattern matching algorithm with space S, Alice feeds x to the algorithm and sends the memory contents of the algorithm to Bob. Then Bob continues the simulation of the algorithm and feeds the text he constructed. Observe that the output of the algorithm reveals s_i. Hence we have a one-way S bits protocol for the augmented indexing problem. Hence by Theorem 3.1, we have $S = \Omega(h \log k) = \Omega(\log m \log(n/m))$, as desired. □

6.3 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 $\mathrm{per}(s) \leq n/4$ or not as follows.

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

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

The reason that we only perform the test for $\min(T \cap [1, n/4])$ is a consequence of Lemma 6.2. 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 6.4. *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 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 6.2 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 6.2, 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 6.2, 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

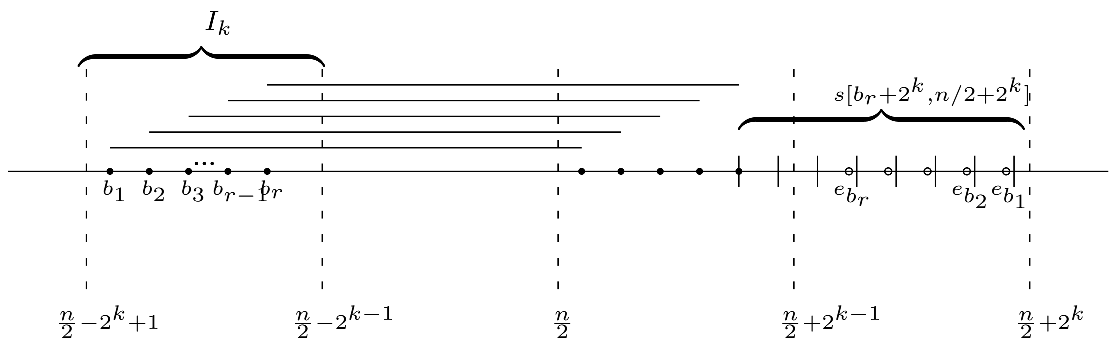

Figure 6.1: A sample run of the algorithm in Section 6.3.

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 6.4, 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 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 $|I_k| \le \frac{1}{2}2^k$, 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 6.4. *If $y < l$ and $|r - j| + y \neq x$ then $b_j \notin T$.*

By Claim 6.4, $|H_k \cap T| \le 1$. Consequently it is enough to maintain $\phi(s_{b_j})$ where $|j - r| + y = x$ and $\phi(s_{b_1})$. So in this case $|R_k| \le 2$.

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 6.4). 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 6.5. *There is a 1-pass randomized streaming algorithm that given $s \in \Sigma^n$ outputs* 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 6.6. *Every 1-pass randomized algorithm that computes* 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 per$(a \circ b)$. We show that any one-way protocol that computes per$(a \circ b)$ requires $\Omega(n)$ communication by a reduction from the augmented indexing problem (see Section 3.2). Suppose Alice and Bob are given an instance of AIND_2^n. Recall that 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, per$(a \circ b) = i$ if and only if $x[i+1] = 1$. Hence by Theorem 3.1 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 per(s) can be converted to a protocol for the above communication game. □

6.4 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 [55, 56, 36] 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 [1] 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 [1] 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 6.7. *There exists a 1-pass streaming algorithm that $(1+\epsilon)$-approximates $F_{0,d}$ using $\tilde{O}(\frac{1}{\epsilon}\sqrt{n})$ space.*

[2]An easy information theoretic observation shows that sliding a fingerprint for d-substrings that preserves equality with high probability requires $\Omega(d)$ 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 6.2.

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 less than or equal to $t-1$. Using Lemma 6.1, 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 $\mathrm{per}(s_j)$. Consequently $\mathrm{per}(s_j) = \mathrm{per}(s_l)$. Therefore $\mathrm{per}(s[j, l+d-1]) = \mathrm{per}(s_j) \leq t-1 = F_{0,d}(s)$. 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. □

