EFFICIENT DELEGATION ALGORITHMS FOR OUTSOURCING COMPUTATIONS ON MASSIVE DATA STREAMS

VED PRAKASH

NATIONAL UNIVERSITY OF SINGAPORE

2015

EFFICIENT DELEGATION ALGORITHMS FOR OUTSOURCING COMPUTATIONS ON MASSIVE DATA STREAMS

VED PRAKASH
(B.Sc.(Hons), NUS)

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

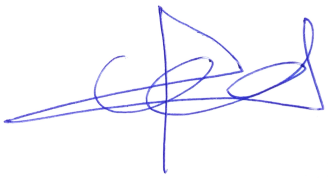
CENTRE FOR QUANTUM TECHNOLOGIES
NATIONAL UNIVERSITY OF SINGAPORE

2015

Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.



VED PRAKASH

July 20, 2015

Acknowledgements

I would like to express my sincere appreciation to my advisor Hartmut Klauck, who has been a remarkable mentor. He has been extremely encouraging and that tide me through this trying journey. The relentless support has also allowed me to grow as a research scientist. His meticulous supervision has provided me much guidance on both my research and career path fronts. I would like to express my gratitude for his thorough reviews throughout the course of the preparation of this thesis. I would also like to thank my thesis committee members, Rahul Jain and Frank Stephan for their provision and guidance in the foundational years of my PhD studies.

I would like to thank the Centre for Quantum Technologies (CQT) for giving me the opportunity to be able to receive this education under extremely privileged circumstances. This dissertation would not have been possible without the funding from CQT.

Following, I would like to thank all of my friends who have been cheering me on via various channels, for they were the sources of motivation for me to strive towards my goal. Words cannot express how grateful I am to my parents for all of the sacrifices they have made. Most imperatively, I would like to express my utmost appreciation to my beloved wife, Ong Phyllis, who spent sleepless nights with me while I penned down my ideas. She has also always been my support in times when there was no one to answer my queries.

Table of Contents

Summary

In numerous real world applications, one needs to store almost the whole data set in order to compute certain functions of the data, where we require the answer to be exact or even approximate in some cases. This thesis will examine a model for data streaming algorithms where we engage the services of external third parties to do difficult computations for the client. The main motivating application of this is cloud computing, where we not only require the cloud to store the massive data set, but execute computations on the data set and communicate the results to the client as well. The client should be able to verify the correctness of the result within his computational restrictions. We will discuss algorithms to achieve this in different streaming models, depending on the interaction between the client and the external third party who is also called the prover.

The communication complexity model augmented with a prover is a very important tool used to analyze the theoretical properties of the data streaming model with a prover. We use this to give an improved lower bound for approximating the frequency moments in the annotation streaming model, where there is a single help message from the prover after the stream has ended. We also investigate a restricted version of this model and show lower bounds in this restricted model. We will use our lower bounds to study the theoretical properties of the streaming model with a prover, where the prover and the client are allowed to interact.

We give an improvement of previous work in [30] which requires $\widetilde{O}(\sqrt{n})$ communication between the prover and the client to compute the number

of distinct elements exactly using $O(\log m)$ messages, where n is the length of the stream and m is the size of the universe. Our algorithm gives an exponential improvement on the total communication needed while maintaining the same number of messages exchanged.

We also investigate a new streaming model that only bounds the communication *overhead*, i.e., the amount of communication sent from the prover to the client per symbol of the data stream. This streaming model is different from previous models defined in [20, 21, 28–30, 56, 102]. We will design algorithms for four different streaming problems in this new model. For one of these streaming problems (perfect matching problem), there is no known efficient interactive streaming algorithm in the previous models [29, 30, 52]. We will analyse the limitations of this new model. We show that the verification phase with a large number of communication rounds between the prover and the client after the stream has ended is unavoidable for certain problems in a restricted streaming model where the messages from the client to the prover are just some of his random bits.

List of Figures

List of Symbols

We list the standard notations and terminology that we will be using in this thesis. They will not be formally defined in this thesis.

$\log x$	binary logarithm of x.
$wt(x)$	Hamming weight of string x.
$\text{polylog}(n)$	$O\left((\log n)^{O(1)}\right)$.
$\text{polylog}(m,n)$	$O\left((\log n)^{O(1)}(\log m)^{O(1)}\right)$.
quasilinear (in n)	$n\,\text{polylog}(n)$.
$\mathbf{R}$	set of real numbers.
$\mathbf{N}$	$\{0,1,2,\cdots\}$.
$\mathbf{Z}^+$	$\{1,2,\cdots\}$.
$\mathbf{Z}$	set of integers.
$\mathbf{F}_q$	finite field of size q, where $q=p^n$ for some prime p and $n\in\mathbf{Z}^+$.
$\mathbf{F}_q^*$	multiplicative group of the nonzero elements in $\mathbf{F}_q$.
$\mathcal{M}_{m,n}(S)$	$m\times n$ matrix where the $(i,j)^{th}$ entry is from the set S. If $m=n$, we simply write $\mathcal{M}_m(S)$.
I_m	identity matrix of size $m\times m$.
$\mathbf{GL}_m(R)$	general linear group of m by m invertible matrices over a commutative ring R, with identity.
$[m]$	$\{1,\cdots,m\}$.
$A\subsetneq B$	$A\subseteq B$ but $A\neq B$.

Chapter 1

Introduction

This thesis is mainly based on the following papers.

- Hartmut Klauck and Ved Prakash. Streaming computations with a loquacious prover [73]. *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 305–320, 2013.

- Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem [74]. In *Automata, Languages, and Programming - 41st International Colloquium*, ICALP 2014, pages 919–930, 2014.

- Ved Prakash. Efficient delegation protocols for data streams [91]. In *Proceedings of the 2014 SIGMOD PhD Symposium*, SIGMOD'14 PhD Symposium, pages 6–10, 2014.

There are two other results mentioned in this thesis which do not appear in the papers listed above. In Corollary 3.3.10, we improve the lower bound given in [21] for approximating frequency moments in the annotation model. In Section 4.2, we show that "IP=PSPACE" holds for online communication complexity classes.

Data streaming algorithms are designed to process massive data sets arriving one at a time in an online fashion, i.e. with small time overhead. The space used by these algorithms should be minimal. Due to the enormous amount of data being generated in this century, designing efficient streaming algorithms and models to handle these huge data sets are important areas to explore. Some of the interesting problems studied in the data stream model include frequency moments which we will define formally in Chapter 2 and graph problems like matching and triangle counting [14,38].

We denote the length of the stream by n where each symbol is drawn from a universe of size m. Many interesting problems in the data stream model (e.g. third or higher frequency moments [6,18]) require large space to even give a constant factor approximation. Due to such limitations in the standard streaming model, more powerful models have been studied which introduces a third party who processes the stream and provides the answer together with a proof of correctness after the stream has ended [20–22, 28–30, 56, 74, 101, 102]. We view the third party as the helper who convinces the client of the correct answer. The client has the usual small space requirement but the helper can store the whole data stream. To make the model realistic, the helper is online in the sense that he cannot predict the future parts of the stream. The help provided should be short and inexpensive to check as well. How can the client be convinced that the results produced by the third party are correct? Ideas from the theory of interactive proofs are used to reject a claim by a dishonest helper with high probability. Throughout this thesis, we will refer to the helper as the prover, and to the client as the verifier.

There are many reasons for using the services of third parties to execute computations for the verifier. One obvious reason would be that the verifier

does not have the resources (mainly due to space constraints) to execute the computations by himself. If one generates massive data only once in a while, it is more practical to rent some hundreds of computers for a few hours and get the third party to do the necessary computations. The cost of buying hundred such computers is too costly and a waste of resources if they are not used frequently. They are many internet companies with enormous data warehouses and powerful computers that offer cloud computing services.

The main aim of this thesis is to study the power and limitations of algorithms in different models for delegating computations on data streams to third parties.

1.1 Structure of this Thesis and Contributions Made

- In Chapter 2, we introduce both the data streaming and one-way communication complexity models. We show how results from communication complexity can be used to prove space lower bounds for streaming algorithms. The purpose of this chapter is to show that many interesting problems in the data streaming model cannot be solved in sublinear space, which will motivate the models that we will introduce in the subsequent chapters.

- In Chapter 3, we introduce the annotation model for verifying computations on data streams, which was first introduced in [21]. In this model, the prover provides an annotation/proof to the verifier after the data stream has ended. The proof is processed by the verifier in a streaming fashion and the verifier is allowed to use randomness to

process the proof stream. As a warmup, we give simple annotation protocols based on fingerprinting techniques before giving annotation protocols for the exact computations of the frequency moments F_2, F_0 and F_∞. The main purpose for doing so is to illustrate that we can obtain sublinear annotation protocols for problems which require linear space in the standard streaming model, which is the model without the prover.

We introduce the Merlin-Arthur communication complexity model to address lower bounds for data stream computations with a prover. By analyzing the number-in-hand (NIH) multi-party online Merlin-Arthur communication model, we improve the lower bound given in [21] for approximating F_k in the annotation model. We show lower bounds for the online Merlin-Arthur communication complexity model with k-messages. Our lower bounds follow from well-known round elimination results in the theory of interactive proofs [9, 10]. Our lower bounds for the online Merlin-Arthur communication model with k messages combined with a result from [22] give an exponential separation between the public and private coin streaming models.

- In Chapter 4, we introduce the interactive streaming model which was first defined in [30]. We show that any language $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$ which is in $\mathcal{PSPACE}_{cc}$ (See Definition 4.2.2) has an online Merlin-Arthur protocol with polylog(n) messages and the cost of this protocol is polylog(n). Combining this with Lautemann's theorem which is known to hold in the communication complexity model [8], we get $OMA_{cc}^{\text{polylog}(n)} = \mathcal{PSPACE}_{cc}$.

  We also briefly discuss the $O(\log m)$ round protocol for the exact computation of F_2, F_0 and F_∞ which was first given in [30]. These pro-

tocols are more practical than the generic streaming protocol which is based on circuit checking [52].

- In Chapter 5, we give a streaming interactive protocol with $\log m$ rounds for the exact computation of F_0 using v bits of space and total communication in bits is h, where

$$v = O\left(\log m \left(\log n + \log m \cdot (\log\log m)^2\right)\right) \quad \text{and}$$
$$h = O\left(\log m \left(\log n + \log^3 m \cdot (\log\log m)^2\right)\right).$$

  The update time of the verifier per symbol received is $O(\log^2 m)$. This solves one of the open problems posed by Cormode, Thaler and Yi [30]. Table 1.1.1 gives a summary of the known results for the exact computation of F_0 in the prover-verifier model.

Paper	Space	Total Communication	Rounds
[21]	$m^{2/3}\log m$	$m^{2/3}\log m$	1
[29]	$\log^2 m$	$\log^3 m$	$\log^2 m$
[30]	$\log^2 m$	$\sqrt{m}\log^2 m$	$\log m$
Our work	$\log^2 m(\log\log m)^2$	$\log^4 m\,(\log\log m)^2$	$\log m$

Table 1.1.1: Comparison of our protocol to previous protocols for computing the exact number of distinct elements in a data stream. The results are stated for the case where $m = \Theta(n)$. The complexities of the space and the total communication are correct up to a constant.

- In Chapter 6, we propose a new model which relaxes the restriction placed on the total communication in prior models [29, 30]. Unlike previous works which bound the total communication between the prover and verifier, in our model we only bound the communication *overhead*, which is the amount of machine words exchanged per symbol seen on the data stream. Our new model disallows a lot of communication or rounds of interaction after the stream has ended.

In particular, this makes the prover more efficient and his work is spread out more compared to previous protocols in [29, 30]. The protocols we design are simpler and more efficient as they are not based on interactive proof techniques which have an additional verification phase after the stream ends. In previous works [29, 30], the main conversations take place after the stream has ended and during this verification phase, the prover has to perform exponentially more operations than the verifier. This additional verification phase is not present in our protocols.

We give streaming protocols in our new model for the following four problems: Median, Longest Increasing Subsequence, FULL RANK and perfect matching. The perfect matching problem is not known to be in the complexity class $\mathcal{NC}$, and thus the generic streaming protocol in [29,30,52] does not apply. By relaxing the total communication restriction, we managed to find an algorithm for the perfect matching problem while maintaining the full online nature of streaming. The natural question to ask is whether all functions in $\mathcal{NC}$ can avoid the additional verification phase after the stream has ended. The answer to this is negative in the public coin model. We show that any function with a "strict" reduction from INDEX on n bits cannot be solved in the public coin model, requiring at least $\Omega(\log n/\log\log n)$ rounds of interaction between the prover and verifier after the stream ends in the public coin model, otherwise either the communication complexity after the stream ends increases to above polylog(n) or the space complexity of the verifier increases to above polylog(n). This simply means that the extra verification phase is an inherent feature in the Merlin-Arthur streaming model for protocols solving problems

which have a strict reduction from INDEX, e.g., computing frequency moments. Our lower bounds shed light on both our new model and prior models.

- Chapter 7 concludes this thesis and discusses some related open problems.

Chapter 2

Data Streaming and Communication Complexity

In this chapter, we introduce the models for data streaming and communication complexity. Different communication complexity models have been used to establish lower bounds for data stream problems in different models. In this chapter, we will introduce the standard model as defined in the seminal work of Alon, Matias and Szegedy [6] and look at the limitations of this model.

2.1 The Data Stream Models

The input stream is denoted by $\sigma = \langle a_1, \cdots, a_n \rangle$, where the a_i's are sometimes referred to as symbols in this thesis. The data stream defines a function $A : [N] \rightarrow \mathbf{R}$. The data elements in the stream arrive in an online fashion, and the system has no control over the order in which the data streams arrive. The main objective of data streaming algorithms is to process a massive data set arriving one item at a time in an online fashion, i.e., with small time overhead, while at the same time minimizing

the workspace used by the algorithm. In this thesis, we use the unit cost RAM model to measure the update time per symbol seen on the stream. In this model, each field operation[1] takes unit time. These algorithms are only allowed to have one pass over the data stream and are allowed to be randomized. These algorithms have to output the right answer with some constant probability larger than $\frac{1}{2}$. There are three different types of models which describe the inputs a_i of the stream. We list these three models below and give a motivating example for each of them.

1. Time Series Model. Here $n = N$ and each $a_i = A(i)$ in increasing order of i. For instance, each a_i could be used to model the price of some stock. The data stream gives the price of the stock at different time intervals. After some fixed period of time, we are given a time period $t_1, t_2 \in [n]$ and we need to output $\sum_{i=t_1}^{t_2} a_i$. If $t_1 = t_2$ and each $a_i \in \{0, 1\}$, this is the famous INDEX problem which is defined in Definition 2.2.2.

2. Cash Register Model: Each $a_i = (j, I_i)$ where $I_i \geq 1$. We update $A(j) \leftarrow A(j) + I_i$. Here, multiple a_i's can update the same $A(j)$. This is the most popular data stream model studied. One example would be to count or estimate the number of distinct queries made to a search engine. Each a_i will be the query made to the search engine. The goal is to output the number of distinct elements in the vector A.

3. Turnstile Model: This is similar to the cash register model but we allow I_i to be positive or negative. If we want $A(i) \geq 0$ for all i at all times, we call this the strict turnstile model. This can be used to

[1]Examples of field operations over $\mathbf{F}_p$ where $p = poly(m, n)$ include addition, subtraction, multiplication, division or choosing a random field element.

model insertions and deletions in a database.

In this thesis, we will work in the cash register model where each $a_i = (j, 1)$. *So from now on, our stream is* $\sigma = \langle a_1, \cdots, a_n \rangle$ *where each* $a_i \in [m]$ *unless otherwise stated.* We say that the stream σ has length n where each symbol is drawn from a universe of size m. Ideally, the space used by the algorithm should be sublinear in m and n, and the update time per item a_i on the stream should be polylog(m, n). We will measure the space used by the streaming algorithm in bits.

Definition 2.1.1. (**Streaming Algorithm**)
Let $f : [m]^n \to \mathbf{R}$ be a function and suppose that the input stream is $\sigma = \langle a_1, \cdots, a_n \rangle$ where each $a_i \in [m]$. A streaming algorithm for f is a randomized algorithm which is given a one-pass access to the input stream σ. The algorithm is also given an error parameter ϵ and a confidence parameter $0 \leq \delta < 1$. For any input stream σ, the algorithm is required to output a value in the interval $((1-\epsilon)f(\sigma), (1+\epsilon)f(\sigma))$ with probability at least $1 - \delta$. If $\epsilon = 0$, we say that the streaming algorithm computes the exact value of the function f.

The two main measures of complexity for streaming algorithms are the space (in bits) and the update time per data symbol. Given ϵ and δ, the space is the maximum amount of workspace the algorithm uses over all possible input streams and all the random choices of the algorithm. The update time per data symbol for a given ϵ and δ is the maximum time[2] the algorithm spends on a single symbol a_i of the stream, where the maximum is taken over all $i \in [n]$, all possible input streams and all the random choices of the algorithm.

[2]The time is measured in the unit cost RAM model, which was mentioned previously in this section.

The interested reader is referred to the survey by Muthukrishnan [86]. This survey contains many interesting applications of streaming algorithms and very well motivates this interesting subject.

2.2 Communication Complexity

We introduce some basic definitions and results in communication complexity that will be useful for this chapter. As we proceed to later chapters, we will define different models in communication complexity which will be inspired by the data stream models that we will be defining later.

Two party communication complexity was first introduced in the seminal paper by Yao [107] in 1979. In a two party communication problem, we have two players Alice and Bob who wish to compute a function $f : X \times Y \to Z$. But Alice is only given $x \in X$ and Bob is given $y \in Y$. Note that the function f is known to both of them. Usually, we will consider Boolean functions, i.e. $Z = \{0, 1\}$. Both Alice and Bob will need to communicate between themselves according to some protocol $\mathscr{P}$ (which depends on f) in order to compute the function $f(x, y)$. $\mathscr{P}$ must specify which player needs to communicate at the different stages of the protocol. If the protocol terminates, the output should be $f(x, y)$. At each stage, the message from the player who needs to communicate depends on his (or her) input and the messages exchanged from all the previous stages. Since we are only interested in the amount of communication between Alice and Bob, we allow them to have unlimited computational power.

Given a protocol $\mathscr{P}$ and input $(x, y) \in X \times Y$, the cost of $\mathscr{P}$ on (x, y) is the total number of bits communicated by Alice and Bob according to $\mathscr{P}$ when Alice and Bob are given x and y respectively. We denote this by

$C(\mathscr{P}(x,y))$. The cost of the protocol is defined as

$$C(\mathscr{P}) := \max_{(x,y)\in X\times Y} C(\mathscr{P}(x,y)),$$

which is just the worst case cost over all inputs. The deterministic communication complexity of a function f is the minimum cost over all protocols that compute f correctly.

Now, we give a formal definition of this model. The following definition closely follows [76].

Definition 2.2.1. A deterministic communication protocol $\mathscr{P}$ over domain $X \times Y$ and range Z is a binary tree where each internal node v is labeled either by the function $A_v : X \to \{0,1\}$ or by the function $B_v : Y \to \{0,1\}$. Each leaf of this binary tree is labeled with an element $z \in Z$.

On input $(x,y) \in X \times Y$, the value of the protocol is the label of the leaf reached by starting from the root. For each internal node v of the binary tree labeled with A_v, move to the left child of v if $A_v(x) = 0$, otherwise move to the right child of v if $A_v(x) = 1$. Likewise, for each internal node v of the binary tree labeled with B_v, move to the left child of v if $B_v(y) = 0$, otherwise move to the right child of v if $B_v(y) = 1$. On input (x,y), the cost of the protocol is the length of the path taken starting from the root to the corresponding leaf. The cost of the protocol $\mathscr{P}$ is the height of the binary tree. The deterministic communication complexity of a function f is the minimum cost over all protocols $\mathscr{P}$ that compute f correctly.

Let us consider an example to illustrate this formal definition. Consider the following Boolean function f on $X \times Y$, where $X = \{x_0, x_1, x_2, x_3\}$ and $Y = \{y_0, y_1, y_2, y_3\}$.

f	y_0	y_1	y_2	y_3
x_0	0	0	0	1
x_1	0	0	0	1
x_2	0	1	1	1
x_3	0	0	0	0

Table 2.2.1: The function f computed by the protocol given in Figure 2.2.1.

The function f can be computed by the protocol given in Figure 2.2.1. For example on input (x_1, y_3), Alice sends the first message to Bob. This message is $A_1(x_1) = 0$. Next, Bob sends the bit $B_2(y_3) = 1$ to Alice and they both conclude that $f(x_1, y_3) = 1$. The cost of the protocol on input (x_1, y_3) is 2. The cost of the protocol is 3.

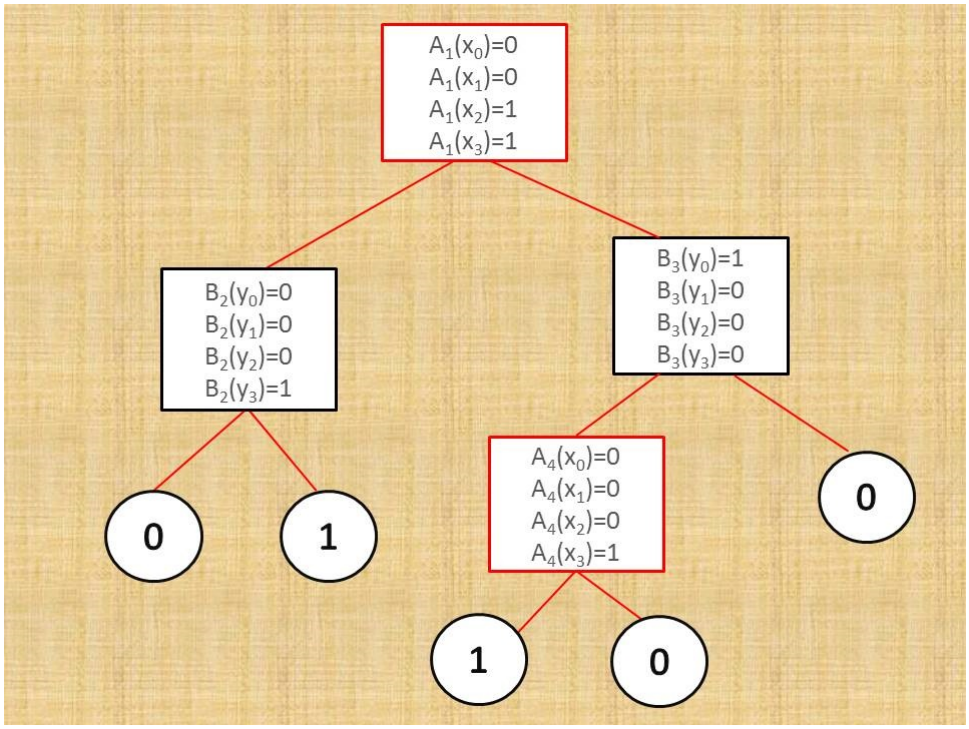


Figure 2.2.1: Protocol tree $\mathscr{P}_f$.

In this thesis, we are mainly interested in randomized protocols which output the correct answer with high probability. There are two variants of such randomized protocols: the private coin model and the public coin model. In the private coin model, Alice's randomness is not known to Bob

and vice versa. The coin flips are private, i.e. unknown to the other party. In the public coin model, they have access to the same random string. Another way of looking at this is that the coin flips are public, so that Alice and Bob get the same random string. One would agree that the private coin model is a more realistic model. Newman [88] showed that for any function f with T different inputs, if there is a protocol that requires c bits of communication in the public coin model, there is a corresponding protocol which requires $c + \log\log T + O(1)$ bits in the private coin model. For the case where the inputs are drawn from $\{0,1\}^n$, the communication complexity of a function f in the public coin model is only away from the communication complexity of f in the private coin model by an additive term of $O(\log n)$. For excellent introductions to communication complexity, we refer the reader to the textbooks by Kushilevitz and Nisan [76] or by Hromkovic [60]. For more advanced topics on different lower bound techniques developed for communication complexity, we refer to [80, 83].

The one-way communication complexity model is important for the study of streaming algorithms for the purpose of proving space lower bounds. In this model, there is a single message from Alice to Bob and Bob has to output the answer based on Alice's message. One-way communication complexity was first introduced by Yao [107] and this subtopic of communication complexity was taken up in greater consideration by several other authors (see e.g. [3, 33, 70, 75, 89, 90]). Given any randomized protocol $\mathscr{P}$, we say $\mathscr{P}$ computes a function f with error ϵ, if for every $(x, y) \in X \times Y$, we have

$$\mathbf{Pr}\left[\mathscr{P}(x, y, r) \neq f(x, y)\right] \leq \epsilon$$

where the probability is over r, the common random string that is generated by the public coin. We denote the randomized one-way communication

complexity of f in the public coin model by $R_{\epsilon}^{A\to B}(f)$ which is the cost of the best protocol that computes f with error at most ϵ. Usually, we will take $\epsilon = \frac{1}{3}$ and in this case, we will just omit ϵ. Note that if we start with a protocol which computes f with error $\frac{1}{3}$, we can always reduce the error to any ϵ by repeating the protocol $O(\log(1/\epsilon))$ times and taking the majority. The error analysis is a simple application of Chernoff's inequality.

We define two functions, INDEX and DISJ whose communication complexity is well studied in the literature.

Definition 2.2.2. For the INDEX function, Alice is given $x \in \{0,1\}^n$ and Bob is given an index $i \in [n]$. The goal is for Bob to output x_i with high probability.

Definition 2.2.3. For the DISJ function, both Alice and Bob are given $x, y \in \{0,1\}^n$ respectively. $\text{DISJ}_n(x,y)$ is a Boolean function which is defined to be 0 if and only if there exists $i \in [n]$ such that $x_i = y_i = 1$. We can also view this as follows: Alice and Bob each hold a subset of $\{1, \cdots, n\}$ (x and y respectively). $\text{DISJ}_n(x,y) = 1$ if and only if $x \bigcap y = \emptyset$. If we drop the subscript from DISJ_n, then for the purpose of this thesis, we will be referring to the DISJ function on n bits.

It is well-known that $R^{A\to B}(\text{INDEX}) = \Omega(n)$ [3, 75, 87]. For a simpler and self contained proof using error correcting codes, the reader is referred to [63]. On the other hand, if Bob is allowed to communicate with Alice as well, INDEX can be solved with $\log n + 1$ bits of communication. The hardness of the INDEX function depends on the one-way model. Using the lower bound on the one-way communication complexity of INDEX, it is easy to see that $R^{A\to B}(\text{DISJ}) = \Omega(n)$ as well. Given an instance (x,i) of INDEX, where $x \in \{0,1\}^n$ and $i \in [n]$, Bob forms a n-bit string y which is zero on all positions except the i-th position where it is one. They run the one-way

DISJ protocol on inputs (x, y). If the output is disjoint, Bob concludes that $x_i = 0$. Otherwise, he concludes that $x_i = 1$.

The DISJ function is the generic co-NP complete problem in communication complexity [8]. Even if multiple rounds of communication are allowed between Alice and Bob and they are allowed to use randomization, DISJ still needs $\Omega(n)$ communication [66, 94].

2.3 Frequency Moments

In this section, we state the main results of the seminal work of Alon, Matias and Szegedy [6]. Given a stream $\sigma = \langle a_1, \cdots, a_n \rangle$, we define f_i to be the frequency of item i, i.e. $f_i := |\{j \mid a_j = i\}|$ for each $1 \leq i \leq m$. For each $k \geq 0$, we define the k-th frequency moment as

$$F_k := \sum_{i=1}^{m} f_i^k.$$

We denote F_∞ as the frequency of the most frequent item, i.e.

$$F_\infty = \max_{1 \leq i \leq m} f_i.$$

F_0 is the number of distinct elements in a data stream. This quantity has many application in areas such as query optimization, IP routing and data mining (see the references cited in [67] for details). F_1 is the length of the stream, which is equal to n. This can be computed exactly using $O(\log n)$ space. The quantity F_2 is useful for computing certain statistical properties of the data such as the Gini coefficient of variance [53].

Since one of the focuses of this thesis is the exact computation of F_0 in different data stream models like the annotation and interactive models

that will be defined in later chapters, for the sake of completeness, we show a reduction from DISJ to F_0 to illustrate that the exact computation of F_0 requires linear space. This illustrates how the rich theory of communication complexity lower bounds is useful for showing lower bounds for streaming algorithms.

Given a stream of length $n \leq 2m$, suppose there is a streaming algorithm $\mathscr{A}$ which uses s bits of space to compute the exact value of F_0. We show a communication protocol that solves DISJ on m bits using s bits of communication, where Alice holds $x \in \{0,1\}^m$ and Bob holds $y \in \{0,1\}^m$ such that $wt(x) = wt(y) = k$ for some $k = \Theta(m)$. Alice treats her input as a subset of $[m]$ whose characteristic vector is x and runs the streaming algorithm $\mathscr{A}$ on this input. In particular, Alice treats her input as a stream of length k and updates the memory content of $\mathscr{A}$ accordingly. She communicates the content of the memory of $\mathscr{A}$ to Bob. Likewise, Bob treats his input as a subset of $[m]$ whose characteristic vector is y and continue updating the memory of $\mathscr{A}$. If the value of $F_0 = 2k$, he outputs that $\text{DISJ}(x,y) = 1$ and if $F_0 \leq 2k-1$, he outputs that $\text{DISJ}(x,y) = 0$. Indeed, this solves the DISJ function using s bits of communication. It has to be the case that $s = \Omega(m)$. On the other hand, it is easy to see that one can compute the exact value of F_0 using m bits of space. Initially, the algorithm maintains a length m Boolean vector v initialized to the all zero vector. Upon seeing an element $j \in [m]$ on the stream, if $v_j = 0$, it is updated to 1. Otherwise if $v_j = 1$, do not update it. The weight of v is the exact value of F_0.

For any nonnegative integer $k \neq 1$, given a stream of length $n \leq 2m$, any randomized algorithm that computes F_k exactly requires $\Omega(m)$ space. This shows that the exact computation of $F_k\,(k \neq 1)$ is hard under ran-

domization. The next natural thing to do is to approximate the frequency moments and see if this can be done in sublinear space. We require that the streaming algorithm $\mathscr{A}$ outputs an estimate $\widehat{F}_k$ of F_k such that

$$\mathbf{Pr}\left[\left|\widehat{F}_k - F_k\right| \leq \epsilon F_k\right] \geq \frac{2}{3}.$$

We call this a $(1 \pm \epsilon)$-approximation of F_k. When ϵ is a constant (independent of m and n), this is called a constant approximation of F_k. For any nonnegative integer k, we say it is easy to approximate F_k if there is a streaming algorithm which gives a constant approximation of F_k using polylog(m, n) amount of space. Otherwise, we say it is hard to approximate F_k.

Estimating F_0 in the data stream model is well studied, beginning with the work of Flajolet and Martin [40]. They gave a $O(\log m)$ constant approximation algorithm for F_0, but their algorithm requires access to a perfectly random hash function. It is not known how to construct such functions with limited space. This was then followed by a long line of research which had improvements to both the lower and upper bounds [6, 13–15, 17, 25, 32, 35, 39, 47, 48, 61, 106]. Finally in 2010, Kane, Nelson and Woodruff [67] gave an algorithm that computes a $(1 \pm \epsilon)$-approximation of F_0 using $O(\epsilon^{-2} + \log m)$ space. Due to the lower bounds in [6, 61, 106], their algorithm is optimal as well.

Alon, Matias and Szegedy [6] gave an algorithm that computes a $(1 \pm \epsilon)$-approximation of F_2 using $O\left(\frac{1}{\epsilon^2}(\log m + \log n)\right)$ space. They also showed that for any $k \geq 6$, any randomized streaming algorithm which gives a constant approximation of F_k requires $\Omega\left(m^{1-\frac{5}{k}}\right)$ bits of space. This lower bound was later improved to $\Omega\left(m^{1-\frac{2}{k}}\right)$ for the space complexity of any streaming algorithm which approximates F_k $(k \geq 3)$ up to a constant fac-

tor [12, 18]. It is hence hard to approximate F_k for $k \geq 3$.

We now mention the series of work done to obtain a tight upper bound for the constant approximation of F_k for any constant $k \geq 3$. In the seminal work [6], the authors were the first to give an algorithm with space complexity $O\left(m^{1-\frac{1}{k}}(\log n + \log m)\right)$. This was then improved to $\widetilde{O}\left(m^{1-\frac{1}{k-1}}\right)$ by Coppersmith and Kumar [26] and independently by Ganguly [45] using a different technique. Ganguly [46] later improved this upper bound to $\widetilde{O}\left(m^{1-\frac{2}{k+1}}\right)$. In 2005, Indyk and Woodruff [62] made a breakthrough and gave a $\widetilde{O}\left(m^{1-\frac{2}{k}}\right)$ upper bound which is optimal up to a factor of polylog(m, n). Bhuvanagiri, Ganguly, Kesh, and Saha [16] gave an simpler algorithm following the ideas from the work of Indyk and Woodruff [62], improving the high constants and polylogaritmic factors present in [62].

2.4 Other Problems in the Streaming Model

Other than the frequency moments, there are many other problems studied in the streaming model. We first introduce the INDEX problem in the streaming context.

Definition 2.4.1. For the INDEX problem in the streaming setting, the input stream is $a_1, \cdots, a_n$ followed by an index $i \in [n]$, where each $a_i \in \{0, 1\}$. The goal is to output a_i with probability at least $2/3$. For the GENERALIZED INDEX problem in the streaming setting, a_i is no longer binary but is drawn from a universe of size m, i.e. $a_i \in [m]$.

Another important area is the study of streaming algorithms for graph problems. For many important graph properties, it is known that it is impossible to determine if the given graph has a certain property using only a single pass over the stream and $o(m)$ space [37], where m is the

number of vertices of the graph. In view of this, many extensions of the streaming model have been introduced. One is to allow multiple passes over the input [58] and another is to consider a new model, which is called the semi-streaming model [38], where the algorithm is allowed to use $O(m \cdot \text{polylog}(m))$ bits of space. Zhang [109] has an excellent survey on streaming algorithms for graph problems.

Other problems commonly studied in the data stream model include matrix approximation problems like low rank approximation, deciding the rank of a matrix etc. [24]. Problems related to the sortedness of a data stream are also well studied [4, 54, 81, 100].

Chapter 3

Constant Round Interactions in Data Streams and Merlin-Arthur Classes

We have seen in Chapter 2 that many interesting problems like frequency moments F_k for $k > 2$ do not admit an efficient data streaming algorithm. In this chapter, we will consider a more powerful model for streaming. A third party is introduced who processes the stream and provides the answer together with a proof of correctness. We view the third party as the helper/prover who convinces the client/verifier of the correct answer. We will call the streaming model without the helper which was introduced in Chapter 2 the standard streaming model.

In this chapter, we will formally define the model where we introduce a helper to process the stream and give some protocols in this model. Like almost all previously known lower bounds on data streams, we will see how the Merlin-Arthur communication complexity model can be used to give further insight on the prover-verifier streaming model.

3.1 The Annotation Model

In this section we define the model of streaming computations with a helper/prover.

In the annotation model we consider two parties, the prover, and the verifier who wish to compute a function $f(\sigma)$. Both parties are able to access the data stream one element at a time, consecutively, and synchronously, i.e., no party can look into the future with respect to the other one.

The prover is a Turing machine that has unlimited workspace, and processes each symbol in some time $T(m, n)$ that will vary from problem to problem. Ideally we want $T(m, n)$ to be polylog(m, n) as well, but this would imply immediately that the problem at hand can be solved in quasilinear time which could be too restrictive for some problems like computing the rank of a matrix.

After the stream has ended, the prover sends a single message to the verifier claiming some particular value for $f(\sigma)$ and the verifier now has to verify this claim. The message that the prover sends to the verifier is viewed as a stream and the verifier need not store this message. He can do some computations with the message on the fly. The prover is said to have annotated the stream. This model was first introduced by Chakrabarti, Cormode, McGregor and Thaler [21] in 2009 and has been investigated further in [20, 28, 102].

We define a valid protocol that verifies the correctness of some function $f(\sigma)$ in the annotation model. Our definition closely follows [21].

Definition 3.1.1. (**Annotation Model**)
Before seeing the stream σ, both the prover $\mathcal{P}$ and verifier $\mathcal{V}$ agree on a protocol to compute $f(\sigma)$. This protocol should fix all the variables that

are to be used (e.g. type of codes, size of finite fields etc.), but should not use randomness to fix these variables.

After the stream ends, $\mathcal{P}$ sends $\mathcal{V}$ a single message. The message from $\mathcal{P}$ to $\mathcal{V}$ need not be stored but can be treated and processed as a stream. We denote the output of $\mathcal{V}$ on input σ, given $\mathcal{V}$'s private randomness $\mathcal{R}$, by $out(\mathcal{V}, \mathcal{P}, \mathcal{R}, \sigma)$. $\mathcal{V}$ can output $\perp$ if $\mathcal{V}$ is not convinced that $\mathcal{P}$'s claim is valid.

We say $\mathcal{P}$ is a valid prover or an honest prover if for all streams σ,

$$\mathbf{Pr}_{\mathcal{R}}\ [out(\mathcal{V}, \mathcal{P}, \mathcal{R}, \sigma) = f(\sigma)] \geq 1 - \delta_c.$$

We say $\mathcal{V}$ is a valid verifier for f if there is at least one valid prover $\mathcal{P}$, and for all provers $\mathcal{P}'$ and all streams σ,

$$\mathbf{Pr}_{\mathcal{R}}\ [out(\mathcal{V}, \mathcal{P}', \mathcal{R}, \sigma) \notin \{f(\sigma), \perp\}] \leq \delta_s.$$

δ_c is known as the completeness error, the probability that the honest prover will fail even if he follows the protocol. If $\delta_c = 0$, we say the protocol has perfect completeness. δ_s is called the soundness error, that is no prover strategy will cause the verifier to output a value outside of $\{f(\sigma), \perp\}$ with probability larger than δ_s. In this thesis, we take $\delta_c = \delta_s = \frac{1}{3}$ from this point onwards. By standard boosting techniques, these probabilities can be made arbitrary close to 1 [7].

The main complexity measure of the protocol is the space requirement of the verifier and the length of the message from the prover to the verifier. We make the following definition which takes into account these complexities.

Definition 3.1.2. We say there is a (h, v) protocol that computes f in the annotation model if there is a valid verifier $\mathcal{V}$ for f such that:

1. $\mathcal{V}$ has only access to $O(v)$ bits of working memory. (v is called the verification cost.)

2. There is a valid prover $\mathcal{P}$ for $\mathcal{V}$ such that the length of the single message from $\mathcal{P}$ to $\mathcal{V}$ is $O(h)$ bits. (h is called the help cost.)

Given a protocol in the annotation model, we define its cost to be $h + v$.

3.1.1 Basic Annotation Protocols

In this section, we give two basic annotation protocols. The first is a $(m \log n, \log m + \log n)$ protocol in the annotation model for the exact computation of F_k and the second is a $(\sqrt{n} \log m, \sqrt{n}(\log n + \log m))$ annotation protocol for the GENERALIZED INDEX problem. Both of these protocols are based on simple fingerprinting techniques which we describe next.

We would like to have a fingerprint of a set such that we can check equality with the same set even when the set is presented in a different order. Given a multiset presented as a stream $\sigma = \langle a_1, \cdots, a_n \rangle$, where each $a_i \in \{1, \cdots, m\}$, we compute the multiset fingerprint of σ as follows: $\widetilde{FP}_q(r, \sigma) = \prod_{i=1}^{n} (r - a_i) \pmod{q}$, where q is a prime chosen as described below before the start of the streaming process and r is chosen uniformly at random from $\mathbf{F}_q$. If n is not known in advance, an upper bound on n will suffice.

Lemma 3.1.3. *Let $q \geq \max\{n/\delta, m\}$ be a prime for some given $0 < \delta < 1$ (we call $1 - \delta$ the reliability of the fingerprint) and choose r uniformly at random from $\mathbf{F}_q$. Given a stream σ of length n where each symbol is drawn from a universe of size m, the multiset fingerprint $\widetilde{FP}_q(r, \sigma)$ can be computed using $O(\log m - \log \delta + \log n)$ bits of memory in a streaming fashion with $O(1)$ update time[1]. Let ς be a stream whose length is at most*

[1]The update time is measured in the unit cost RAM model, see Section 2.1.

n where each symbol is drawn from a universe of size m. If $\varsigma \not\simeq \sigma$ (the two streams ς and σ are not equal as multisets), then the collision probability

$$\mathbf{Pr}_{r \in_R \mathbf{F}_q} \left[\widetilde{FP}_q(r, \sigma) = \widetilde{FP}_q(r, \varsigma) \right] \leq \delta.$$

If $\varsigma \neq \sigma$, then $\widetilde{FP}_q(X, \sigma) - \widetilde{FP}_q(X, \varsigma)$ is an nonzero polynomial in X of degree at most n. The proof of the collision probability in Lemma 3.1.3 is a simple application of the Schwartz-Zippel lemma which can be found in Appendix A.1 of this thesis.

The prover can sort the stream and announce the frequencies of all the items after the stream has ended. The verifier can check that the correct frequencies of all the items were announced with high probability using the multiset fingerprint in Lemma 3.1.3. This gives a $(m \log n, \log m + \log n)$ protocol in the annotation model for the exact computation of F_k. If the help is not present, i.e. $h = 0$, then the verification cost is $m \log n$.

For the GENERALIZED INDEX function, we need the fingerprint to be variant under permutations. Given a stream $\sigma = \langle a_1, \cdots, a_n \rangle$ where each $a_i \in [m]$, we define the vector fingerprint

$$FP_q(r, \sigma) := \sum_{i=1}^{n} a_i \, r^{i-1} \pmod{q}. \tag{3.1}$$

for some prime q.

Lemma 3.1.4. *Let $q \geq \max\{n/\delta, m\}$ be a prime for some given $0 < \delta < 1$ (we call $1 - \delta$ the reliability of the fingerprint) and choose r uniformly at random from $\mathbf{F}_q$. Given a stream σ of length n where each symbol is drawn from a universe of size m, the vector fingerprint $FP_q(r, \sigma)$ can be computed using $O(\log m - \log \delta + \log n)$ bits of memory in a streaming fashion with*

$O(1)$ *update time*[2]. *Let* ς *be a stream whose length is at most* n *where each symbol is drawn from a universe of size* m. *If* $\varsigma \neq \sigma$, *then the collision probability*

$$\mathbf{Pr}_{r \in_R \mathbf{F}_q} \left[FP_q(r, \sigma) = FP_q(r, \varsigma) \right] \leq \delta.$$

As in the case for Lemma 3.1.3, the bound on the collision probability follows from the Schwartz-Zippel lemma.

For the GENERALIZED INDEX problem, the verifier partitions the stream into $\sqrt{n}$ blocks $B_1, \cdots, B_{\sqrt{n}}$ where $B_i := \left(a_{(i-1)\sqrt{n}+1}, \cdots, a_{i\sqrt{n}}\right)$. The verifier computes the vector fingerprint with reliability $2/3$ of each of the $\sqrt{n}$ blocks in a streaming fashion, using $O(\sqrt{n}(\log n + \log m))$ bits of memory. Given the index $i \in [n]$, the prover will send the vector B_k where $k := \left\lceil \frac{i}{\sqrt{n}} \right\rceil$ from which the verifier can obtain a_i. The soundness error of this protocol is $1/3$. This gives an $(\sqrt{n} \log m, \sqrt{n}(\log n + \log m))$ annotation protocol for the GENERALIZED INDEX problem. We note that for the GENERALIZED INDEX problem, any streaming algorithm in the standard model requires $\Omega(n \log m)$ space. To see this, consider the INDEX function on $n \log m$ bits. The simpler task of determining the bit of any index $i \in [n \log m]$ requires $\Omega(n \log m)$ space in the standard model [3, 75, 87]. Hence we can obtain cheaper protocols in the annotation model using simple fingerprinting techniques.