6.5 Approximating the distance to periodicity

Recall that $D_p(s)$ is the minimum number of character changes on $s \in \Sigma^n$ to make it p-periodic. Assume WLOG that p divides n where $n = dp$, and view s as a $p \times d$ matrix A where $A(i,j) = s[(i-1)p + j]$. If p does not divide n, s can be represented by two matrices. Then, $D_p(s)$ is the the minimum number of substitutions in A to make every row consist of d repetitions of the same character. Also, $D_p(s) = L_1 \circ F_1^{res(1)}(A) = \sum_{i=1}^{p} F_1^{res(1)}(A_i)$. It is challenging to compute this quantity since we receive A in the column order: $A(1,1),\ldots,A(p,1),A(1,2),\ldots,A(p,2),\ldots$ To compute $L_1 \circ F_1^{res(1)}(A)$ exactly, one can compute the residual tail of each row in parallel using independent counters, in $O(|\Sigma|p)$ words of space. On the other hand, one can estimate $F_i^{res(1)}(A_i)$ within $1-\epsilon$ factor in $O(1/\epsilon)$ words of space in several ways. For instance, using the Heavy Hitters algorithms in [69, 10] we can approximate $F_\infty(A_i)$ with additive error $\epsilon F_1^{res(1)}(A_i)$, giving the following bound.

Theorem 6.8. *There is a deterministic streaming algorithm that approximates $L_1 \circ F_1^{res(1)}(A)$ within $1-\epsilon$ factor using $O(\frac{p}{\epsilon})$ words of space.*

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

6.5.1 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 6.9. $F(A_i) \geq \frac{1}{d}\sum_{a<b} f_i(a)f_i(b) \geq \frac{1}{2}F(A_i)$.

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. [55].

Theorem 6.10. [55] *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 6.10 and Fact 6.9, we get a $2+\epsilon$ approximation for $F(A)$ in space $O(1/\epsilon^2 \log(1/\epsilon)\log(n))$ bits. However, per-item processing time is $\Omega(d)$. To overcome this, we pick a random 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 6.11. *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* $1/4$.

6.5.2 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 6.12. *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

$$F'(A) + \frac{1}{2d}\sum_{i=1}^{p}(F(A_i)^2 + F_2^{res(1)}(A_i)) = F(A). \tag{6.1}$$

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 (6.1), 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 6.12 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 6.12 and (6.1).

$$F(A) \geq F'(A) + \sum_{i \in H_{\epsilon d}} F''(A_i) \geq (1-\epsilon)F(A). \tag{6.2}$$

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.

The overall algorithm. 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. At the end, we state the considerations about the space usage and the final theorem.

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 6.2: Description of $\varphi(B_i)$

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 6.5.2. 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 the exact sparse recovery procedure that we described in the preliminaries section. 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 6.3.

We analyze 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 6.5. *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 [8] (cf. Lemma 5.3) to estimate F_2. From [8], 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 $1 - p/(8n)$,

$$H_{(\epsilon+\delta)d} \subseteq G_{\epsilon d}, \quad ([p] \setminus H_{(\epsilon-\delta)d}) \cap G_{\epsilon d} = \emptyset \tag{6.3}$$

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, (6.2), and (6.3) together we get the desired statement.

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 (6.3) 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 6.6. *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,

$$(1+\epsilon)^{\hat{j}} < (1 + \Theta(\epsilon))|G_{\epsilon d}| \tag{6.4}$$

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 6.3: Description of the sampling procedure.

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

$$\Pr(|v_{\hat{j}} - \frac{t_1}{|G_{\epsilon d}|}\sum_{i\in G_{\epsilon d}} \varphi(B_i)| > \epsilon \frac{t_1}{|G_{\epsilon d}|}\sum_{i\in G_{\epsilon d}} \varphi(B_i)) < \exp(-\frac{\epsilon^4 t_1}{8}) < \frac{1}{n^2}. \tag{6.5}$$

This and (6.4) 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 (6.4) and (6.5), 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

$$\Pr\left(|r_{j',u} - \mathbb{E}[r_{j',u}]| > s\,\mathbb{E}[r_{j',u}]\right) \leq \frac{1}{s^2}(\frac{(1+\epsilon)^{j'}}{|G_0|} + \frac{1}{q-1}). \tag{6.6}$$

Now we observe that, by part (II) in Lemma 6.5, $\mathbb{E}[r_{j',u}] \leq \frac{32k}{\epsilon}$. After plugging this in (6.6) 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 6.5, Lemma 6.6, and (6.2) 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 6.13. *There is a randomized 1-pass streaming algorithm that outputs a $1 \pm \epsilon$ approximation of $L_1 \circ F_1^{res}(A_{p\times d})$ with probability at least $3/4$ using $O(\frac{1}{\epsilon^{10.5}}\log^5 n)$ words of space.*

Bibliography

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

[2] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 794–803, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[3] Alex Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *FOCS*, 2011.

[4] Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. In *SODA*, pages 1190–1197, 2010.

[5] László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *FOCS*, pages 337–347, 1986.