3.2 Frequency Moments Revisited in the Annotation Model

In this section, we study the exact computation of the frequency moments in the annotation model. In Section 2.3, we showed that the exact

[2] In the more general turnstile model, the update time is $O(\log n)$.

computation of F_k requires $\Omega(m)$ space in the standard streaming model for $k \neq 1$. Almost all non-trivial protocols in the annotation model can be viewed as modifying the Merlin-Arthur communication protocol of Aaronson and Wigderson [2] for the inner product function, which is based on "arithmetization". This was first observed by Chakrabarti, Cormode, McGregor and Thaler [21] who used the idea to devise many interesting protocols in the annotation model. This line of approach to devise protocols in the annotation model was used by several authors later [20, 28, 56].

3.2.1 Protocols for Frequency Moments

We begin by showing a protocol for computing the exact value of F_2 in the annotation model. This protocol gives a tradeoff between the help cost and the verification cost. The ideas for the protocol for the exact computation of F_2 given in Theorem 3.2.1 are the basic building blocks of many other protocols in the annotation model.

Theorem 3.2.1. *Let $h, v \in \mathbf{Z}^+$ such that $hv \geq m$. There is a $(h(\log n + \log m), v(\log n + \log m))$ protocol that computes the exact value of F_2 in the annotation model.*

Proof. Choose the smallest prime $p > \max\{n^2, 6h, v\}$. By Bertrand's postulate, such a prime can be represented by $O(\log n + \log m)$ bits. We work over the finite field $\mathbf{F}_p$. Consider any injective map $\phi : [m] \to [h] \times [v]$. Define the function $f : [h] \times [v] \to [n]$ such that for any $(x, y) \in [h] \times [v]$, if there exists a $z \in [m]$ such that $\phi(z) = (x, y)$, then $f(x, y) = f_z$. Otherwise, define $f(x, y) = 0$.

We consider the polynomial $\tilde{f} : \mathbf{F}_p^2 \to \mathbf{F}_p$ such that $\tilde{f}(x, y) = f(x, y)$ for all $(x, y) \in [h] \times [v]$. We say $\tilde{f}$ is a low degree extension of f over the field

$\mathbf{F}_p$. $\tilde{f}$ is obtained by interpolation, i.e.

$$\tilde{f}(X,Y) := \sum_{i=1}^{h}\sum_{j=1}^{v} f(i,j)\,\delta_i(X)\,\delta_j(Y) \in \mathbf{F}_p[X,Y]$$

where

$$\delta_i(X) := \frac{\prod\limits_{\substack{k=1\\k\neq i}}^{h}(X-k)}{\prod\limits_{\substack{k=1\\k\neq i}}^{h}(i-k)} \in \mathbf{F}_p[X] \text{ of degree at most } h-1,$$

$$\delta_j(Y) := \frac{\prod\limits_{\substack{k=1\\k\neq j}}^{v}(Y-k)}{\prod\limits_{\substack{k=1\\k\neq j}}^{v}(j-k)} \in \mathbf{F}_p[Y] \text{ of degree at most } v-1.$$

It is easy to see that

$$F_2 = \sum_{X=1}^{h}\sum_{Y=1}^{v}\left(\tilde{f}(X,Y)\right)^2.$$

Before observing the stream, $\mathcal{V}$ will choose $r \in \mathbf{F}_p$ uniformly at random. As $\mathcal{V}$ observes the stream, he will compute $\tilde{f}(r,y)$ for each $1 \leq y \leq v$. This can be computed in a streaming fashion due to the following observation: for any $1 \leq y^* \leq v$, we have

$$\begin{aligned}\tilde{f}(r,y^*) &= \sum_{i=1}^{h}\sum_{j=1}^{v} f(i,j)\,\delta_i(r)\,\delta_j(y^*)\\ &= \sum_{i=1}^{h} f(i,y^*)\,\delta_i(r)\end{aligned}$$

Initially $\tilde{f}(r,y)$ is zero for all $1 \leq y \leq v$. As $\mathcal{V}$ observes a symbol a on the stream, he computes $\phi(a) = (h^*, v^*)$ and updates $\tilde{f}(r,v^*) \leftarrow \tilde{f}(r,v^*) + \delta_{h^*}(r)$. Now it is clear that $\mathcal{V}$ can compute the values of $\tilde{f}(r,y)$ for each

$1 \leq y \leq v$ in a streaming fashion using $O(v(\log n + \log m))$ space. After the end of the stream, $\mathcal{V}$ will compute $\alpha := \sum_{y=1}^{v} \left(\tilde{f}(r, y)\right)^2$.

After the stream has ended, the prover should send the polynomial

$$s(X) := \sum_{y=1}^{v} \left(\tilde{f}(X, y)\right)^2 \in \mathbf{F}_p[X] \text{ of degree at most } 2(h-1).$$

The prover will define the polynomial $s(X)$ by communicating $\{(i, s(i)) : 0 \leq i \leq 2h-2\}$ using communication $O(h(\log n + \log m))$ bits. The verifier will output $F_2 = \sum_{X=1}^{h} s(X)$ if $s(r) = \alpha$. Note that F_2 can be computed in a streaming fashion given the representation of the polynomial $s(X)$. It is easy to see that $s(X) = \sum_{i=0}^{2h-2} s(i)\, \hat{\delta}_i(X)$ with

$$\hat{\delta}_i(X) = \prod_{\substack{k=0 \\ k \neq i}}^{2h-2} (X-k)/(i-k).$$

As a result, $\mathcal{V}$ can compute the value of $s(r)$ in a streaming fashion without having to store the polynomial $s(X)$ explicitly.

It is clear that if the prover is honest, the verifier will always accept. On the other hand, if the prover is dishonest, the verifier is fooled only if the polynomial $\tilde{s}(X)$ he receives does not represent $s(X)$ but $\tilde{s}(r) = s(r)$. By the Schwartz-Zippel Lemma, the probability that this happens is at most $(2h-2)/|\mathbf{F}_p| < 1/3$. □

By choosing $h = v = \sqrt{m}$, we get an protocol for F_2 in the annotation model with cost $O(\sqrt{m}(\log n + \log m))$. It is easy to see that for any constant $k \geq 2$, there is an annotation protocol with cost $O(\sqrt{m}(\log n + \log m))$ which computes F_k exactly. Now let us consider the problem of computing F_0 exactly in the annotation model. Using the notations and a

similar line of approach as in Theorem 3.2.1, it easy to see that

$$F_0 = \sum_{X=1}^{h} \sum_{Y=1}^{v} g \circ \tilde{f}(X, Y). \tag{3.2}$$

where $g : \mathbf{N} \to \{0, 1\}$ is a function which satisfies the following conditions: $g(0) = 0$ and $g(x) = 1$ for $1 \leq x \leq n$. Since g depends on $n+1$ points, the degree of the polynomial $\tilde{g}$, obtained via interpolation, is at most n, where $\tilde{g}$ agree with g on $\{0, 1, \cdots, n\}$. If we were to use the approach in Theorem 3.2.1 directly, this will cause the degree of the polynomial which is to be communicated to be $n(h-1)$. The approach of Chakrabarti, Cormode, McGregor and Thaler [21] was to reduce the degree of the polynomial g by removing all the heavy hitters[3] from the stream. As a result, they obtained a $\left(n^{2/3} \log^{4/3} n,\ n^{2/3} \log^{4/3} n\right)$ protocol for computing F_0 exactly in the annotation model where $m = \Theta(n)$. For the case when $m = \Theta(n)$, we can also obtain an protocol with the same cost using the ideas from Theorem 3.2.1 with a minor modification. The verifier divides the stream into $n^{1/3} \log^{2/3} n$ intervals each of size $n^{2/3}/\log^{2/3} n$. $\mathcal{V}$ stores all the elements in each interval and remove all duplicates of that interval. $\mathcal{V}$ needs $O(n^{2/3} \log^{1/3} n)$ space to do this. Now for the derived stream $\tilde{\sigma}$, $F_\infty \leq n^{1/3} \log^{2/3} n$. For $v = O(n^{2/3} \log^{1/3} n)$ and $h = O(n^{1/3}/\log^{1/3} n)$, we can use the approach of Theorem 3.2.1 to verify the sum in (3.2), where the degree of g now is at most $n^{1/3} \log^{2/3} n$. The verification cost will be $O(n^{2/3} \log^{1/3} n \cdot \log n)$. The degree of the polynomial communicated will be at most $n^{1/3} \log^{2/3} n \cdot (h-1)$ which gives a help cost of $O(n^{2/3} \log^{4/3} n)$.

Theorem 3.2.2. *Assume that $m = \Theta(n)$. In the annotation model, there is a (h, v) protocol which computes F_0 exactly with $h = v = n^{2/3} \log^{4/3} n$.*

[3] Those items whose frequencies exceed a fraction of n.

Now, let us consider the case for the exact computation of F_∞ in the annotation model. Note that the verifier can use the GENERALIZED INDEX annotation protocol on $(f_1, \cdots, f_m, i)$ to output f_i for any $i \in [m]$. In particular, the verifier can obtain $k := f_{i^*}$ in the annotation model with cost $O(\sqrt{m}(\log n + \log m))$ where $i^* \in \arg\max_i f_i$ is provided by the prover. The verifier can hence be convinced that with high probability that $F_\infty \geq k$. Using the notations in Theorem 3.2.1, the verifier needs to check that

$$0 = \sum_{X=1}^{h} \sum_{Y=1}^{v} g \circ \tilde{f}(X, Y). \tag{3.3}$$

where $g : \mathbf{F} \to \mathbf{F}$ is an interpolating polynomial such that $g(x) = 0$ for $x = 0, \cdots, k$ and $g(x) = 1$ for $x = k+1, \cdots, n$. The approach used in [21] was to reduce the degree of the polynomial g by removing all the items in the stream whose frequencies exceed a certain threshold. The ϕ-heavy hitters of a stream are all the items $i \in [m]$ such that $f_i > \phi n$ for some $\phi \in [0, 1]$.

Lemma 3.2.3. *[21]*

Let $h, v \in \mathbf{Z}^+$ such that $hv \geq 2m - 1$. There is a $(h(\log n + \log m) + \frac{1}{\phi} \log^2 m, v(\log n + \log m))$ protocol in the annotation model that identifies all the ϕ-heavy hitters of the data stream.

Consider the case where $m = \Theta(n)$. By choosing $\phi = \frac{\log^{2/3} n}{n^{2/3}}$, $h = O(n^{1/3}/\log^{1/3} n)$ and $v = O(n^{2/3} \log^{1/3} n)$, we obtain a $(n^{2/3} \log^{4/3} n,$ $n^{2/3} \log^{4/3} n)$ protocol in the annotation model which can identify all the items in the stream whose frequencies exceed $n^{1/3} \log^{2/3} n$. W.L.O.G., assume that $F_\infty \leq n^{1/3} \log^{2/3} n$. Both the verifier and the prover will check the sum (3.3) in the annotation model in a similar fashion as Theorem 3.2.1, where now the degree of g is low. In particular, g is the interpolat-

ing polynomial such that $g(x) = 0$ for $x = 0, \cdots, k$ and $g(x) = 1$ for $x = k+1, \cdots, n^{1/3}\log^{2/3} n$.

Theorem 3.2.4. *Assume that $m = \Theta(n)$. In the annotation model, there is a (h, v) protocol which computes F_∞ exactly with $h = v = n^{2/3}\log^{4/3} n$.*

3.3 Merlin-Arthur Communication Models

In Section 2.2, we introduced the basic model of communication complexity and showed that it is an important tool used to derive lower bounds on the space needed by a streaming algorithm in the standard model. Similarly, as we will see, the different Merlin-Arthur communication models can be used to derive lower bounds for the prover-verifier data stream model.

As before, Alice and Bob are given $x \in X$ and $y \in Y$ respectively and they need to compute the function $f : X \times Y \to \{0, 1\}$. Merlin-Arthur communication complexity was first introduced by Babai, Frankl and Simon [8] and was studied in greater detail by Klauck [71, 72]. In this model, there is an unreliable "super-player" called Merlin who knows the entire input (x, y) and can help Alice and Bob (who together constitute of Arthur) by interacting with them. The patterns in which Merlin and Arthur interact give rise to distinct communication models. The different communication models will be used to study the power and limitations of different modes of data streaming algorithms with a prover.

We first begin with the standard MA model which was first introduced in [8] before we look at other related models which are inspired by data streaming algorithms.

Definition 3.3.1. (**MA Communication**)

In a Merlin-Arthur (MA) protocol, Merlin sends a help message $h(x, y)$

to Bob. After this, Alice and Bob communicate using public key randomness (the proof cannot depend on the randomness). In other words, after receiving the proof, Alice and Bob use a randomized protocol with a public random string $\mathcal{R}$ to output a bit $out(x, y, \mathcal{R}, h(x, y))$. We say that the MA protocol $\mathscr{P}$ computes f with error δ if there exist a function $h : X \times Y \to \{0,1\}^*$ such that

1. If $f(x, y) = 1$, then $\mathbf{Pr}_{\mathcal{R}}\left[out(x, y, \mathcal{R}, h(x, y)) = 0\right] \leq \delta_c$. (3.4)

2. If $f(x, y) = 0$, then $\forall\, h' \in \{0,1\}^*$, we have

$$\mathbf{Pr}_{\mathcal{R}}\left[out\left(x, y, \mathcal{R}, h'\right) = 1\right] \leq \delta_s. \tag{3.5}$$

We define the error $\delta := \max\{\delta_c, \delta_s\}$. For any MA protocol $\mathscr{P}$ which computes f with error δ, we define the help cost

$$hcost_\delta(\mathscr{P}) := \max_{x,y} |h(x, y)|$$

and verification cost $vcost_\delta(\mathscr{P})$ to be the maximum number of bits communicated by Alice and Bob over all x, y and the public random string $\mathcal{R}$. We define

$$MA_\delta(f) := \min\big\{hcost_\delta(\mathscr{P}) + vcost_\delta(\mathscr{P}) \mid \mathscr{P} \text{ is a MA protocol which computes } f \text{ with error } \delta\big\},$$

and $MA(f) := MA_{1/3}(f)$ following standard notations.

It was shown by Klauck [71] that $MA(\textsc{Disj}) = \Omega(\sqrt{n})$ which is almost tight [2]. Furthermore, Aaronson and Widgerson [2] showed by a counting argument that there exist functions f such that $MA(f) = \Omega(n)$, although

no explicit construction of such a function is known.

In the MA model of communication complexity, Merlin first sends a help message $h(x, y)$ to Bob, who then uses public randomness to communicate with each other. If Alice and Bob generate the public random string first before Merlin sends them a message each, this is called the Arthur-Merlin communication model. In this model, the help message that Merlin sends depends on the randomness as well. Arthur-Merlin communication complexity was first introduced by Babai, Frankl and Simon [8] which is the natural communication complexity analogue of the Turing machine version of AM [10].

Definition 3.3.2. (**AM Communication**)
In an Arthur-Merlin (AM) protocol, Alice tosses a random coin r, which is seen by both Bob and Merlin. Merlin sends a help message $h(r, x, y)$ to both Alice and Bob[4]. After this, Alice and Bob communicate using a deterministic communication protocol to arrive at a Boolean output. Note that Alice and Bob are not allowed to use any fresh randomness after receiving the help message from Merlin.

Similar to the definition of the correctness of a MA protocol (See equations (3.4),(3.5)), we say that an AM protocol computes f correctly if for every 1-input, there is a prover strategy such that with probability at least 2/3, the protocol accepts, and for every 0-input, no matter what Merlin's strategy is, the protocol rejects with probability at least 2/3. The cost of such a protocol is the sum of the maximum length of the message sent by Merlin and the maximum length of the communication between Alice and Bob over all x, y, r. The complexity $AM(f)$ is the cost of an optimal AM protocol that computes f with error 1/3.

[4]In the model where Merlin can only send the help message to Bob, the cost only differs by a constant factor as Alice and Bob are allowed to communicate.

Note that Merlin can determine the exact transcript of Alice and Bob after sending the help message. As such, Alice and Bob only need to communicate one bit each after receiving Merlin's message. This is because Merlin can include the whole conversation between Alice and Bob in his message, and they only need to check if the transcript is valid. We note that there are no general methods known till date to establish non-trivial AM lower bounds.

Figure 3.3.1 illustrates a MA communication protocol while Figure 3.3.2 illustrates an AM communication protocol.

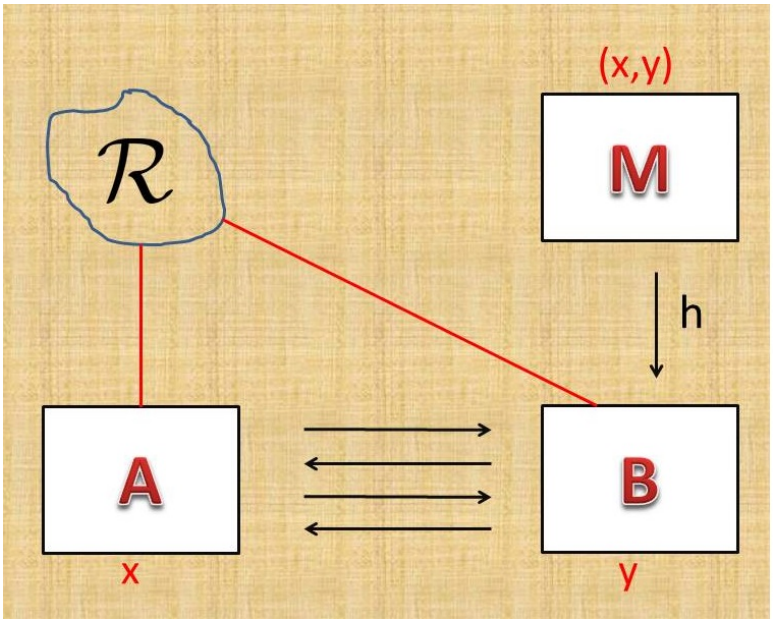


Figure 3.3.1: MA communication protocol. The help message h depends on x and y only. After Bob(B) receives h from Merlin(M), Alice(A) and Bob communicate using a randomized protocol in the public coin model.

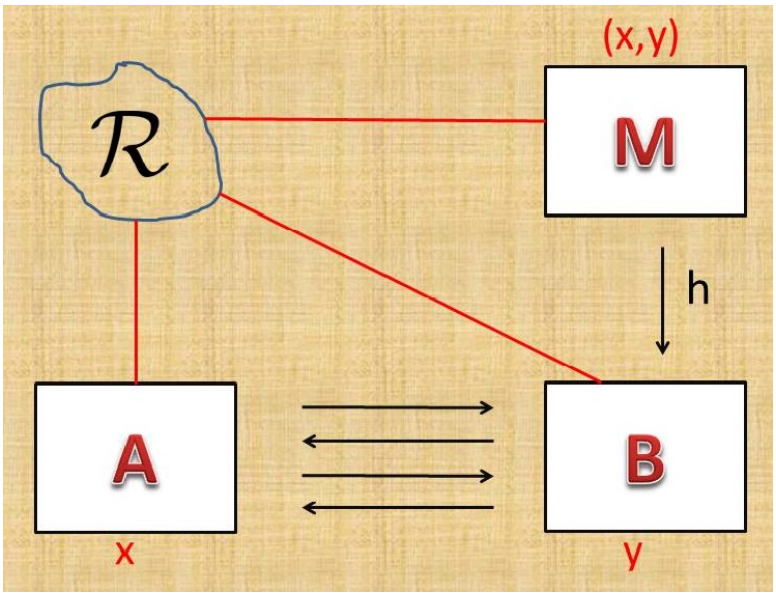


Figure 3.3.2: AM communication protocol. In this model, a random string $r \in \mathcal{R}$ is drawn and known to all the three players. Merlin(M) then sends Bob(B) the help message h, which depends on x, y and r. After this, Alice(A) and Bob communicate using a deterministic protocol.

3.3.1 Online Merlin-Arthur Communication Models

For applications to the streaming model with a prover, we use a weaker model where Alice can only communicate with Bob and Merlin can only send his help message to Bob. This restricted version of the MA communication model is called the online MA model of communication complexity. We allow Merlin and Bob to send k messages to each other, with Merlin always sending the last message. Allowing Merlin and Bob to interact is analogous to the case for streaming algorithms that allow the prover and verifier to interact after the stream has ended. There are two main variants of the interaction between Merlin and Bob. The first is what is commonly known as the public-coin proof system (a.k.a. Arthur-Merlin proof systems). The second is the general interactive proof system as defined in [50].

Definition 3.3.3. (**Online MA Communication with k messages**)
In an online Merlin-Arthur protocol with k messages (OMA^k protocol), Alice receives input x, Bob receives input y and Merlin sees both inputs x and y. Merlin and Bob exchange k messages in total, where Merlin always sends the last message to Bob. Bob always replies to Merlin with a random challenge, i.e. his response to Merlin does not depend on his input and is just a sequence of random strings $r_1, \cdots, r_{\lfloor \frac{k}{2} \rfloor}$, which is revealed round by round by a public coin and thus all three parties know the outcome of the coin flips. After receiving the last message from Merlin, $r_{\lfloor \frac{k}{2} \rfloor + 1}$ is generated from the public coin and Alice sends Bob a message $m_A(x, r_1, \cdots, r_{\lfloor \frac{k}{2} \rfloor + 1})$.

After receiving all the messages from Merlin and Alice, Bob has to accept or reject. Note that $r_{\lfloor \frac{k}{2} \rfloor + 1}$ is only generated after Merlin sends his last message and hence this randomness can be viewed as hidden from Merlin and only shared between Alice and Bob. Let $h_1, \cdots, h_{\lceil k/2 \rceil}$ be the messages sent by Merlin and $r_1, \cdots, r_{\lfloor k/2 \rfloor + 1}$ be the random coin tosses generated by the public coin. The sequence of the conversation between Merlin and Bob depends on whether an even or odd number of messages were exchanged between them. For the case where $k = 2j - 1$ is odd, the sequence of messages is $(h_1, r_1, h_2, r_2, \cdots, h_{j-1}, r_{j-1}, h_j, r_j, m_A(x, r_1, \cdots, r_j))$. Note that in this case, h_1 depends on x, y and for $2 \leq i \leq j$, h_i depends on $(x, y, h_1, \cdots, h_{i-1}, r_1, \cdots, r_{i-1})$. Similarly for the case when $k = 2j$ is even, the sequence of messages is
$(r_1, h_1, r_2, h_2, \cdots, h_{j-1}, r_j, h_j, r_{j+1}, m_A(x, r_1, \cdots, r_{j+1}))$, where h_1 depends on x, y, r_1 and for $2 \leq i \leq j$, h_i depends on $(x, y, h_1, \cdots, h_{i-1}, r_1, \cdots, r_i)$.

Let $\mathcal{R} = \left(r_1, \cdots, r_{\lfloor k/2 \rfloor + 1}\right)$ and Bob's output for the OMA^k protocol be $out_B\left(y, m_A(x, \mathcal{R}), \mathcal{R}, h_1, \cdots, h_{\lceil k/2 \rceil}\right)$. We say that the OMA^k protocol computes f if there exist functions $h_1, \cdots, h_{\lceil k/2 \rceil}$ such that

1. If $f(x,y)=1,$ then

$$\mathbf{Pr}_{\mathcal{R}}\left[out_B\left(y, m_A(x,\mathcal{R}), \mathcal{R}, h_1, \cdots, h_{\lceil k/2\rceil}\right)=1\right] \geq 1-\epsilon_c. \quad (3.6)$$

2. If $f(x,y)=0,$ then $\forall\ \left(h'_1, \cdots, h'_{\lceil k/2\rceil}\right) \in (\{0,1\}^*)^{\lceil k/2\rceil}$, we have

$$\mathbf{Pr}_{\mathcal{R}}\left[out_B\left(y, m_A(x,\mathcal{R}), \mathcal{R}, h'_1, \cdots, h'_{\lceil k/2\rceil}\right)=1\right] \leq \epsilon_s. \quad (3.7)$$

The error of the protocol is the maximum of ϵ_c and ϵ_s. The help cost of the protocol is $|h_1|+\cdots\left|h_{\lceil k/2\rceil}\right|+|r_1|+\cdots\left|r_{\lfloor k/2\rfloor}\right|$ for the worst case instance and the verification cost is defined as $\left|m_A(x, r_1, \cdots, r_{\lfloor \frac{k}{2}\rfloor+1})\right| + \left|r_{\lfloor k/2\rfloor+1}\right|$ for the worst case instance as well. The cost of the protocol is the sum of the help cost and verification cost. The complexity $OMA^k(f)$ is the cost of an optimal OMA^k protocol that computes f with error at most $1/3$.

We note that for a k-round online MA protocol, the messages from Bob to Merlin are random challenges, i.e. these messages neither depend on Bob's input nor Alice's message. The next model is more general as it allows Bob to send messages to Merlin based on his input and the message sent by Alice.

Definition 3.3.4. (**Online IP Communication with k messages**)
In an online interactive proof protocol with k messages (OIP^k protocol), Alice and Bob toss some coins first which are hidden from Merlin and Alice sends Bob a message based on this shared hidden randomness $\mathcal{R}_{A,B}$. Then Bob and Merlin interact as in an OMA^k protocol but the messages from Bob to Merlin depend on his input, the shared randomness between Alice and Bob, and the history of previous messages from Merlin, but not on Alice's message to Bob.

Similar to the definition of the correctness of an OMA^k protocol (See (3.6), (3.7)), we say an OIP^k protocol computes f correctly if for every 1-input, there is a prover strategy such that with probability at least $2/3$, the protocol accepts, and for every 0-input, no matter what Merlin's strategy is, the protocol rejects with probability at least $2/3$. For any OIP^k protocol $\mathscr{P}$ which computes f with error $1/3$, we define the verification cost to be the worst case number of bits communicated by Alice to Bob plus the length of $\mathcal{R}_{A,B}$ and the help cost $hcost(\mathscr{P})$ to be the maximum number of bits communicated by Merlin and Bob over all x, y and the random string $\mathcal{R}_{A,B}$. The cost of the protocol is the sum of the help cost and verification cost. The complexity $OIP^k(f)$ is the cost of an optimal OIP^k protocol that computes f with error at most $1/3$.

If Bob's messages to Merlin also depend on Alice's message, then we denote the protocol by $\widetilde{OIP^k}$. As before, we denote $\widetilde{OIP^k}(f)$ to be the cost of the optimal $\widetilde{OIP^k}$ which computes f correctly.

We illustrate the differences between the OMA^2, OIP^2 and $\widetilde{OIP^2}$ protocols in Figures 3.3.3, 3.3.4 and 3.3.5 respectively. For more details of OMA^k, OIP^k and $\widetilde{OIP^k}$ protocols, the interested reader is referred to [22].

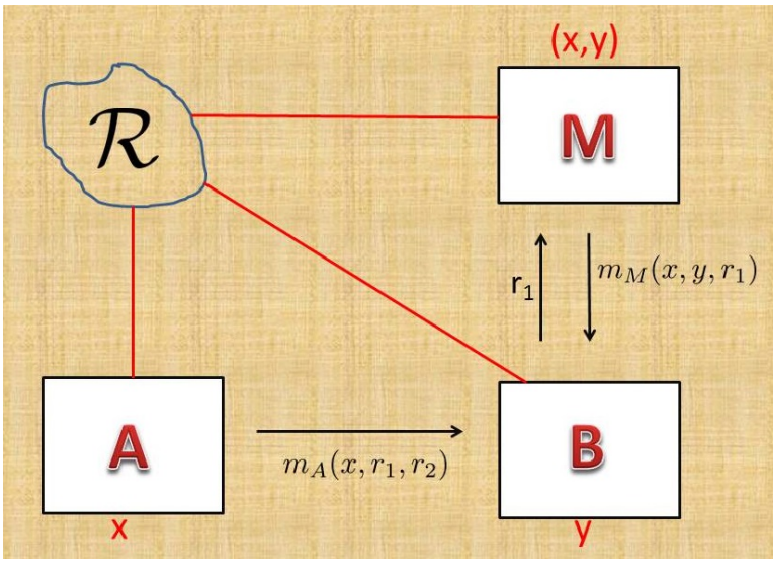


Figure 3.3.3: OMA^2 communication protocol. In this model, a random string $r_1 \in \mathcal{R}$ is drawn and known to all the three players. Merlin(M) then sends Bob(B) the help message m_M, which depends on x, y and r_1. After this, the second random string $r_2 \in \mathcal{R}$ is drawn and Alice(A) sends Bob a message m_A, which depends on x, r_1, r_2. Bob is then required to produce an output.

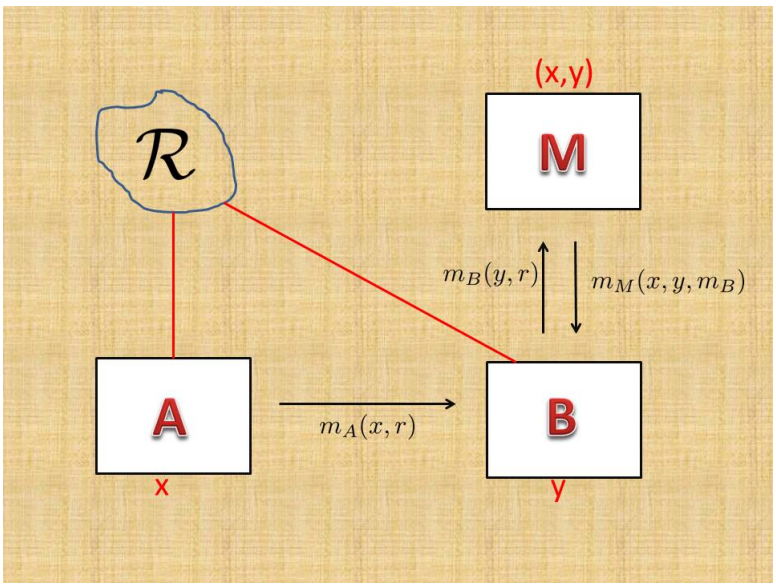


Figure 3.3.4: OIP^2 communication protocol. In this model, a random string $r \in \mathcal{R}$ is drawn together by Alice(A) and Bob(B) only. Note that Merlin(M) does not have access to r. After this, Bob sends Merlin a message m_B, which depends on y, r. Merlin replies Bob with a message m_M, which depends on x, y, m_B. Finally, Alice sends Bob a message m_A, which depends on x, r. Bob is then required to produce an output.

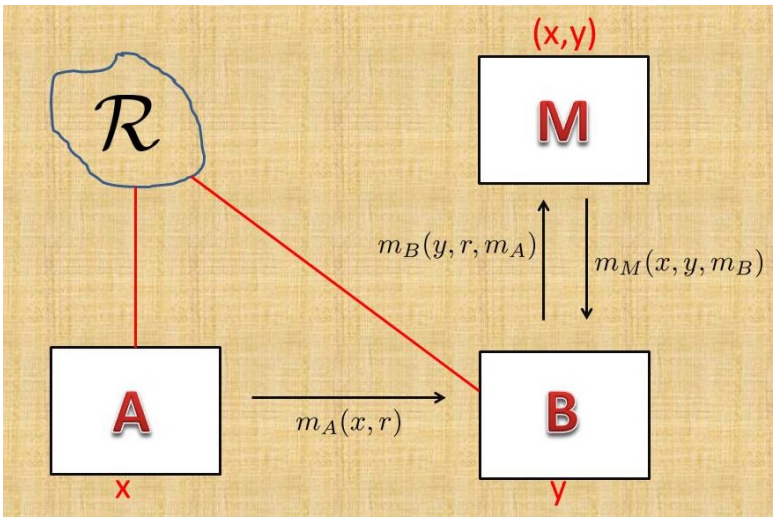


Figure 3.3.5: $\widetilde{OIP^2}$ communication protocol. In this model, a random string $r \in \mathcal{R}$ is drawn together by Alice(A) and Bob(B) only. Note that Merlin(M) does not have access to r. After this, Alice sends Bob a message m_A, which depends on x, r. Bob sends Merlin a message m_B, which depends on y, r and m_A. Finally, Merlin replies Bob with a message m_M, which depends on x, y, m_B. Bob is then required to produce an output.

An OMA^1 protocol is the same as a MA communication protocol except that the communication from Alice to Bob is one-way. In a similar manner, we can define the online version of Arthur-Merlin protocol.

Definition 3.3.5. (**Online AM Communication**)

In an online Arthur-Merlin protocol (OAM protocol), Alice receives input x, Bob receives input y, and Merlin sees both inputs x and y. Alice tosses a random coin r (which is seen by Bob and Merlin), and sends a single message $m_A(x, r)$ to Bob. Merlin then sends Bob a message $h(x, y, r)$ and Bob has to accept or reject without using any more randomness. We define the correctness of the OAM protocol in a similar fashion as Definition 3.3.1, i.e. for all inputs x, y such that $f(x, y) = 1$, there is a prover strategy such that Bob will accept with probability at least $2/3$, and for every inputs x, y such that $f(x, y) = 0$, no matter what Merlin's strategy is, Bob will reject with probability at least $2/3$. The cost of an OAM protocol is

$|r| + |m_A(x,r)| + |h(x,y,r)|$ on the worst case instance. The complexity $OAM(f)$ is the cost of an optimal OAM protocol that computes f with error $1/3$.

Figure 3.3.6 illustrates an OAM communication protocol.

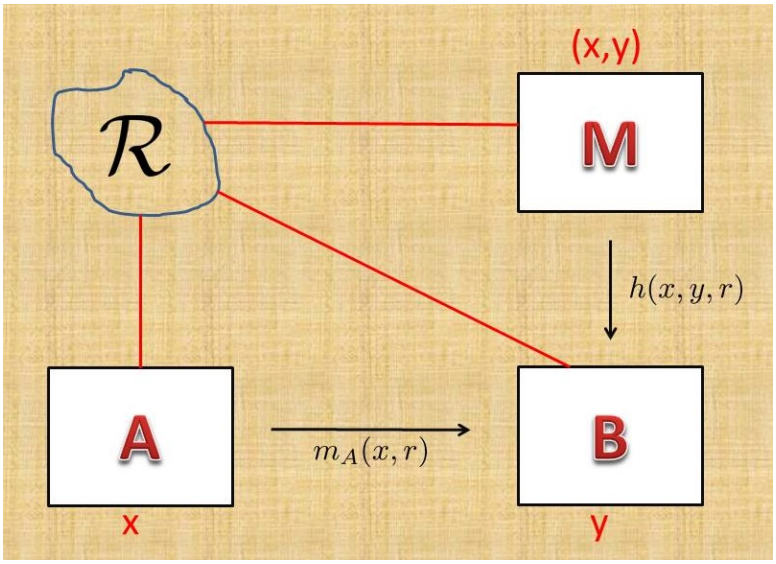


Figure 3.3.6: OAM communication protocol. In this model, a random string $r \in \mathcal{R}$ is drawn and known to all the three players. Merlin(M) then sends Bob(B) the help message h, which depends on x, y and r. Alice(A) sends Bob a message m_A, which depends on x, r. Note that in this set-up, Merlin knows the content of Alice's message to Bob. Bob is then required to produce an output after receiving these two messages.

3.3.2 Communication Complexity Classes

Babai, Frankl and Simon [8] were the first authors to define communication complexity classes, which are natural analogs of the complexity classes defined in the Turing machine world. We define the randomized one-way communication complexity class as

$$R_{cc}^{A\to B} := \{f : R^{A\to B}(f) = \text{polylog}(n)\}.$$

In a similar spirit, we can define communication complexity classes for the different Merlin-Arthur models which we have defined till now. The classes that we will refer to later in this thesis include

$$\begin{aligned} AM_{cc} &= \{f : AM(f) = \text{polylog}(n)\}, \\ OMA^k_{cc} &= \{f : OMA^k(f) = \text{polylog}(n)\} \quad \text{and} \\ OIP^k_{cc} &= \{f : OIP^k(f) = \text{polylog}(n)\}. \end{aligned}$$

3.3.3 Lower Bounds for the Annotation Model

We have shown earlier in this chapter annotation protocols for F_0, F_2, F_∞ and Generalized Index. A natural question to ask is whether the protocols that we gave for these problems can be further improved. The natural model of communication complexity which corresponds to the annotation streaming model is the OMA^1 model, where there is a single message from Merlin to Bob. The following lower bound is known for this communication model.

Theorem 3.3.6. *[1, 21]*

Suppose there is an OMA^1 protocol which solves f with error at most $1/3$, where the help cost is h and the verification cost is v. Then $(h+1)v = \Omega\left(R^{A\to B}(f)\right)$.

Proof. The conclusion clearly holds when $h = 0$. For $h \geq 1$, note that the error probability can be reduced to $2^{-\Omega(h)}$ where the communication from Merlin is still h bits and the message from Alice is hv bits now. We denote this new protocol with error $2^{-\Omega(h)}$ by $\mathscr{P}$.

Consider the following $R^{A\to B}$ protocol which solves f with error $1/3$. Given Alice's message in $\mathscr{P}$, Bob loops over all 2^h possible messages of Merlin and accept if and only if there is a message that would cause him

to accept. It is easy to see that if $f(x,y) = 1$, there exist a message $m_M \in \{0,1\}^h$ such that Bob accepts with probability at least $1 - 2^{-\Omega(h)}$. If $f(x,y) = 0$, by the union bound, the probability that there exist a string from $\{0,1\}^h$ that causes Bob to accept is at most $1/3$. Hence it follows that $hv = \Omega\left(R^{A\to B}(f)\right)$. □

Corollary 3.3.7. *$OMA^1(f) = \Omega\left(\sqrt{R^{A\to B}(f)}\right)$. Since $OMA^1(f) \leq R^{A\to B}(f)$, we have $OIP^1_{cc} = R^{A\to B}_{cc}$.*

This idea of reducing the error probability first and then trying all possible proofs was first introduced by Aaronson [1]. For the INDEX function, we have $OMA^1(\text{INDEX}) = \Omega(\sqrt{n})$. The annotation protocol that we gave in Section 3.1.1 for GENERALIZED INDEX shows that $OMA^1(\text{INDEX}) = O(\sqrt{n}\log n)$. This protocol can be modified to obtain a $O(\sqrt{n})$ upper bound on the OMA^1 complexity of the INDEX function [1]. For completeness sake, we briefly describe this protocol. As before, Alice will partition her input $x \in \{0,1\}^n$ into $\sqrt{n}$ blocks $B_1, \cdots, B_{\sqrt{n}}$ where $B_i := \left(x_{(i-1)\sqrt{n}+1}, \cdots, x_{i\sqrt{n}}\right)$. Instead of computing the vector fingerprint of each of these $\sqrt{n}$ blocks, she applies a linear error correcting code to each of her blocks. In more details[5], let $G \in \mathcal{M}_{c\sqrt{n},\sqrt{n}}(\mathbf{F}_2)$ be the generator matrix of a binary Justesen code $\mathcal{C} = \left[c\sqrt{n}, \sqrt{n}, \frac{c\sqrt{n}}{\alpha}\right]_2$ for some constants $c > 1,\ 0 < \alpha < 1$. Alice picks $k = O(1)$ indices $\tau_1, \cdots, \tau_k \in \{1, \cdots, c\sqrt{n}\}$ uniformly at random and sends them to Bob. For each $1 \leq i \leq \sqrt{n}$, Alice computes $z_i := GB_i^T$ and sends $(z_i)_{\tau_1}, \cdots, (z_i)_{\tau_k}$ to Bob. Note that the length of Alice's message to Bob is $O(\sqrt{n})$. Given Bob's index $i \in [n]$, Merlin will send the vector B_k where $k := \left\lceil \frac{i}{\sqrt{n}} \right\rceil$ to Bob which will enable him to output x_i. Due to the error correcting properties of $\mathcal{C}$, the soundness

[5]The reader is referred to Appendix A.2 for the notations in coding theory that we will be using in this thesis.

error can be made to be $1/3$ by an appropriate choice of the constant k. As a result, we have $OMA^1(\textsc{Index}) = O(\sqrt{n})$.

This shows that the lower bound in Corollary 3.3.7 is tight up to constant factors. Since Justesen codes are linear and locally logspace constructible[6], it is easy to see that there exist an $(\sqrt{n}, \sqrt{n})$ streaming protocol for $\textsc{Index}$.

It is easy to see that the exact computations of F_k $(k \neq 1)$ is as hard as $\textsc{Disj}_m$. Since $OMA^1(\textsc{Disj}_m) = \Omega(\sqrt{m})$, for any constant $k \geq 2$, the annotation protocol given in Section 3.2.1 for F_k is tight up to polylogarithmic factors. For the case of the exact computation of F_0 and F_∞, there is a gap between the best known streaming protocol and the lower bound.

How about approximating F_k in the annotation model? Chakrabarti, Cormode, McGregor and Thaler [21] showed that any (h, v) streaming protocol which approximates F_k up to constant factors in the annotation model requires $hv = \Omega(m^{1-5/k})$. They obtained this lower bound by studying the number-in-hand (NIH) multi-party Merlin-Arthur communication complexity of a promise version of the $\textsc{Disj}$ function. We note that the same function was first introduced in [6] to study the space complexity of approximating F_k in the standard streaming model.

In the NIH communication model, instead of having just two players (Alice and Bob), there are t players $P_1, \cdots, P_t$ who wish to compute a function $f : \mathcal{X}_1 \times \cdots \mathcal{X}_t \to \{0, 1\}$ where P_i holds $x_i \in \mathcal{X}_i$. To compute the joint function $f(x_1, \cdots, x_t)$, they need to collaborate with each other. We are only interested in the amount of communication used to compute the joint function. As such, all the players are assumed to have unlimited computational power. There are three different variants of the NIH model, depending on the mode of communication. They are

[6] See Appendix A.2 for the definition of locally logspace constructible.

1. The blackboard model. Each player writes his message on a common blackboard, visible to all the other players.

2. The message-passing model. Player P_i sends a message to another player P_j.

3. The coordinator model. There is another player P_{t+1} who does not receive any inputs. In this context, P_{t+1} is called the coordinator. All the players can only communicate with the coordinator.

We can improve the lower bound given in [21] for approximating F_k in the annotation model. To do so, we need to consider the NIH multi-party one-way communication complexity model and the corresponding annotation model. Each of the players has access to his own private randomness. Of special interest to us will be the one-way communication model, which is a restricted version of the NIH message-passing model. In this model, P_i can only send a single message to P_{i+1} for $1 \leq i \leq t-1$. Note that the message from P_1 to P_2 is $m_1(x_1, r_1)$, where r_1 is the random string generated by P_1. For $2 \leq i \leq t-1$, P_i sends a message $m_i(x_i, m_{i-1}, r_i)$ to P_{i+1} where r_i is the random string generated by P_i. P_t will output the answer based on his input, his private randomness and m_{t-1}. Given a protocol which computes f correctly, we define its cost to be $|m_1| + \cdots + |m_{t-1}|$ on the worst case instance. We will denote the δ error one-way t-party communication complexity of f by $R_\delta^{P_1 \to \cdots \to P_t}(f)$, which is the cost of the optimal protocol which computes f correctly with error δ.

Consider the multi-party unique disjointness function $\textsc{Disj}_{m,t}$ where for $1 \leq i \leq t$, P_i holds the subset $A_i \in [m]$ with the promise that either one of the following is true:

1. $A_i \bigcap A_j = \emptyset$ for all $i \neq j$. (**YES Instance.**[7])

[7] DISJ evaluates to 1 on a YES instance and evaluates to 0 otherwise.

2. There exist a $x \in [m]$ such that $A_i \bigcap A_j = \{x\}$ for all $i \neq j$. (**NO Instance.**)

$\textsc{Disj}_{m,t}$ is a promise version of the $\textsc{Disj}$ function. It is well known that $R_{1/3}^{P_1 \to \cdots \to P_t}(\textsc{Disj}_{m,t}) = \Omega(m/t)$ [18]. This result also holds in the blackboard model [55].

We can also study the $\textsc{Disj}_{m,t}$ problem in the multi-party online Merlin-Arthur model. The setup is similar to Definition 3.3.3.