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

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

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

[9] Radu Berinde, Graham Cormode, Piotr Indyk, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '09, pages 157–166, New York, NY, USA, 2009. ACM.

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

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

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

[13] Amit Chakrabarti, Graham Cormode, Ranganath Kondapally, and Andrew McGregor. Information cost tradeoffs for augmented index and streaming language recognition. In *FOCS*, pages 387–396, 2010.

[14] Amit Chakrabarti and Ranganath Kondapally. International workshop on randomization and computation. In *RANDOM*, 2011.

[15] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

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

[17] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity (extended abstract). In *FOCS*, pages 429–442, 1985.

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

[19] Richard Cole and Ramesh Hariharan. Approximate string matching: a simpler faster algorithm. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, SODA '98, pages 463–472, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[20] Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 271–282, New York, NY, USA, 2005. ACM.

[21] Graham Cormode, S. Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, pages 25–36, 2005.

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

[23] Graham Cormode, Mike Paterson, Süleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 197–206, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

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

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

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

[27] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54(6), 2007.

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

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

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

[31] Robert M. Fano. *Transmission of Information: A Statistical Theory of Communication.* MIT Press, March 1961.

[32] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23:1001–1018, October 1994.

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

[34] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31:182–209, September 1985.

[35] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, SCG '05, pages 142–149, New York, NY, USA, 2005. ACM.

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

[37] Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Practical algorithms for tracking database join sizes. In *FSTTCS*, pages 297–309, 2005.

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

[39] E. N. Gilbert. A comparison of signalling alphabets. *Bell Syst. Tech. J.*, 31:504–522, 1952.

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

[41] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability Theory.* 1997.

[42] Johan Håstad and Avi Wigderson. The randomized communication complexity of set disjointness. *Theory of Computing*, 3(1):211–219, 2007.

[43] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[44] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20. ACM, 1986.

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

[46] Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions. Technical Report TR10-071, Electronic Colloquium on Computational Complexity, `http://eccc.hpi-web.de/`, April 19 2010. Revised July 5, 2010.

[47] T. S. Jayram. Information complexity: a tutorial. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 159–168, New York, NY, USA, 2010. ACM.

[48] T. S. Jayram and David P. Woodruff. The data stream space complexity of cascaded norms. In *FOCS*, pages 765–774, 2009.

[49] T. S. Jayram and David P. Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with sub-constant error. In *SODA*, pages 1–10, 2011.

[50] J. Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30:175–193, 1906. 10.1007/BF02418571.

[51] A. Joffe. On a set of almost deterministic k-independent random variables. *Annals of Probability*, 2(1):161–162, 1974.

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

[53] Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM J. Discret. Math.*, 5:545–557, November 1992.

[54] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 745–754. ACM, 2011.

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

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

[57] Mauricio Karchmer. *A New Approach to Circuit Depth.* PhD thesis, MIT, 1989.

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

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

[60] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms.* Addison-Wesley, 1969.

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

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

[63] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 614–623, New York, NY, USA, 1998. ACM.

[64] Oded Lachish and Ilan Newman. Testing periodicity. In *APPROX-RANDOM*, pages 366–377, 2005.

[65] R. C. Lyndon and M. P. Schutzenberger. The equation am=bncp in a free group. 1962.

[66] FrÃ©dÃ©ric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. In Leonard J. Schulman, editor, *STOC*, pages 261–270. ACM, 2010.

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

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

[69] Jayadev Misra and David Gries. Finding repeated elements. Technical report, Ithaca, NY, USA, 1982.

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

[71] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21:840–842, October 1978.

[72] S. Muthukrishnan. *Data Streams: Algorithms and Applications.*

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

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

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

[76] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106:385–390, December 1992.

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

[78] Pranab Sen and Srinivasan Venkatesh. Lower bounds for predecessor searching in the cell probe model. *CoRR*, cs.CC/0309033, 2003.

[79] D. V. Smirnov. Shannon's information methods for lower bounds for probabilistic communication complexity. Master's thesis, Moscow University, 1988.

[80] Pantelimon Stanica. Good lower and upper bounds on binomial coefficients. *J. Inequalities in Pure and Applied Math.*, 2(3), 2001.

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

[82] Jun Tarui. Finding a duplicate and a missing item in a stream. In Jin-Yi Cai, S. Cooper, and Hong Zhu, editors, *Theory and Applications of Models of Computation*, volume 4484 of *Lecture Notes in Computer Science*, pages 128–135. Springer Berlin / Heidelberg, 2007.

[83] David Woodruff and T. S. Jayram. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with low error. In *SODA*, 2011.

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