Definition 3.3.8. (**Online MA Communication with t parties**)
In an online Merlin-Arthur protocol with t parties (t-party OMA^1 protocol), we have t players $P_1, \cdots, P_t$ who wish to compute a function $f : \mathcal{X}_1 \times \cdots \mathcal{X}_t \to \{0,1\}$ where P_i holds $x_i \in \mathcal{X}_i$. Each of the players has access to their own private randomness. P_i can only send a single message m_i to P_{i+1} for $1 \leq i \leq t-1$. Merlin will send P_t a message m_M. Note that Merlin only knows the inputs of all the players but not the messages $m_1, \cdots, m_{t-1}$, as these messages depend on the players randomness which is not known to Merlin. Let $\mathcal{R} = (r_1, \cdots, r_t)$ be the random string used where r_i is the randomness generated by P_i. Let $out_{P_t}(x_t, m_M, m_{t-1}, r_t) \in \{0,1\}$ be the output of P_t.

We say that the t-party OMA^1 protocol computes f with error $\delta := \max\{\delta_c, \delta_s\}$ if there exist a function $m_M : \mathcal{X}_1 \times \cdots \times \mathcal{X}_t \to \{0,1\}^*$ such that

1. If $f(x_1, \cdots, x_t) = 1$, then

$$\mathbf{Pr}_{\mathcal{R}}\left[out_{P_t}(x_t, m_M, m_{t-1}, r_t) = 0\right] \leq \delta_c.$$

2. If $f(x_1, \cdots, x_t) = 0$, then $\forall\, m'_M \in \{0,1\}^*$, we have

$$\mathbf{Pr}_{\mathcal{R}}\left[out_{P_t}(x_t, m'_M, m_{t-1}, r_t) = 1\right] \leq \delta_s.$$

Given a t-party OMA^1 protocol, we define the help cost to be $|m_M|$ for the worst case instance and the verification cost to be $|m_1| + \cdots + |m_{t-1}|$ for the worst case instance as well.

A t-party OMA^1 communication protocol is a natural generalization of the OMA^1 model where there are only two players, Alice and Bob (See Definition 3.3.3).

Theorem 3.3.9. *Suppose there is a t-party OMA^1 protocol which computes $\textsc{Disj}_{m,t}$ with error $1/3$ where the help cost is h and the verification cost is v. Then*

$$(h+1)v = \Omega\left(R_{1/3}^{P_1 \to \cdots \to P_t}(\textsc{Disj}_{m,t})\right) = \Omega(m/t)$$

Proof. The conclusion clearly holds when $h = 0$ [18]. For $h \geq 1$, note that the error probability can be reduced to $2^{-\Omega(h)}$ where the help cost is still h bits and the verification cost being $O(hv)$ bits now. This can be done by getting P_1 to P_t to repeat the verification process $k = O(h)$ times in parallel, using fresh randomness each time they do so. P_t will receive k different messages $m_{t-1}^{(1)}, \cdots, m_{t-1}^{(k)}$ from P_{t-1}. For each message $m_{t-1}^{(j)}$ that P_t receives, he will compute the output of the protocol using the same message from Merlin and fresh randomness. P_t will accept if the majority of these outputs resulted in acceptance, otherwise he will reject. By standard Chernoff bounds, it is easy to see that the error probability can be reduced to $2^{-\Omega(h)}$ by such a procedure. We denote this new protocol with error $2^{-\Omega(h)}$, help cost h and verification cost hv by $\mathscr{P}$.

Consider the following $R^{P_1 \to \cdots \to P_t}$ protocol which solves f with error $1/3$. Given player P_{t-1}'s message in $\mathscr{P}$, P_t loops over all 2^h possible messages of Merlin and accepts if and only if there is a message that would cause him to accept. It is easy to see that if $\textsc{Disj}_{m,t}(A_1, \cdots, A_t) = 1$, there exist a message $m_M \in \{0,1\}^h$ such that P_t accepts with probability at least

$1-2^{-\Omega(h)}$. If $\textsc{Disj}_{m,t}(A_1, \cdots, A_t) = 0$, by the union bound, the probability that there exists a string from $\{0,1\}^h$ that causes P_t to accept is at most $1/3$. Hence it follows that $hv = \Omega\left(R_{1/3}^{P_1 \to \cdots \to P_t}(\textsc{Disj}_{m,t})\right)$. □

We now give a reduction from $\textsc{Disj}_{m,t}$ to approximating F_k [6]. Suppose there exists a protocol $\mathscr{P}$ in the annotation model which gives a $(1 \pm \epsilon)$-approximation of F_k for $0 < \epsilon < 1$ with help cost h and verification cost v. Let $t = ((1 + c\epsilon)m)^{1/k}$ where c is chosen such that $c > \frac{2}{1-\epsilon}$. Given an instance of $\textsc{Disj}_{m,t}$, $P_1, \cdots, P_t$ will simulate protocol $\mathscr{P}$ as follows. P_1 will execute $\mathscr{P}$ on the elements of A_1, passing the content of the memory of $\mathscr{P}$ to P_2, and so on and so forth. Finally, P_t will produce $out_{\mathscr{P}}$, the output of the annotation protocol $\mathscr{P}$ on the input stream $A_1 \cup \cdots \cup A_t$. If $out_{\mathscr{P}} \leq m(1+\epsilon)$, P_t will output 1, otherwise P_t will output 0.

Indeed, if $\textsc{Disj}_{m,t}(A_1, \cdots, A_t) = 1$, then for the stream $A_1 \cup \cdots \cup A_t$, $F_k \leq m$ and $out_{\mathscr{P}} \leq (1+\epsilon)m$ with probability at least $2/3$. On the other hand, if $\textsc{Disj}_{m,t}(A_1, \cdots, A_t) = 0$, then $F_k \geq t^k = (1 + c\epsilon)m$ and $out_{\mathscr{P}} \geq (1-\epsilon)(1+c\epsilon)m > (1-\epsilon)m$ with probability at least $2/3$. This gives us a t-party OMA^1 protocol which computes $\textsc{Disj}_{m,t}$ where the help cost is h and the verification cost is tv. As a result of Theorem 3.3.9, we have $hv = \Omega\left(\frac{m}{t^2}\right)$.

Corollary 3.3.10. *For $k \geq 3$, suppose there is a (h, v) annotation protocol which gives a $(1\pm\epsilon)$-approximation of F_k for some constant $\epsilon \in (0,1)$. Then $hv = \Omega(m^{1-2/k})$.*

We saw in Section 2.3 that it is easy to approximate F_0 and F_2 in the standard streaming model. We remind the reader that F_1 is simply the length of the stream. For $k \geq 3$, it is hard to approximate F_k in the standard streaming model [6, 12, 18].

For any nonnegative integer k, we say it is easy to approximate F_k in the annotation model if there exists a (h, v) annotation protocol which gives a constant approximation of F_k such that $h + v = \text{polylog}(m, n)$. Otherwise, we say it is hard to approximate F_k in the annotation model. Previously it was known that for $k \geq 6$, it is hard to approximate F_k in the annotation model [21]. Prior to this thesis, it was an open problem whether it is easy or hard to approximate F_k in the annotation model for $k = 3, 4, 5$.

For $k \geq 3$, Corollary 3.3.10 shows that it is hard to approximate F_k in the annotation model. For $k \geq 6$, our lower bound in Corollary 3.3.10 is only a polynomial improvement when compared to the lower bound presented in [21].

3.3.4 A Lower Bound for OMA^k

In this subsection, we generalize the lower bound for the OMA^1 communication model given in Subsection 3.3.3. We will show a lower bound for the online Merlin-Arthur communication complexity model with k messages. At this point, we will use a result by Babai and Moran [9] showing that a constant number of rounds of interaction between the prover and verifier can be reduced to one round (i.e., that the AM hierarchy collapses). We need the following, more detailed statement for online Merlin-Arthur protocols, which holds for all functions f.

Lemma 3.3.11. *For any function $f : X \times Y \to \{0, 1\}$, we have*

1. $OMA^{2k-1}(f) = \Omega(OAM(f))^{1/(k+1)}$.
2. $OMA^{2k}(f) = \Omega(OAM(f))^{1/(k+1)}$.

Proof. Assume we are given an online MA protocol with $2k - 1$ messages exchanged between Bob and Merlin. We will remove one round after an-

other until we get a 1-round protocol, i.e. a protocol in which Alice, Merlin and Bob share a public coin R, Alice sends one message to Bob, Merlin sends one message to Bob such that if $f(x,y) = 1$, then with probability at least $2/3$ (over r), there is a message from Merlin that makes Bob accept, while if $f(x,y) = 0$, then with probability at least $2/3$, no matter what Merlin sends, Bob will reject.

For a fixed input (x,y), let a denote the length of the message sent by Alice. Let $M_1, \cdots, M_k$ be the messages that Merlin sends to Bob, where M_1 is the first message sent etc. We denote the length of the message M_i by m_i, i.e. $|M_i| = m_i$ for $i = 1, \cdots, k$. Also, let $R_1, \cdots, R_k$ be the random strings generated by the public coin (R_i is generated after M_i has been sent) and denote the length of R_i by r_i, i.e. $|R_i| = r_i$ for $i = 1, \cdots, k$. The sequence of the messages is

$$(M_1, R_1, M_2, R_2, \cdots, M_{k-1}, R_{k-1}, M_k, R_k, M_A(x, r_1, \cdots, r_k)),$$

where M_A is Alice's message to Bob. Then $a + \sum_{i=1}^{k}(m_i + r_i) \leq c$, where $c := OMA^{2k-1}(f)$.

We first wish to reduce the error to $1/2^{2m_k}$. This can be done by performing the whole procedure $O(m_k)$ times in parallel, i.e. after Merlin sends the first message, Bob will send his message $O(m_k)$ times for different random strings, Merlin will send his second message several times (each time depending on the random strings chosen by Bob) etc. Note that parallel repetition is not an issue here. By standard Chernoff bounds, this decreases the error probability to $1/2^{2m_k}$. This increases the total communication to $O(m_k\, c)$. We now have an OMA^{2k-1} protocol which computes f correctly whose error is $1/2^{2m_k}$ and cost $O(m_k\, c)$.

Now consider the time after M_k has been sent. We change the protocol

such that Bob sends R_k before M_k. Since $|M_k| = m_k$, by the union bound, the probability that there is a message M_k for Merlin that makes Bob accept wrongly is at most 2^{-m_k}, even though Merlin knows Bob's last random string now. Now, we have a protocol whose message sequence is $\Big(M_1, R_1, \cdots, M_{k-2}, R_{k-2}, M_{k-1}, \tilde{R}, M_k, M_A \Big)$, where $\tilde{R} := (R_{k-1}, R_k)$. Note that the error of this new protocol is 2^{-m_k} and the cost is $O(m_k\, c)$.

Next we reduce the error to $1/2^{2m_{k-1}}$, again by parallel repetition. This will increase the communication to $O(m_{k-1} m_k\, c)$. Again, we can push Bob's last random choice one step ahead etc.

After k such steps, we end up with an OAM protocol which computes f correctly and has cost $O(m_1 m_2 \cdots m_k c) = O(c^{k+1})$.

For the case where $2k$ messages are exchanged between Merlin and Bob, note that the message sequence is $\big(R_1, M_1, R_2, M_2, \cdots, M_{k-1}, R_k, M_k, R_{k+1}, M_A(x, R_1, \cdots, R_{k+1}) \big)$. A similar argument can be used to push Bob's last random choice one step ahead at a time, obtaining an OAM protocol which computes f correctly and whose cost is $\big(OMA^{2k}(f)\big)^{k+1}$. □

We now proceed to give a lower bound for the online Arthur-Merlin model.

Lemma 3.3.12. $OAM(f) = \Omega(R^{A\to B}(f))$.

We omit the original proof of Lemma 3.3.12 which first appeared in [73]. Instead, we give a simplified proof which is less technical than the original proof in [73].

Proof. (**Simplified proof of Lemma 3.3.12.**)
Suppose we are given an OAM protocol which solves f correctly with error $1/3$ and whose cost is c. Let $m_A(x, r)$ be Alice's message to Bob given

the random string r. Note that $|m_A(x,r)| \leq c$. Let $out_B(y, m_A(x,r), h) \in \{0,1\}$ be Bob's output given the help message h from Merlin, where $|h| \leq c$.

The following is a one-way randomized communication protocol which computes f with the same error and cost as the OAM protocol. Upon receiving $m_A(x,r)$ from Alice, Bob will output $f(x,y) = 1$ if and only if there exist an $h \in \{0,1\}^c$ such that $out_B(y, m_A(x,r), h) = 1$. Indeed, if $f(x,y) = 1$, then with probability at least $2/3$ over r, there exist an $h \in \{0,1\}^c$ such that $out_B(y, m_A(x,r), h) = 1$. On the other hand, if $f(x,y) = 0$, then with probability at least $2/3$ over r, for all $h \in \{0,1\}^c$, we have $out_B(y, m_A(x,r), h) = 0$. Hence, $R^{A\to B}(f) \leq OAM(f)$. □

For AM protocols, we note that a similar approach cannot be used to remove Merlin. This is because for each help message from Merlin in an AM protocol, Alice and Bob need to communicate to verify if the help message is correct.

Theorem 3.3.13. $OMA^k(f) = \Omega\left(\left(R^{A\to B}(f)\right)^{\frac{1}{\lceil k/2\rceil+1}}\right)$.

Proof. Combine Lemmas 3.3.11 and 3.3.12. □

Corollary 3.3.14. $OMA^k_{cc} = R^{A\to B}_{cc}$ *for* $k = O(1)$.

Proof. Since $OMA^k(f) \leq R^{A\to B}(f)$, Corollary 3.3.14 is a simple consequence of Theorem 3.3.13. □

3.4 Merlin-Arthur and IP Streaming Model

Just as the OMA^1 model was used to analyze streaming protocols in the annotation model, developing lower bounds for the OMA^k model can be used to analyze streaming protocols where the prover and verifier are allowed to interact. In particular, they are allowed to exchange k messages

after the stream has ended, where the prover sends the last message in the interaction. Similar to the discussion in Definition 3.1.1, we can define a valid prover and a valid verifier.

Definition 3.4.1. We say there is a (h, v) streaming interactive protocol (SIP) with r messages ($r \geq 2$) that computes f, if there is a valid verifier $\mathcal{V}$ for f such that:

1. $\mathcal{V}$ has only access to $O(v)$ bits of working memory. (v is called the verification cost.)

2. There is a valid prover $\mathcal{P}$ for $\mathcal{V}$ such that $\mathcal{P}$ and $\mathcal{V}$ exchange r messages in total after the stream has ended with the prover sending the last message and the sum of the length of the messages is $O(h)$ bits. (h is called the help cost.)

Sometimes, instead of writing a SIP with r messages, we will use the term a SIP with r rounds. By such, we mean a SIP where at most $2r$ messages are exchanged between the prover and verifier with the prover sending the last message. Given any SIP, we define its complexity to be $O(h + v)$.

As before, there are two main variants of the interaction between the prover and the verifier. This is analogous to the public-coin proof system and the general interactive proof system which we defined for the online Merlin-Arthur communication complexity models.

Definition 3.4.2. The verifier has access to his own private randomness. Before seeing the stream, he generates all the randomness that he needs for the streaming protocol. In a Merlin-Arthur streaming model, the verifier's messages to the prover are just some of his random bits. To be more precise, suppose that the verifier has to send k messages to the prover according to

some given protocol. In this case, the verifier will generate random strings $r_1, \cdots, r_k$ and r_V from a source of random bits before observing the stream. For $1 \leq i \leq k$, the i-th message of the verifier to the prover is r_i and r_V is the verifier's private randomness that is not revealed to the prover.

In the IP streaming model, the messages from the verifier to the prover are some function of his private randomness, the stream and the messages received from the prover up to that point.

It is known that for the Turing machine model, the complexity classes[8] IP^k_{TM} and $\mathcal{AM}_{\text{TM}}$ are equivalent when k is a constant, i.e. $IP^k_{TM} = \mathcal{AM}_{\text{TM}}$ for $k = O(1)$ [9,51]. For standard communication complexity classes (where both Alice and Bob can communicate with each other) as defined in [8], Lokam [82] observed that the classes IP^k_{cc} and AM_{cc} behave the same way as the corresponding classes in the Turing machine model, i.e. $IP^k_{cc} = AM_{cc}$ for $k = O(1)$.

Surprisingly, in a recent work by Chakrabarti, Cormode, McGregor, Thaler and Venkatasubramanian [22], it was shown that there is an exponential separation between the Merlin-Arthur and IP streaming model. Chakrabarti et al. [22] showed that if the verifier's message to the prover depends on the stream and his private randomness, then INDEX can be solved with just 2 messages exchanged and the cost of the streaming protocol which does this is $O(\log n \log \log n)$. Before we provide the details of their construction, let us see why it is not possible to use the Goldwasser-Sipser transformation [51] to convert any IP streaming algorithm to a Merlin-Arthur streaming algorithm, while increasing the space complexity and communication only polynomially and increasing the number of rounds by only 2. The main tool used to prove that for Turing machines, Arthur-

[8] See Appendix A.3 for the definitions of IP^k_{TM} and $\mathcal{AM}_{\text{TM}}$.

Merlin protocols can be used to simulate protocols in the IP model with just a polynomial blowup in cost is the *Goldwasser-Sipser set lower bound protocol* (see Chapter 8.2 in [7]). Following the set lower bound protocol as described in [7], the verifier has to randomly pick a hash function h from a pairwise independent hash family and a value y randomly from the codomain of h. The prover replies with r such that $h(r) = y$ and the correct messages, which will lead the verifier to accept given his private randomness was r. But the verifier has no way to check this, as the value of r is given after the stream has ended. In summary, the Goldwasser-Sipser transformation does not work because of the online nature of streaming protocols.

A simple consequence of Theorem 3.3.13 is that INDEX cannot be solved by any streaming algorithm in the Merlin-Arthur model with polylog(n) cost, requiring only constant number of messages to be exchanged between the prover and verifier. For completeness sake, we provide the IP streaming protocol which solves INDEX with just 2 messages and cost $O(\log n \log\log n)$. A similar problem called the Retrieval problem was studied by Raz in [92] and the protocol presented here is a simple extension of Raz's idea to the streaming setting. We note that this was first observed by Chakrabarti et al. [22].

Given $x \in \{0,1\}^n$, define $f_x : [n] \to \{0,1\}$ such that $f_x(i) = x_i$ for any $i \in [n]$. We can also interpret each $f_x : \{0,1\}^{\log n} \to \{0,1\}$ by associating each $i \in [n]$ with its binary expansion. Let $d = \log n$ and choose the smallest prime $p \geq 3\log n + 1$. Let $\mathbf{F} := \mathbf{F}_p$ and $\widetilde{f_x} : \mathbf{F}^d \to \mathbf{F}$ be the low

degree extension of f_x over $\mathbf{F}$ which is given by

$$\begin{aligned}\widetilde{f}_x\left(Z_1,\cdots,Z_d\right) &:= \sum_{v\in\{0,1\}^d} f_x(v)\,\chi_v\left(Z_1,\cdots,Z_d\right) \quad \text{where}\\ \chi_v\left(Z_1,\cdots,Z_d\right) &:= \prod_{j=1}^{d}\left[(2v_j-1)\,Z_j+(1-v_j)\right].\end{aligned}$$

It is easy to see that $\widetilde{f}_x(Z_1,\cdots,Z_d)$ is a polynomial of total degree at most d, i.e.

$$\widetilde{f}_x(Z_1,\cdots,Z_d)=\sum_{\substack{\alpha_i\in\{0,1\}\\ 1\le i\le d}} c_{\alpha_1,\cdots,\alpha_d}\,Z_1^{\alpha_1}\cdots Z_d^{\alpha_d}.$$

where $c_{\alpha_1,\cdots,\alpha_d}\in\mathbf{F}$. The verifier chooses $r\in\mathbf{F}^d$ uniformly at random and computes $\widetilde{f}_x(r_1,\cdots,r_d)$ in a streaming fashion, where the update time is $O(\log n)$ per symbol seen on the stream. Upon observing the index $i^*\in\mathbf{F}^d$ on the stream[9], the verifier computes $b:=\mu(r-i^*)$ for some $\mu\in\mathbf{F}^*$ chosen uniformly at random. The verifier sends b to the prover and this defines the line $\mathcal{L}:=\{i^*+\lambda b\mid\lambda\in\mathbf{F}\}\subseteq\mathbf{F}^d$. The prover sends the polynomial $\widetilde{f}_x$ restricted on the line $\mathcal{L}$ to the verifier, i.e the verifier receives $g(\lambda):=\widetilde{f}_x(i_1^*+\lambda b_1,\cdots,i_d^*+\lambda b_d)\in\mathbf{F}[\lambda]$ whose degree is at most d. The verifier computes $g(0)$ and $g(\mu^{-1})$ and outputs $g(0)$ as the answer if $g(\mu^{-1})$ agrees with the fingerprint $\widetilde{f}_x(r_1,\cdots,r_d)$.

It is easy to see that this protocol has perfect completeness. For the case of the dishonest prover, if the polynomial $\tilde{g}$ sent is not identically equal to the polynomial g in the honest case, then by the Schwartz-Zippel lemma, $\mathbf{Pr}_{r\in_R\mathbf{F}^*}[\tilde{g}(r)=g(r)]\le d/(|\mathbf{F}|-1)\le 1/3$. The cost of the protocol is $O(d\log|\mathbf{F}|)=O(\log n\log\log n)$. It is important to note that the message of the verifier to the prover depends on the stream and his private randomness.

[9]The index $i^*\in[n]$ can be thought of as an element of $\mathbf{F}^d$ by associating it with its binary expansion, where $d=\log n$.

Theorem 3.4.3. *[22]*

The INDEX *streaming problem can be solved in the IP streaming model with* 2 *messages exchanged and cost* $O(\log n \log\log n)$. *In the Merlin-Arthur streaming model, there is no protocol which can solve* INDEX *with a constant number of messages and cost* $\text{polylog}(n)$.

Corollary 3.4.4. *For any constant* $k \geq 1$, $OIP^2_{cc} \nsubseteq OMA^k_{cc}$.

The IP streaming protocol for INDEX can be used to design constant message protocols for various query problems like Median and pattern matching queries [22]. One may wonder if we can obtain constant message protocols for more general streaming problems like frequency moments. To do so, we need to study the $\widetilde{OIP^k}$ communication complexity model. It is known that for any constant $k > 0$, there exist a constant $0 < \beta_k \leq 1$ such that $\widetilde{OIP^k}(f) = \Omega\left((AM(f))^{\beta_k}\right)$ [22]. Although proving a superlogarithmic lower bound on the AM complexity of DISJ is wide open, it is widely believed that it is unlikely that $AM(\text{DISJ}) = \text{polylog}(n)$ since DISJ is the generic co-NP complete problem in communication complexity [8]. Hence, constant message protocols for exact F_k ($k \neq 1$) with polylogarithmic complexity probably do not exist, but the current techniques in communication complexity (which fail to provide strong lower bounds on the AM communication complexity of Disjointness) are not sufficient to prove this.

3.5 Related Results

In this section, we give a brief summary of other related work in the annotation model and some recent results for online Merlin-Arthur communication complexity classes.

Graph problems on n nodes which is determined by m edges which appear on the stream was first studied in [21] in the annotation model. They studied graph problems like counting the exact number of triangles in a graph and to decide if a given graph is bipartite. By designing a generic protocol in the annotation model for the linear programming problem, Cormode, Mitzenmacher and Thaler [28] gave annotation protocols for graph problems like shortest $s-t$ path and the minimum weight bipartite perfect matching problem.

The cost of the annotation protocols for INDEX and F_k given in this thesis are sublinear in m, the size of the universe. One may also consider designing annotation protocols for sparse data sets, i.e. $n \ll m$. In such a case, ideally the cost of the annotation protocol should be sublinear in n instead of m. Chakrabarti, Cormode, Goyal and Thaler [20] were the first authors to give annotation protocols for INDEX, DISJ and F_k where the cost is sublinear in n. They considered the sparse version of INDEX and showed that for this problem, it is just as hard to solve in the annotation model as compared to the standard streaming model. More concretely, they considered the $\text{INDEX}_{\log n}$ function where Alice is given $x \in \{0,1\}^n$ whose Hamming weight $wt(x) = \log n$ and Bob is given an index $i \in [n]$. It was shown in [20] that $OMA^1(\text{INDEX}_{\log n}) = \Omega(\log n)$. Consider $\text{INDEX}_{\log n}$ as a streaming problem (See Definition 2.4.1). It is easy to see that $\text{INDEX}_{\log n}$ can be solved in the standard streaming model with space $O(\log n)$ via standard hashing techniques[10]. We briefly sketch this algorithm. Before observing the stream, the verifier chooses $h : [n] \to [3 \log n]$ from a family of pairwise independent hash functions. The space

[10]The authors in [20] showed that $R^{A \to B}(\text{INDEX}_{\log n}) = O(\log n \log\log n)$. This is because they used a hash function that has the property that it is injective on $\{i \mid x_i = 1\}$ with high probability. This is necessary for the sparse disjointness problem, which the authors also studied but not for $\text{INDEX}_{\log n}$.

needed to store this hash function is $O(\log n)$ (See Section 8.4 in [85] for a standard construction of such a hash function). Let $z \in \{0,1\}^{3\log n}$ be the indicator vector such that $z_i = 1$ if and only if there exists a $1 \leq j \leq n$ satisfying $x_j = 1$ and $h(j) = i$. It is clear that the verifier can update the vector z in a streaming fashion with $O(1)$ update time. Upon observing the index $i^* \in [n]$, the verifier outputs 1 if and only if $z_{h(i^*)} = 1$. It is easy to see that the error of this streaming algorithm is $1/3$. This was the first explicit example that was known in the literature which shows that certain streaming problems are just as hard in the annotation model as they are in the standard streaming model. Recently, Thaler showed that for the connectivity and bipartiteness streaming problems in the XOR update model, any protocol in the annotation model can only be cheaper by at most a polylogarithmic factor from the cost of any streaming algorithm in the standard model [102]. The results also holds for dense graphs, unlike the previous example for INDEX which only holds for the sparse instance.

For the Turing machine based complexity class IP^k_{TM} which is defined in Definition A.3.1, it is well known that this class collapses to the second level [9, 51], i.e. $IP^k_{TM} = IP^2_{TM}$ for any constant $k \geq 2$. A similar result holds for the corresponding communication complexity class which was first pointed out by Lokam [82]. For the online version of the communication complexity class, the situation is slightly different as OIP^k_{cc} collapses to the fourth level. In particular, it was shown in [22] that

$$OIP^1_{cc} \subsetneq OIP^2_{cc} \subsetneq OIP^3_{cc} \subsetneq OIP^4_{cc} = OIP^k_{cc} = AM_{cc} \tag{3.8}$$

for any constant $k \geq 4$.

Chapter 4

Interactive Streaming Model

In Chapter 3, we studied the annotation streaming model where we showed that for functions like INDEX and F_k, it is not possible to obtain protocols with polylog(m, n) complexity. For streaming interactive protocols (SIP) where a constant number of messages are exchanged between the prover and verifier, we saw that problems like INDEX have protocols with polylog(n) cost but it is conjectured that for harder problems like DISJ and F_k, this is not possible. The purpose of this chapter is to study SIP with polylog(m, n) rounds of interaction between the prover and verifier.

4.1 Generic Protocol for $\mathcal{NC}$

Let us begin by defining the complexity class $\mathcal{NC}$ (Nick's class). We refer the reader to [49] for a definition of Boolean circuits and the functions computed by them.

Definition 4.1.1. A family of circuits is LOGSPACE-uniform if the description of the n^{th} circuit can be generated by a Turing machine in $O(\log n)$ space.

Definition 4.1.2. The complexity class $\mathcal{NC}$ is the class of languages that can be recognized using LOGSPACE-uniform families of finite fan-in circuits with depth polylog(n) and size $n^{O(1)}$.

Goldwasser, Kalai and Rothblum [52] proposed a delegation interactive protocol for any language that can be computed by $\mathcal{NC}$ circuits. Their protocol is efficient in the sense that the communication complexity is polynomial in the depth of the circuit rather than its size and the running time of the verifier is linear in the input and polynomial in the depth. Cormode, Thaler and Yi [30] were the first authors to observe that the result of Goldwasser, Kalai and Rothblum [52] can be extended to the streaming setting, which will give efficient streaming protocols for any problem in $\mathcal{NC}$. Their protocol was presented formally in the streaming setting by Cormode, Mitzenmacher, and Thaler [29]. We will refer to this generic interactive protocol given below as the GKR protocol in this thesis.

Theorem 4.1.3. *[Theorem 3.1 from [29]]*
Let $f : \mathbf{F} \to \mathbf{F}$ be a function that can be computed by a family of $O(\log S(n))$-space uniform arithmetic circuits[1] *(over $\mathbf{F}$) of fan-in 2, size $S(n)$ and depth $d(n)$. Then in the streaming model with a prover, there is a protocol for f which requires $O(d(n) \log S(n))$ rounds such that the verifier needs $O(\log S(n) \log |\mathbf{F}|)$ bits of space and the total communication between the prover and the verifier is $O(d(n) \log S(n) \log |\mathbf{F}|)$.*

It is easy to see that for any fixed finite field, arithmetic circuits can simulate Boolean circuits with only a constant factor loss in both size and depth. The standard transformation of a Boolean circuit into an arithmetic circuit can be found in [43].

[1] See [49] for more details on arithmetic circuits.

It follows from Theorem 4.1.3 that if f is in $\mathcal{NC}$, there is a SIP with cost polylog(m,n) which computes f requiring polylog(m,n) rounds of interaction. The main tool used to construct the SIP is the sum check protocol of Lund, Fortnow, Karloff and Nisan [84].

Lemma 4.1.4. *[Sum Check Protocol of Lund et al. [84]]*
Let $g \in \mathbf{F}[X_1, \cdots X_q]$ whose total degree is d. Suppose the prover claims that

$$h = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_q \in \{0,1\}} g\,(x_1, \cdots x_q) \tag{4.1}$$

for some $h \in \mathbf{F}$. There is an interactive protocol with q rounds which accepts (4.1) with perfect completeness and the soundness error is at most $qd/|\mathbf{F}|$.
We assume that for any $(r_1, \cdots, r_q) \in \mathbf{F}^q$, $g(r_1, \cdots, r_q)$ can be evaluated efficiently. The total communication between the prover and verifier is $(q + \sum_{i=1}^{q} deg_i(g)) \log(|\mathbf{F}|)$, where $deg_i(g)$ denotes the degree of g in variable i.

For a detailed description of the sum check protocol and the proof of soundness, the reader is referred to Chapter 8 of [7]. Alternatively, the reader can refer to Theorem 4.2.3 or Section 5.2 for the details of the sum check protocol.

The sum check protocol is applied to the $d(n)$ different layers of the circuit in Theorem 4.1.3, giving a protocol in the Merlin-Arthur streaming model. For a succinct overview of the GKR protocol as well as the technical details of the proof of Theorem 4.1.3, the reader is referred to the full version of [101]. Due to the nature of the sum check protocol, the communication between the prover and verifier can only begin after the whole stream is seen. The running time of the prover was shown to be $O(S(n) \log S(n))$

in [29] which was improved by Thaler [101] to $O(S(n))$ for a large class of circuits.

4.2 An Online Merlin-Arthur Protocol for $\mathcal{PSPACE}_{cc}$

Definition 4.2.1. The class of decision problems that are solvable in polynomial space is denoted by $\mathcal{PSPACE}_{TM}$.

Shamir [96] proved that any language which is in $\mathcal{PSPACE}_{TM}$ has an interactive proof system, and a simplified proof was later given by Shen [97]. In this section, we show that Shen's proof can be adapted to give an online Merlin-Arthur communication protocol for any function in $\mathcal{PSPACE}_{cc}$, the communication complexity analogue of $\mathcal{PSPACE}_{TM}$ which we define in Definition 4.2.2. We note that it is not always the case that any transformation that works for Turing machines should carry over to online communication complexity classes. In particular, for Turing machines, by using the Goldwasser-Sipser transformation [51], we can convert any interactive proof protocol which uses private coins into one which uses public coins only, by adding at most two rounds and only incurring a polynomial blowup in complexity. Lokam [82] observed that a similar result hold for the non-online communication complexity counterpart[2]. As we have seen in Corollary 3.4.4, we have $OIP^2_{cc} \nsubseteq OMA^k_{cc}$ for any constant $k \geq 1$. This shows that the Goldwasser-Sipser transformation does not carry over to online communication.

Similar to the notion of alternating Turing machines, Babai, Frankl and

[2]For the non-online version of Merlin-Arthur communication complexity, Alice and Bob are allowed to communicate with each other. As an example, the AM protocol in Definition 3.3.2 is the non-online version of the OAM protocol in Definition 3.3.5.

Simon [8] also defined the notion of alternating communication protocols which gives rise to the complexity class $\mathcal{PSPACE}_{cc}$.

Definition 4.2.2. Let $l_1(n), \cdots, l_k(n)$ be nonnegative integer such that $l(n) := \sum_{i=1}^{k} l_i(n)$. We say that a $\mathcal{PSPACE}$ protocol $\mathscr{P}$ correctly computes $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ if there exist $l_1(n), \cdots, l_k(n)$ and Boolean functions $\varphi, \psi : \{0,1\}^{n+l(n)} \to \{0,1\}$ such that

$$f(x,y) = 1 \iff \exists\, u_1 \forall\, u_2 \cdots Q_k\, u_k \;\; [\varphi(x,u) \clubsuit \psi(y,u)]$$

where $u_i \in \{0,1\}^{l_i(n)}$, $u = u_1 \cdots u_k$, Q_k is the existential ($\exists$) or universal ($\forall$) quantifier if k is odd or even respectively and $\clubsuit$ represents $\bigvee$ if k is even and $\bigwedge$ if k is odd. The cost of $\mathscr{P}$ is $l(n)$ and $\mathcal{PSPACE}(f)$ is the cost of the optimal protocol which computes f correctly. We define the class $\mathcal{PSPACE}_{cc}$ to be the collection of all functions f which can be computed by a $\mathcal{PSPACE}$ protocol of cost at most polylog(n).

Any function f which has a $\mathcal{PSPACE}$ communication protocol with cost c can be written in the following form:

$$f(x,y) = 1 \iff Q_1 v_1\, Q_2 v_2 \cdots Q_c\, v_c \;\; [\varphi(x,v) \clubsuit \psi(y,v)]$$

where $v_i \in \{0,1\}$, $v = v_1, \cdots, v_c$, $Q_i \in \{\exists, \forall\}$ and $\clubsuit \equiv \bigwedge$ if $Q_c \equiv \exists$. Otherwise, if $Q_c \equiv \forall$, then let $\clubsuit \equiv \bigvee$.

Let us define the complexity class $OMA_{cc}^{\text{polylog}(n)} := \bigcup_{1 \le k \le \text{polylog}(n)} OMA_{cc}^{k}$. In Theorem 4.2.3, we adapt the proof techniques of [96, 97] to show that $\mathcal{PSPACE}_{cc} \subseteq OMA_{cc}^{\text{polylog}(n)}$.

Theorem 4.2.3. *Suppose $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ can be computed by a $\mathcal{PSPACE}$ protocol with cost c. Then there exist an OMA^{c^2+3c} protocol $\mathscr{P}$*

which computes f correctly where $vcost(\mathscr{P}) = O(c)$ and $hcost(\mathscr{P}) = O(c^3)$. The completeness and soundness error of $\mathscr{P}$ is $2^{-\Omega(c)}$.

Proof. We need to derive an OMA protocol to check if the formula

$$Q_1 v_1 \, Q_2 v_2 \cdots Q_c \, v_c \;\; [\varphi(x, v) \clubsuit \psi(y, v)] \tag{4.2}$$

is true for some Boolean functions $\varphi, \psi : \{0,1\}^{n+c} \to \{0,1\}$ if and only if $f(x, y) = 1$. Let $\widetilde{\varphi} : \mathbf{F}^c \to \mathbf{F}$, $\widetilde{\psi} : \mathbf{F}^c \to \mathbf{F}$ be the multilinear extension of φ, ψ to a finite field $\mathbf{F}$, whose size will be decided later. That is,

$$\widetilde{\varphi}(x, v_1, \cdots, v_c) = \sum_{z \in \{0,1\}^c} \varphi(x, z) \chi_z(v_1, \cdots, v_c)$$

where $\chi_z(v_1, \cdots, v_c) := \prod_{i=1}^{d} [(2z_i - 1) \, v_i + (1 - z_i)]$. Given x and y, define $P_{x,y}(v_1, \cdots, v_c) := \widetilde{\varphi}(x, v_1, \cdots, v_c) \, \widetilde{\clubsuit} \, \widetilde{\psi}(y, v_1, \cdots, v_c)$ where

$$a \, \widetilde{\clubsuit} \, b := \begin{cases} ab & \text{if } \clubsuit \equiv \bigwedge \\ a + b - ab & \text{if } \clubsuit \equiv \bigvee . \end{cases}$$

To check whether the formula in (4.2) is true, it is equivalent to checking the following identity:

$$\widetilde{Q_1} \, \widetilde{Q_2} \cdots \widetilde{Q_c} \, P_{x,y}(v_1, \cdots, v_c) = k > 0$$

where

$$\widetilde{Q_i} \equiv \begin{cases} \sum_{v_i \in \{0,1\}} & \text{if } Q_i \equiv \exists \\ \prod_{v_i \in \{0,1\}} & \text{if } Q_i \equiv \forall . \end{cases}$$

There are two main obstacles with this approach. The first is that the value of k can be as large as 2^{2^c}. Alice and Bob randomly choose a prime

$p \in \{2^c, \cdots, 2^{O(c)}\}$ using their shared randomness. Since k can have at most 2^c distinct prime factors, by the Prime Number theorem[3], it follows that the probability that $k = 0 \pmod{p}$ is at most $2^{-\Omega(c)}$. We work over the finite field $\mathbf{F} = \mathbf{F}_p$.

The second obstacle is that if we were to use the sum check protocol, the degree of the polynomial that needs to be communicated can be as large as 2^c. Since we are only evaluating the polynomial $P_{x,y}$ on the Boolean cube, following Shen's technique [97], we define the following linearization operator L_i in variable i as follows: Given any $g(X_1, \cdots, X_c) \in \mathbf{F}[X_1, \cdots, X_c]$, let

$$\begin{aligned} L_i\, g(X_1, \cdots, X_c) := X_i \cdot g(X_1, \cdots, X_{i-1}, 1, X_{i+1}, \cdots X_c) \\ + (1 - X_i) \cdot g(X_1, \cdots, X_{i-1}, 0, X_{i+1}, \cdots X_c). \end{aligned}$$

It is easy to see that whenever $X_i \in \{0, 1\}$, then the two polynomials $L_i\, g$ and g are equivalent. Merlin has to prove that over $\mathbf{F}$,

$$\widetilde{Q_1} L_1 \widetilde{Q_2} L_1 L_2 \cdots \widetilde{Q_c} L_1 \cdots L_c\, P_{x,y}(v_1, \cdots, v_c) \neq 0.$$

Protocol 4.2.1 shows how the sum check protocol can be used to give an OMA protocol for the following identity:

$$\diamondsuit_1 \cdots \diamondsuit_R\, P_{x,y}(v_1, \cdots, v_c) = k \neq 0 \pmod{p} \tag{4.3}$$

where $R = (c^2 + 3c)/2$ and $\diamondsuit_i$ is either $\sum_{v_j \in \{0,1\}}, \prod_{v_j \in \{0,1\}}$ or L_j for some $1 \leq j \leq c$. It is easy to see that if Merlin is honest, the protocol rejects

[3]The weaker statement $\pi(x) = \Theta\left(\frac{x}{\log x}\right)$ suffices for most computer science applications, where $\pi(x)$ is the prime-counting function that gives the number of primes between 1 and n. For a proof of this weaker statement, the reader is referred to Theorem A.23 in [7].

1. Alice and Bob randomly choose a prime $p \in \{2^c, \cdots, 2^{O(c)}\}$. Bob sends this prime p to Merlin.

2. (i) If $\diamondsuit_1 = \sum_{v_j \in \{0,1\}}$, then Merlin sends the polynomial

$$f(V) = \diamondsuit_2 \cdots \diamondsuit_R \, P_{x,y}\,(v_1, \cdots, v_{j-1}, V, v_{j+1}, \cdots, v_c)$$

to Bob. Bob checks if $f(0) + f(1) = k$.

(ii) If $\diamondsuit_1 = \prod_{v_j \in \{0,1\}}$, then Merlin sends the polynomial

$$f(V) = \diamondsuit_2 \cdots \diamondsuit_R \, P_{x,y}\,(v_1, \cdots, v_{j-1}, V, v_{j+1}, \cdots, v_c)$$

to Bob. Bob checks if $f(0) \cdot f(1) = k$.

(iii) If $\diamondsuit_1 = L_j$, then v_j has already been set to r_j before. In this case, Merlin sends the polynomial

$$f(V) = \diamondsuit_2 \cdots \diamondsuit_R \, P_{x,y}\,(v_1, \cdots, v_{j-1}, V, v_{j+1}, \cdots, v_c)$$

to Bob. Bob checks if $r_j \cdot f(1) + (1 - r_j) \cdot f(0) = k$.

3. Bob chooses $a \in \mathbf{F}_p$ uniformly at random and Bob and Merlin recursively check whether

$$f(a) = \diamondsuit_2 \cdots \diamondsuit_R \, P_{x,y}(v_1, \cdots, v_{j-1}, a, v_{j+1}, \cdots, v_c).$$

4. In the last round, Merlin and Bob need to check that

$$\beta = L_c P_{x,y}(r_1, \cdots, r_{c-1}, \gamma)$$

for some β which is known to both Bob and Merlin. Merlin sends Bob the polynomial $f(V) = P_{x,y}\,(r_1, \cdots, r_{c-1}, V)$. Alice sends $\widetilde{\varphi}(x, r_1, \cdots, r_c)$ to Bob.
Bob accepts if $\gamma \cdot f(1) + (1 - \gamma) \cdot f(0) = \beta$ and
$f(r_c) = P_{x,y}(r_1, \cdots, r_c)$ for a randomly chosen $r_c \in \mathbf{F}$.

Protocol 4.2.1: An OMA protocol which establishes (4.3).

with probability at most $2^{-\Omega(c)}$. By the union bound, since the degree of the polynomial communicated at each round is at most 2, the soundness error is at most $O(c^2/|\mathbf{F}|)$. The total communication between Merlin and Bob is at most $O(c^3)$ as the polynomial communicated at each round has degree at most two.

In the last round, Merlin sends Bob the polynomial $f(V) = P_{x,y}(r_1, \cdots, r_{c-1}, V)$ and Bob only would need to check if $f(r_c) = P_{x,y}(r_1, \cdots, r_c)$ for a randomly chosen $r_c \in \mathbf{F}$. Alice has to send the value $\widetilde{\varphi}(x, r_1, \cdots, r_c)$ to Bob in order for Bob to be able to evaluate $P_{x,y}(r_1, \cdots, r_c)$, which gives an $O(c)$ verification cost of this protocol. □

It follows from Theorem 4.2.3 that $\mathcal{PSPACE}_{cc} \subseteq OMA_{cc}^{\text{polylog}(n)}$. We denote the non-online version[4] of an OMA^k protocol by MA^k (Merlin-Arthur protocol with k messages). Following the notations in Chapter 3, we can define the communication complexity class MA_{cc}^k to be the collection of all functions that can be computed by a MA^k protocol with polylogarithmic cost. We define $MA_{cc}^{\text{polylog}(n)} := \bigcup_{1 \leq k \leq \text{polylog}(n)} MA_{cc}^k$. Lokam proved that $MA_{cc}^{\text{polylog}(n)} \subseteq \mathcal{PSPACE}_{cc}$ [82]. Note that trivially $OMA_{cc}^{\text{polylog}(n)} \subseteq MA_{cc}^{\text{polylog}(n)}$. As a result, "IP=PSPACE" holds for online communication complexity.

Corollary 4.2.4. $OMA_{cc}^{\text{polylog}(n)} = \mathcal{PSPACE}_{cc}$.

It is known that $\mathcal{NC} \subseteq \mathcal{PSPACE}_{cc}$ [82]. As a result, we have $\mathcal{NC} \subseteq OMA_{cc}^{\text{polylog}(n)}$, i.e. any problem which is decidable by circuits of size polynomial in n and depth polylogarithmic in n can be computed by a online Merlin-Arthur protocol with polylog(n) messages and whose cost is polylog(n).

[4]Please see the footnote on page 64.

4.3 Practical Interactive Protocols

For most functions in $\mathcal{NC}$ (like frequency moments), the GKR protocol is not optimal for practical purposes. In most cases, the GKR protocol requires $O(\log^2 m)$ rounds of interaction between the prover and verifier which may be large enough to be offputting. Cormode, Thaler and Yi [30] gave $\log m$ round protocols with polylog(m, n) complexity for many other interesting problems including F_2, Inner product and Range-sum, which they argued are more practical than the generic GKR protocol as they require less interaction between the prover and verifier. In an article in Forbes [41] in 2013, it was reported that the National Security Agency's data center in Utah will be capable of storing a yottabyte[5] of data. For a yottabyte-sized input, this corresponds to about 80 rounds of interaction if one uses a protocol with $\log m$ rounds. For a protocol with $\log^2 m$ rounds, more than 6000 rounds of interaction are needed. The reader is referred to [29] for the experimental evaluations of the GKR protocol and the improved interactive protocol with $\log m$ rounds when used to compute the exact value of F_2. It is evident from their experimental evaluations that the improved interactive protocol with $\log m$ rounds improves the prover's running time and the total communication between the prover and the verifier.

Let us briefly describe the polylog(m, n) cost SIP which uses $\log m$ rounds to compute the exact value of F_2. Let $f : [m] \to [n]$ be the frequency function, i.e. for any $1 \leq i \leq m$, $f(i)$ is the frequency of item i on the stream. In a natural way, we can view $f : \{0,1\}^{\log m} \to [n]$. Choose the smallest prime $p > \max\{n^2, 6 \log m\}$ and let $\mathbf{F} := \mathbf{F}_p$. Let $\widetilde{f} : \mathbf{F}^{\log m} \to \mathbf{F}$

[5] One yottabyte is 10^{24} bytes.

be the low degree extension of f over $\mathbf{F}$ which is given by

$$\widetilde{f}(X_1, \cdots, X_{\log m}) := \sum_{v \in \{0,1\}^{\log m}} f(v)\, \chi_v(X_1, \cdots, X_{\log m}) \quad \text{where}$$

$$\chi_v(X_1, \cdots, X_{\log m}) := \prod_{j=1}^{\log m} \left[(2v_j - 1)\, X_j + (1 - v_j)\right].$$

Note that $deg_i(\widetilde{f}) = 1$ for any $1 \leq i \leq m$. We can write

$$F_2 = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_{\log m} \in \{0,1\}} \widetilde{f}^{\,2}(x_1, \cdots, x_{\log m}).$$

Applying the sum check protocol, we get a SIP with $\log m$ rounds which computes the exact value of F_2. Note that the total communication is $O(\log m \log(|\mathbf{F}|))$ and the soundness error is at most $\frac{2 \log m}{|\mathbf{F}|} < 1/3$. The verifier needs to compute $\widetilde{f}(r_1, \cdots, r_{\log m})$ for a randomly chosen $r \in \mathbf{F}^{\log m}$ which can be computed in a streaming fashion as in Theorem 3.2.1. The space required by the verifier is $O(\log m \log(|\mathbf{F}|))$.

Theorem 4.3.1. *[30]*

There is a $(\log m(\log n + \log\log m), \log m(\log n + \log\log m))$ *SIP in the Merlin-Arthur streaming model with* $\log m$ *rounds that computes the exact value of* F_2*. This protocol has perfect completeness and the update time for the verifier per symbol received is* $O(\log m)$*.*

The SIP for the exact computation of F_2 can be modified to give SIPs with $\log m$ rounds for other interesting problems like higher order frequency moments, Inner Product and Range-sum. For more details, the reader is referred to [30].

In the work of [30], the authors gave SIP with $\log m + 1$ rounds, poly-logarithmic space and $\widetilde{O}(\sqrt{n})$ help cost for the exact computation of F_0 and F_∞. Let us first describe the $\log m + 1$ round protocol of [30] for the

exact computation of F_0. As we have seen in Chapter 3, applying the sum check protocol to (3.2) naively will result in the degree of the polynomial that is communicated at each round to be n. Similar to what we did in Chapter 3, we will remove the heavy hitters from the stream. We note that the verifier need not store all the heavy hitters in his memory.

Lemma 4.3.2. *There is a* $(\phi^{-1}\log m(\log m+\log n)\,,\,\log m(\log m+\log n))$ *SIP with* $\log m+1$ *rounds in the Merlin-Arthur streaming model that identifies all the* ϕ*-heavy hitters in a data stream. This protocol has perfect completeness.*

Proof. Consider a binary tree $\mathbb{T}$ of depth $\log m$ where the value of the i-th leaf is f_i. For any node $v \in V_{\mathbb{T}}$, denote $L(v)$ to be the set of leaves of the subtree rooted at v and let $p(v)$ be the parent of v. For every node $v \in V_{\mathbb{T}}$, we denote its value by $\widehat{f}(v) := \sum_{i\in L(v)} f_i$. We denote the witness set $W \subseteq V_{\mathbb{T}}$ which consists of all leaves l with $\widehat{f}(l) > T$ and all nodes v which satisfy $\widehat{f}(v) \leq T$ and $\widehat{f}(p(v)) > T$. This witness set ensures that no heavy hitters are omitted by the prover. Label the nodes of $\mathbb{T}$ in some canonical order from $\{1, 2, \cdots, 2m-1\}$. Let $x \in \{0,1\}^{2m-1}$ be the indicator vector for W, i.e. $x_j = 1$ if and only if the j-th node of $\mathbb{T}$ belongs to W. The prover gives the set W together with the claimed frequency $f^*(w)$ for each $w \in W$. If $w \notin W$, then define $f^*(w) = 0$. The verifier needs to check that the set W does cover the whole universe and that $f^*(w) = \widehat{f}(w)$ for all $w \in W$. This is equivalent to checking that

$$\sum_{j=1}^{2m-1} x_j \left(f^*(j) - \widehat{f}(j)\right)^2 = 0. \tag{4.4}$$

The sum check protocol applied to (4.4) will require $\log m + 1$ rounds of interaction. We have to choose a prime $q > (2m-1)n^2$ which will re-

quire $O(\log n + \log m)$ bits to represent q. At each level of the binary tree, there can be at most $2\phi^{-1}$ nodes that belong to W, which implies that $|W| = O(\phi^{-1} \log m)$. In each round of the sum check protocol, the prover communicates a polynomial of degree at most 3. Hence the total communication is dominated by $O(\phi^{-1} \log^2 m + \phi^{-1} \log m \log n)$.

The verifier need to pick a random point $r \in \mathbf{F}_q^{\log m+1}$ and needs to evaluate $\widetilde{x}(r)$, $\widetilde{f^*}(r)$ and $\widetilde{\widehat{f}}(r)$, where $\widetilde{x}$, $\widetilde{f^*}$ and $\widetilde{\widehat{f}}$ are the low degree extensions of x, f^* and $\widehat{f}$ respectively. As described in [30], $\widetilde{x}(r)$, $\widetilde{f^*}(r)$ and $\widetilde{\widehat{f}}(r)$ can be calculated in a single pass over the stream. Hence the space complexity of the verifier is $O(\log m(\log m + \log n))$. □

Theorem 4.3.3. *[30]*

There is a $(\sqrt{n} \log m(\log m+\log n)\,,\, \log m(\log m+\log n))$ *SIP with* $\log m+1$ *rounds that computes the exact value of* F_0 *in the Merlin-Arthur streaming model. This protocol has perfect completeness.*

Proof. The verifier removes all the ϕ-heavy hitters from the stream with the help of the prover. This gives rise to a new stream $\widetilde{\sigma}$ where the frequency of each element is at most ϕn. The sum check protocol is then applied to

$$F_0 = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_{\log m} \in \{0,1\}} \widetilde{g} \circ \widetilde{f'}(x_1, \cdots, x_{\log m}). \tag{4.5}$$

where $f' : \{0,1\}^{\log m} \to [\phi n]$ is the frequency function of the derived stream $\widetilde{\sigma}$, $g : \mathbf{N} \to \{0,1\}$ is given by $g(0) = 0$ and $g(x) = 1$ for $1 \leq x \leq \phi n$ and $\widetilde{f'}$ and $\widetilde{g}$ are the low degree extensions of f' and g respectively. We can work over the finite field $\mathbf{F}_q$, where q is chosen in Lemma 4.3.2. The sum check protocol on (4.5) requires $\log m$ rounds of interaction which will give a total communication of $O(\phi n(\log m + \log n) \log m)$. The space required by the verifier is $O(\log m(\log m + \log n))$. Note that the heavy hitter protocol and

the sum check protocol on (4.5) can be carried out in parallel. By choosing $\phi = \frac{1}{\sqrt{n}}$, we get Theorem 4.3.3. □

For the case of the exact computation of F_∞, we also have to use the heavy hitters protocol to remove all the items with high frequency from the stream. By a simple generalization of the 2 message IP streaming protocol for the INDEX problem, one can construct a protocol with cost $O(\log m(\log n + \log\log m))$ which will convince the verifier that $F_\infty \geq b$ where only three messages[6] are exchanged between the prover and verifier. In the case of an honest prover, b is the exact value of F_∞. If $b \geq \sqrt{n}$, then they execute the $\frac{b}{n}$-Heavy Hitter protocol to check that they are no items with frequency larger than b in the data stream. On the other hand, if $b < \sqrt{n}$, by executing the $\frac{1}{\sqrt{n}}$-Heavy Hitters protocol, the verifier can remove all the items whose frequencies are larger than $\sqrt{n}$. This gives rise to a new stream $\widetilde{\sigma}$ where the frequency of each element is at most $\sqrt{n}$. The sum check protocol is then used to check if

$$0 = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_{\log m} \in \{0,1\}} \widetilde{g} \circ \widetilde{f'}(x_1, \cdots, x_{\log m}). \tag{4.6}$$

where $f' : \{0,1\}^{\log m} \to [\sqrt{n}\,]$ is the frequency function of the derived stream $\widetilde{\sigma}$, $g : \mathbf{N} \to \{0,1\}$ is given by $g(x) = 0$ for $x = 0, \cdots, b$ and $g(x) = 1$ for $x = b+1, \cdots, \sqrt{n}$ and $\widetilde{f'}$ and $\widetilde{g}$ are the low degree extensions of f' and g respectively.

Theorem 4.3.4. *[30]*

There is a $(\sqrt{n}\log m(\log m + \log n)\,,\ \log m(\log m + \log n))$ *SIP with* $\log m + 1$ *rounds that computes the exact value of* F_∞. *This protocol has perfect completeness.*

[6] The "index" $i \in [m]$ such that $f_i = F_\infty$ has to be provided by the prover, which adds one more message to the protocol.

Cormode et al. [30] posed the problem of finding SIPs with polylogarithmic space and communication and $O(\log m)$ rounds for the exact computation of F_0 and F_∞. Cormode, Mitzenmacher, and Thaler [29] gave an alternative interactive protocol for F_0 based on linearization, whereby the prover is more efficient in terms of running time. Their protocol still requires $\log^2 m$ rounds of interaction where the verifier's space is $O(\log^2 m)$ bits and the total communication is $O(\log^3 m)$.

Chapter 5

An Improved Interactive Streaming Algorithm for F_0

As we have seen in Chapter 4, if we use the GKR generic protocol to compute the exact value of F_0, it will require $O(\log^2 m)$ rounds of interaction which can very quickly become impractical. In this chapter, we give a $\log m$ round protocol which computes the exact value of F_0 with $\text{polylog}(m, n)$ cost. This improves the $\log m$ round protocol for the exact computation of F_0 given in Theorem 4.3.3.

5.1 Overview of Our Techniques

Instead of removing the heavy hitters from the stream which we did in Section 4.3, we write F_0 as a different formula. Such an approach was first used by Gur and Raz [56] to obtain a protocol for the exact computation of F_0 in a slightly modified form of the annotation model where the prover and verifier are allowed to flip public coins before observing the stream. Here, the main technical point is to replace the OR polynomial on n variables which has high degree with an approximating polynomial over a smaller

finite field $\mathbf{F}_q$, so that this new polynomial has low degree. Such approximating polynomials were first constructed in [93, 99] to prove circuit lower bounds. The degree of the approximating polynomial $p : \mathbf{F}_q^n \to \mathbf{F}_q$ depends on q. But choosing q to be small forces us to work over the field $\mathbf{F}_q$ and the arithmetic will be correct only modulo q, i.e. F_0 will be calculated modulo q. Note that we cannot choose $q > m$ as the approximating polynomial degree would then be larger than m. By using the smallest $\log m$ primes, we can compute F_0 modulo these $\log m$ many primes with the help and verifier's cost being polylogarithmic in m and n. This does not increase the number of rounds because all these executions can be done in parallel. The exact value of F_0 can be constructed by the Chinese Remainder Theorem. As a result of decreasing the degree of the polynomial, our protocol no longer has perfect completeness. By parallel repetition, the probability that an honest prover succeeds can be made close to 1.

We will make heavy use of the Gur and Raz technique [56] to devise a $\log m$ round protocol with polylog(m, n) cost for the exact computation of F_0. Our main contributions are using the sum check protocol (See Lemma 4.1.4) on the formula for F_0 used by Gur and Raz [56] and showing that in this setup, the verifier can compute the value of a certain polynomial at any point in a streaming fashion.

5.2 The Algorithm

Given a multiset presented as a stream $\sigma = \langle a_1, \cdots, a_n \rangle$, where each $a_i \in [m]$, we give a interactive protocol with $\log m$ rounds which computes F_0 exactly. For each $j \in [m]$ and $i \in [n]$, we denote $\chi_i(j)$ to be the element indicator of element j at position i of the stream, i.e. $\chi_i : [m] \to \{0, 1\}$ such that $\chi_i(j) = 1 \Leftrightarrow a_i = j$. We can also interpret each $\chi_i : \{0, 1\}^{\log m} \to \{0, 1\}$

by associating each $j \in [m]$ with its binary expansion. It is easy to see that

$$F_0 = \sum_{j=1}^{m} \left(\bigvee_{i=1}^{n} \chi_i(j) \right) = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_d \in \{0,1\}} \mathrm{OR}_n \left(\chi(x_1, \cdots, x_d) \right),$$

where $d = \log m$, $\chi : \{0,1\}^d \to \{0,1\}^n$ is

$$\chi(x_1, \cdots, x_d) := (\chi_1(x_1, \cdots, x_d), \cdots, \chi_n(x_1, \cdots, x_d))$$

and $\mathrm{OR}_n : \{0,1\}^n \to \{0,1\}$ is the OR function on n variables.

Let us consider the low degree extension of χ_i over a larger field. Let q be a prime and λ be an integer to be determined later. We extend the domain of χ_i from $\mathbf{F}_2^d$ to $\mathbf{F}_{q^\lambda}^d$. If we denote $\theta_\sigma : [m] \to \mathbf{F}_2^d$ where $\theta_\sigma(i) = \left(a_i^{(1)}, \cdots, a_i^{(d)} \right)$ is the binary expansion of a_i, then the extension $\widetilde{\chi_i} : \mathbf{F}_{q^\lambda}^d \to \mathbf{F}_{q^\lambda}$ is given by

$$\widetilde{\chi_i}(x_1, \cdots, x_d) := \prod_{j=1}^{d} \left[\left(2a_i^{(j)} - 1 \right) x_j + \left(1 - a_i^{(j)} \right) \right]. \tag{5.1}$$

Note that $\widetilde{\chi_i}(x_1, \cdots, x_d) = \chi_i(x_1, \cdots, x_d)$ for all $x \in \mathbf{F}_2^d$. Similarly, define $\widetilde{\chi} : \mathbf{F}_{q^\lambda}^d \to \mathbf{F}_{q^\lambda}^n$ in the natural way:

$$\widetilde{\chi}(x_1, \cdots, x_d) := (\widetilde{\chi_1}(x_1, \cdots, x_d), \cdots, \widetilde{\chi_n}(x_1, \cdots, x_d)).$$

With this notation,

$$F_0 = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_d \in \{0,1\}} \mathrm{OR}_n \left(\widetilde{\chi}(x_1, \cdots, x_d) \right). \tag{5.2}$$

Applying the sum check protocol naively to (5.2) would require the prover to send a degree n polynomial in each round. We replace the OR

function in (5.2) with a low degree polynomial p chosen from a probability distribution over polynomials such that p approximates the OR function with high probability. This idea was first introduced in [93, 99] and was also used in [56] to obtain a protocol for exact F_0 in the slightly modified form of the annotation streaming model where the prover and verifier are allowed to flip public coins before observing the stream.

Definition 5.2.1. Let $p : \mathbf{F}^n \to \mathbf{F}$ be a polynomial over a field $\mathbf{F}$. We say that the individual degree of p is at most d in each variable if

$$p(X_1, \cdots, X_n) = \sum_{\substack{\alpha_i \in \mathbf{N} \\ \text{s.t. } 0 \leq \alpha_i \leq d}} c_{\alpha_1, \cdots, \alpha_n} X_1^{\alpha_1} \cdots X_n^{\alpha_n}.$$

where $c_{\alpha_1, \cdots, \alpha_n} \in \mathbf{F}$.

Lemma 5.2.2. *Using $O(L \log n)$ bits of randomness, we can construct a random polynomial $p : \mathbf{F}_q^n \to \mathbf{F}_q$ of individual degree at most $L(q-1)$ in each variable, such that for every $x \in \{0,1\}^d$,*

$$\mathbf{Pr}\left[p\left(\widetilde{\chi}(x_1, \cdots, x_d)\right) = \mathrm{OR}_n\left(\widetilde{\chi}(x_1, \cdots, x_d)\right)\right] \geq 1 - \frac{1}{6m \log m},$$

where L is the least integer such that

$$\left(\frac{2}{3}\right)^L \leq \frac{1}{6m \log m}. \tag{5.3}$$

Proof. Start with a $[\zeta n, n, \frac{1}{3}\zeta n]_q$-linear code $\mathcal{C}$, where $\zeta > 1$ is a constant to be chosen such that $\mathcal{C}$ exists[1]. Let G be the generator matrix of $\mathcal{C}$. Choose uniformly at random $\alpha_1, \cdots, \alpha_L \in [\zeta n]$ where L is the least integer that

[1] Justesen codes [65] are one example of a family of codes which have both constant relative distance and constant rate.

satisfies (5.3) and define

$$p(x_1, \cdots, x_n) := 1 - \prod_{i=1}^{L} \left[1 - ((Gx)_{\alpha_i})^{q-1}\right].$$

It is easy to see that the individual degree of p is at most $L(q-1)$ in each variable. By properties of the code $\mathcal{C}$, for any $x \in \{0,1\}^n$,

$$\mathbf{Pr}_{\alpha_1,\cdots,\alpha_L}\left[p(x) \neq \bigvee_i x_i\right] \leq \left(\frac{2}{3}\right)^L \leq \frac{1}{6m\log m}. \qquad \square$$

We note that $L = O(\log m)$. Since $\mathbf{F}_{q^\lambda}$ can be viewed as a vector space over $\mathbf{F}_q$, we can view $p : \mathbf{F}_q^n \to \mathbf{F}_q$ as $\widetilde{p} : \mathbf{F}_{q^\lambda}^n \to \mathbf{F}_{q^\lambda}$, by applying p componentwise. By the union bound, the probability that

$$\mathbf{Pr}\left[F_0 \pmod q = \sum_{x_1\in\{0,1\}} \cdots \sum_{x_d\in\{0,1\}} \widetilde{p}\left(\widetilde{\chi}(x_1,\cdots,x_d)\right)\right] \geq 1 - \frac{1}{6\log m}. \tag{5.4}$$

We first give an interactive protocol to compute $F_0 \pmod q$ with high probability. Let $q \leq 2\log m \log\log m + 2$ be a prime and λ be the smallest integer such that $q^{\lambda-1} \geq 6Ld\log m$. Before observing the stream, the prover and verifier agree on the code $\mathcal{C}$ as in Lemma 5.2.2. The verifier chooses $O(L\log n)$ random bits to define the polynomial $\widetilde{p}$ and sends this randomness to the prover. The verifier chooses randomly $r \in \mathbf{F}_{q^\lambda}^d$ and computes $\widetilde{p}\left(\widetilde{\chi}(r_1,\cdots,r_d)\right)$ in a streaming fashion. We now illustrate how the verifier computes $\widetilde{p}\left(\widetilde{\chi}(r_1,\cdots,r_d)\right)$ given a one-pass over the stream without storing the whole input.

For $1 \leq i \leq n$, let $y_i = \left(y_i^{(1)},\cdots,y_i^{(\lambda)}\right)$ where $y_i = \widetilde{\chi}_i(r_1,\cdots,r_d)$ and each $y_i^{(j)} \in \mathbf{F}_q$ for $1 \leq j \leq \lambda$. Each y_i can be computed when a_i is seen on

the stream using (5.1). Then

$$\widetilde{p}\left(\widetilde{\chi}(r_1,\cdots,r_d)\right) = \widetilde{p}\left(y_1,\cdots,y_n\right)$$
$$= \left(p\begin{bmatrix} y_1^{(1)} \\ \vdots \\ y_n^{(1)} \end{bmatrix}, \quad \cdots, \quad p\begin{bmatrix} y_1^{(\lambda)} \\ \vdots \\ y_n^{(\lambda)} \end{bmatrix} \right).$$

We show how to compute $p\begin{bmatrix} y_1^{(j)} \\ \vdots \\ y_n^{(j)} \end{bmatrix}$ for any $1 \leq j \leq \lambda$ in a streaming fashion. For $1 \leq j \leq \lambda$ and $1 \leq i \leq L$, $\mathcal{V}$ needs to compute

$$B_{ij} = \left(G\begin{bmatrix} y_1^{(j)} \\ \vdots \\ y_n^{(j)} \end{bmatrix} \right)_{\alpha_i} \tag{5.5}$$
$$= \left(G\begin{bmatrix} y_1^{(j)} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right)_{\alpha_i} + \cdots + \left(G\begin{bmatrix} 0 \\ \vdots \\ 0 \\ y_n^{(j)} \end{bmatrix} \right)_{\alpha_i}.$$

This can be done given one pass over the stream σ. Each time $\mathcal{V}$ observes a new entry a_k, he updates $B_{ij} \leftarrow B_{ij} + y_k^{(j)} \cdot g_{\alpha_i,k}$. Note that the computation of $y_k^{(j)}$ depends only on a_k and the verifier need not store matrix G. Upon observing entry a_k, only the $\alpha_1,\cdots,\alpha_L$ entries of the k-th column of G are relevant. Since G is locally logspace constructible, each $g_{\alpha_i,k}$ can be constructed in $O(\log n)$ space. After the stream has ended, the verifier

$$\text{computes } p \begin{bmatrix} y_1^{(j)} \\ \vdots \\ y_n^{(j)} \end{bmatrix} \text{ using}$$

$$p \begin{bmatrix} y_1^{(j)} \\ \vdots \\ y_n^{(j)} \end{bmatrix} = 1 - \prod_{i=1}^{L} \left(1 - B_{ij}^{q-1}\right).$$

After the stream ends, the verification protocol proceeds in d rounds to compute $F_0 \pmod q$ with probability at least $1 - \frac{1}{6 \log m}$. In the first round, the prover sends a polynomial $g_1(X_1)$ which is claimed to be

$$g_1(X_1) = \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_d \in \{0,1\}} \widetilde{p}\left(\widetilde{\chi_1}\left(X_1, x_2, \cdots, x_d\right), \cdots, \widetilde{\chi_n}\left(X_1, x_2 \cdots, x_d\right)\right).$$

The polynomial $g_1(X_1)$ has degree $L(q-1)$ which can be described in $O(Lq \log q^{\lambda})$ bits. The verifier need not store $g_1(X_1)$ but just need to compute $g_1(r_1)$, $g_1(0)$ and $g_1(1)$, which can be done in a streaming fashion. Note that if the prover is honest, then

$$F_0 \pmod q = g_1(0) + g_1(1). \tag{5.6}$$

In round $2 \leq j \leq d-1$, the verifier sends r_{j-1} to the prover who then sends the polynomial $g_j(X_j)$, which is claimed to be

$$g_j(X_j) = \sum_{x_{j+1} \in \{0,1\}} \cdots \sum_{x_d \in \{0,1\}} \widetilde{p}(\widetilde{\chi_1}\left(r_1, \cdots, r_{j-1}, X_j, x_{j+1}, \cdots, x_d\right), \cdots \\ \cdots, \widetilde{\chi_n}\left(r_1, \cdots, r_{j-1}, X_j, x_{j+1}, \cdots, x_d\right)). \tag{5.7}$$

The verifier computes $g_j(r_j)$, $g_j(0)$ and $g_j(1)$ and proceeds to the next round only if the degree of g_j is at most $L(q-1)$ and $g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$.

In the final round, the verifier sends r_{d-1} to the prover who then sends the polynomial $g_d(X_d)$, which is claimed to be

$$g_d(X_d) = \widetilde{p}\left(\widetilde{\chi_1}\left(r_1, \cdots, r_{d-1}, X_d\right), \cdots, \widetilde{\chi_n}\left(r_1, \cdots, r_{d-1}, X_d\right)\right).$$

The verifier only accepts that (5.6) is computed correctly if g_d is of the correct degree, $g_{d-1}(r_{d-1}) = g_d(0) + g_d(1)$ and $g_d(r_d) = \widetilde{p}\left(\widetilde{\chi}(r_1, \cdots, r_d)\right)$. Next, we show that if the prover is dishonest, the verifier will reject the claimed value of $F_0 \pmod q$ with high probability.

Lemma 5.2.3. *In the case of the honest prover, the verifier will accept the wrong value of* $F_0 \pmod q$ *with probability at most* $\frac{1}{6 \log m}$. *If*

$$\sum_{x_1 \in \{0,1\}} \cdots \sum_{x_d \in \{0,1\}} \widetilde{p}\left(\widetilde{\chi}(x_1, \cdots, x_d)\right) \tag{5.8}$$

correctly represents $F_0 \pmod q$ *and if the prover cheats by sending some polynomial which does not meet the requirements of the protocol, the verifier will accept with probability at most* $\frac{L(q-1)d}{q^\lambda}$.

Proof. In the case of an honest prover, since the interactive protocol always evaluates (5.8) correctly, the prover will fail in the case that the approximating polynomial $\widetilde{p}$ does not represent the OR function. By (5.4), the probability that the honest prover will fail is at most $\frac{1}{6 \log m}$.

For the case of the dishonest prover, the argument proceeds inductively from the d-th round to the first round. Indeed, if g_d is not as claimed, by the Schwartz-Zippel lemma,

$$\mathbf{Pr}\left[g_d(r_d) = \widetilde{p}\left(\widetilde{\chi}(r_1, \cdots, r_d)\right)\right] \leq \frac{L(q-1)}{q^\lambda}.$$

By induction, suppose for $1 \leq j \leq d-1$ that the verifier is convinced that $g_{j+1}(X_{j+1})$ is correct with high probability. He can verify the correctness of $g_j(X_j)$ with high probability; since

$$g_{j+1}(X_{j+1}) = \sum_{x_{j+2}\in\{0,1\}} \cdots \sum_{x_d\in\{0,1\}} \widetilde{p}(\widetilde{\chi_1}(r_1,\cdots,r_j,X_{j+1},x_{j+2},\cdots,x_d),\cdots \\ \cdots,\widetilde{\chi_n}(r_1,\cdots,r_j,X_{j+1},x_{j+2},\cdots,x_d)),$$

it is easy to see by (5.7) that $g_j(r_j) = g_{j+1}(0) + g_{j+1}(1)$. By the Schwartz-Zippel lemma,

$$\mathbf{Pr}\left[\hat{g}_j(r_j) = g_{j+1}(0) + g_{j+1}(1)\right] \leq \frac{L(q-1)}{q^\lambda} \qquad \text{where}$$
$$\hat{g}_j(X_j) \neq \sum_{x_{j+1}\in\{0,1\}} \cdots \sum_{x_d\in\{0,1\}} \widetilde{p}(\widetilde{\chi_1}(r_1,\cdots,r_{j-1},X_j,x_{j+1},\cdots,x_d),\cdots \\ \cdots,\widetilde{\chi_n}(r_1,\cdots,r_{j-1},X_j,x_{j+1},\cdots,x_d)).$$

For the cheating prover to succeed, he has to give

$$\hat{g}_1(X_1) \neq \sum_{x_2\in\{0,1\}} \cdots \sum_{x_d\in\{0,1\}} \widetilde{p}\left(\widetilde{\chi_1}(X_1,x_2,\cdots,x_d),\cdots,\widetilde{\chi_n}(X_1,x_2\cdots,x_d)\right)$$

such that either in some round $j+1$ (for some $1 \leq j \leq d-1$), when the verifier reveals r_j, it should satisfy

$$\hat{g}_j(r_j) = g_{j+1}(0) + g_{j+1}(1)$$

or $\hat{g}_d(r_d) = \widetilde{p}\left(\widetilde{\chi}(r_1,\cdots,r_d)\right)$ in the final round. By the union bound,

$$\mathbf{Pr}\left[\hat{g}_1(X_1) \neq g_1(X_1) \text{ and the verifier accepts}\right] \leq \frac{L(q-1)d}{q^\lambda} \leq \frac{1}{6\log m}. \quad \square$$

Analysis of space and communication. We now analyse the space

needed by the verifier and the total communication between the prover and verifier over the $\log m$ rounds to verify $F_0 \pmod q$. First, let us look at the space complexity of the verifier. He needs to store $\alpha_1, \cdots, \alpha_L$ which will take $O(\log m \log n)$ bits of space. With $O(\log m \cdot \log\log m \cdot \log q)$ bits of space, the verifier can compute $\widetilde{p}\,(\widetilde{\chi}(r_1, \cdots, r_d))$ when observing the stream. Note during the interaction with the prover after the stream ends, at each round $1 \le j \le d$, the verifier need not store the polynomial $g_j(X_j)$ but only needs to evaluate g_j at a constant number of points. Hence, the space complexity of the verifier is $O\left(\log m\,[\log n + \log\log m \cdot \log q\,]\right)$ bits.

We now bound the total communication between the prover and verifier. The verifier needs to communicate $\alpha_1, \cdots, \alpha_L$ and $r_1, \cdots, r_{d-1}$ to the prover, with cost $O(\log m \log n)$ and $O(\log m \log\log m)$ respectively. The prover, who needs to send $g_1(X_1), \cdots, g_d(X_d)$, uses $O(dLq \log q^\lambda) = O(q \log^2 m \cdot \log\log m)$ bits to communicate all these polynomials. Hence, the total communication is $O\left(\log m\,(\log n + q \log m \cdot \log\log m)\right)$ bits. We summarize our result below.

Lemma 5.2.4. *There exists a (h, v) SIP with $\log m$ rounds with*

$$h = \log m\,(\log n + q \log m \cdot \log\log m)\,,$$
$$v = \log m\,(\log n + \log\log m \cdot \log q\,)$$

that computes $F_0 \pmod q$ for any prime $q \le 2\log m \log\log m + 2$, where the completeness and soundness errors are

$$\epsilon_c = \frac{1}{6\log m} \quad \textit{and} \quad \epsilon_s = \frac{1}{3\log m}.$$

Computing F_0 exactly. Lemma 5.2.4 gives us a streaming interactive protocol to verify the correctness of $F_0 \pmod q$ with high probability for

any prime $q \leq 2\log m \log\log m + 2$. Now, we show how the prover can verify F_0 with high probability. Let $Q = \{q_1, \cdots, q_{\log m}\}$ be the first $\log m$ primes. Note that $q_{\log m} \leq 2\log m \log\log m + 2$ for all $m \geq 2$ [11] and $\prod_{i=1}^{\log m} q_i > m$. So, the verifier will compute $F_0 \pmod{q_i}$ for $i = 1, \cdots, \log m$. Note that this can be done in parallel and will cause the working space of the verifier and the total communication to increase, but the number of rounds is still $\log m$. By using the Chinese remainder theorem, the verifier can compute F_0 exactly given $F_0 \pmod{q_i}$ for $i = 1, \cdots, \log m$. By the union bound, the completeness and soundness error are $1/6$ and $1/3$ respectively.

In the preprocessing phase (even before seeing the data), the verifier and prover agree on a constant $\zeta > 0$ such that the linear codes $\mathcal{C}_i := [\zeta n, n, \frac{1}{3}\zeta n]_{q_i}$ exist for all $1 \leq i \leq \log m$. Note that the same $\alpha_1, \cdots, \alpha_L$ can be used to define the polynomial $\widetilde{p}_i : \mathbf{F}_{q_i^\lambda}^n \to \mathbf{F}_{q_i^\lambda}$ for each $1 \leq i \leq \log m$. For each $1 \leq i \leq \log m$, the verifier needs to choose uniformly at random $r^{(i)} \in \mathbf{F}_{q_i^\lambda}^d$ and compute $\widetilde{p}\left(\widetilde{\chi}\left(r^{(i)}\right)\right)$. The space needed to store $r^{(i)}$ for $1 \leq i \leq \log m$ is $O(d \log q_i^\lambda \cdot \log m) = O(\log^2 m \log\log m)$. The space needed to compute $\widetilde{p}\left(\widetilde{\chi}\left(r^{(i)}\right)\right)$ for $1 \leq i \leq \log m$ is

$$O\left(\log m \log\log m \sum_{i=1}^{\log m} \log q_i\right) = O\left(\log^2 m \cdot (\log\log m)^2\right),$$

where we used the fact that $\sum_{\substack{\text{p : p is prime} \\ \text{and } p \leq x}} \log p = \Theta(x)$ [11]. Hence, the total space used by the verifier is $O(\log m\,(\log n + \log m \cdot (\log\log m)^2))$.

To bound the total communication, we need the following fact: Let p_n be the n^{th} prime. Then it is known that $\sum_{i=1}^{n} p_i = \Theta(n^2 \log n)$ for all

$n \geq 2$ [11]. Hence, the total communication is

$$O\left(\log m \log n + (\log^2 m \log\log m) \sum_{i=1}^{\log m} q_i\right)$$
$$= O\left(\log m \log n + \log^4 m \cdot (\log\log m)^2\right).$$

Running time of the verifier. First, we analyze the processing time of each symbol seen in the stream. We suppose it takes unit time to add and multiply two field elements from $\mathbf{F}_{q^\lambda}$. For each symbol a_k seen, the verifier needs to compute $\widetilde{\chi_k}\left(r^{(q)}\right)$ where $r^{(q)} \in \mathbf{F}_{q^\lambda}^d$ for each $q \in Q$. From (5.1), it is easy to see that the verifier needs $O(d) = O(\log m)$ time to compute $\widetilde{\chi_k}\left(r^{(q)}\right)$ for each $q \in Q$. Hence the total time taken by the verifier to process each symbol is $O(\log^2 m)$. After this, the verifier can discard a_k and only needs to update matrices B for each $q \in Q$. In a separate work space, the verifier can update matrices B using (5.5) after computing $\widetilde{\chi_k}\left(r^{(q)}\right)$ for each $q \in Q$. Note that after $\widetilde{\chi_k}\left(r^{(q)}\right)$ is computed, the updating of B does not require a_k anymore.

We summarize our results.

Theorem 5.2.5. *There exists a (h, v) SIP with $\log m$ rounds with*

$$h = \log m \left(\log n + \log^3 m \cdot (\log\log m)^2\right),$$
$$v = \log m \left(\log n + \log m \cdot (\log\log m)^2\right)$$

that computes F_0 exactly, where the completeness and soundness error are $1/6$ and $1/3$ respectively. The update time for the verifier per symbol received is $O(\log^2 m)$.

5.3 Comparison of Our Results

We compare our results with previously known non-interactive and interactive protocols that compute F_0 exactly. For comparison purposes, we assume that $m = \Theta(n)$. The results are collected in Table 5.3.1.

Paper	Space	Total Communication	Rounds
[21]	$m^{2/3} \log m$	$m^{2/3} \log m$	1
[29]	$\log^2 m$	$\log^3 m$	$\log^2 m$
[30]	$\log^2 m$	$\sqrt{m} \log^2 m$	$\log m$
Our work	$\log^2 m (\log \log m)^2$	$\log^4 m \, (\log \log m)^2$	$\log m$

Table 5.3.1: Comparison of our protocol to previous protocols for computing the exact number of distinct elements in a data stream. The results are stated for the case where $m = \Theta(n)$. The complexities of the space and the total communication are correct up to a constant.

We note that if we fix the number of rounds to $\log m$, our work improves the total communication from $O\left(\sqrt{m} \log^2 m\right)$ to $O\left(\log^4 m \, (\log \log m)^2\right)$, while only increasing the verifier's space by a multiplicative factor of $(\log \log m)^2$.

Gur and Raz studied the exact computation of F_0 in a more general annotation model where the prover and verifier are allowed to flip public coins before observing the stream [56]. In this streaming model, they gave a protocol for the exact computation of F_0 whose cost is $\sqrt{m} \cdot \text{polylog}(m, n)$.

Chapter 6

A New Model for Verifying Computations on Data Streams

In this chapter, we define a new model for data streaming algorithms that employ a prover/helper to outsource difficult computations in a verifiable way. While for the verifier the usual time (per symbol read) and space constraints of the data streaming model are in place, the prover has unbounded space. Neither party can look into the future (i.e., they do not know data arriving later). As we have seen in the previous chapters of this thesis, prior work on such models [20, 21, 28–30, 56, 74, 102] either severely restricted the *total* communication between the prover and the verifier, or extended the computation by a long annotation that has to be streamed from the prover to the verifier offline after the original stream has ended, delaying the computation of the result.

We will investigate a streaming model that only bounds the communication overhead, i.e., the amount of communication sent from the prover to the verifier per symbol of the data stream. This allows for vastly more

communication between the prover and verifier while maintaining the online nature of the model (in particular long annotations sent after the stream has ended are not allowed).

6.1 Motivation

In this chapter, we propose to relax the restriction on the total communication placed in previous works. We believe this to be an unnatural restriction. In fact, since the data stream does not usually originate with the provider processing it (the prover), giving the prover access can be understood as uploading of data. In practice, uploading speed for most internet users is smaller by some factor than downloading speed. So why should the length of the messages from the prover be restricted massively? We find it more natural to restrict the communication overhead, i.e., the amount of communication exchanged per symbol on the data stream. In short, while the user/verifier uploads a symbol, he should also be able to download the next chunk of a proof, potentially even a constant factor larger than the coding length of the symbol. Furthermore, as we have seen in the previous chapters, interactive proofs are known to be much more powerful than noninteractive proofs in most scenarios, so the user/verifier might as well upload short questions to the helper/prover.

What can we hope to gain from this relaxation of the model? First of all, in the model allowing only polylogarithmic total communication between the prover and the verifier, it is known that all functions in $\mathcal{NC}$ can be computed efficiently (see Chapter 4). So we might hope to find algorithms for problems not known to be in $\mathcal{NC}$. Beside this, the generic protocols for $\mathcal{NC}$ functions are complex and often not well tuned to specific problems. One might wish for simpler and more efficient algorithms. Furthermore

there is the following bottleneck in the algorithms described in [29, 30, 74]. Their algorithms are based on the sum check protocol and involve an additional verification phase after the stream has ended (or, once an output is required). This verification phase is quite short (polylogarithmic rounds and communication), but is in fact the only communication between the prover and verifier performed at all. This implies that all the work of the prover that depends on random challenges sent from the verifier must essentially be done during this phase, i.e., a computation that is at least linear in the size of the stream (and often worse) must be squeezed into this short phase, in which no new inputs arrive, and the verifier is no more busy than usual. This leads to a huge computation bottleneck for the prover, which we hope to address with our model. Essentially in previous protocols, the prover is almost completely idle before the onset of the verification phase, and then has to crank up his machines to perform exponentially more operations than the verifier during the verification phase. We will also investigate whether for all functions in $\mathcal{NC}$, is it possible to avoid the additional verification phase after the stream has ended. This is a natural question to ask.

6.2 The New Model

For simplicity, in this chapter, we will consider m and n such that $\log m = \Theta(\log n)$ where, as in the previous chapters, m is the size of the universe and n is the length of the stream.

Both the prover and verifier observe the stream in an online fashion, i.e. the prover does not know the input stream beforehand. The verifier is a Turing machine that has space bounded by polylog(n) and processes each symbol in time polylog(n). The prover, on the other hand is a Turing

machine that has unlimited workspace, and processes each symbol in some time $T(n)$ that will vary from problem to problem. In our model, both the prover and verifier have access to their own private randomness. This seems necessary in the online case as the prover does not know the whole input beforehand. Indeed, we do require the verifier to run in total time $n \cdot \text{polylog}(n)$, and the prover to run in time $n \cdot T(n)$, but we furthermore require this time to be spread out evenly.

Both the prover and verifier can communicate, under the following restrictions. For each symbol seen (i.e., right after that symbol a_i has appeared on the stream), the verifier and prover can exchange $O(1)$ messages, each of length $C(m, n) \cdot \log m$, where $C(m, n) = O(1)$ in general. We refer to $C(m, n)$ as the communication overhead, because we think of a_i as a message of length $\log m$ sent from the verifier to the prover. Note that this allows the total communication from the prover to the verifier to exceed $n \log m$, so proofs can be long (and highly interactive), but must be produced online, i.e., before the stream has been seen completely, and the communication cannot contain very long messages between two consecutive symbols. In the annotation model (See Definition 3.1.1), only the prover can send a message to the verifier.

In short, the goal of defining our model in such a manner is to make the verifier extremely storage efficient and operate in an online fashion, offloading storage and computational load to the prover, while keeping the results verifiable. In order to fully describe an algorithm in our model, one also has to give instructions for the honest prover (since we are interested in his computation overhead). We define the correctness of a computation with regard to the soundness and completeness as in Definition 3.1.1. The completeness error is the probability over the random choices of the verifier

that a correct claim is rejected by the verifier. The soundness error is the probability that an incorrect claim is accepted. The total error is the maximum of the soundness and completeness errors.

6.3 Algorithms In Our New Model

In this section, we give algorithms for four different problems in our new model which bounds the communication overhead instead of the total communication. To begin, let us define these four different problems.

Definition 6.3.1. (**Median**)

Given a stream $\sigma = \langle a_1, \cdots, a_n \rangle$ where $a_i \in [m]$, let $\pi : [n] \to [n]$ be a permutation that sorts the stream, i.e. π is a function such that $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$. If n is odd, the median is $a_{\pi(\lceil n/2 \rceil)}$. Otherwise, if n is even, then the median is $\frac{1}{2}\left(a_{\pi(n/2)} + a_{\pi(n/2+1)}\right)$. The parity of n is irrelevant from an algorithmic point of view[1]. For the purpose of this thesis, we redefine the median of the stream σ to be $a_{\pi(\lceil n/2 \rceil)}$.

Definition 6.3.2. (**LIS**)

Let $\sigma = \langle a_1, \cdots, a_n \rangle$ where $a_i \in [m]$. A subsequence of σ of length k is a sequence $a_{i_1}, a_{i_2}, \cdots, a_{i_k}$ with $i_1 < i_2 < \cdots < i_k$. Such a subsequence is said to be increasing if $a_{i_1} \leq a_{i_2} \leq \cdots \leq \cdots a_{i_k}$. $LIS(\sigma)$ is the length of the longest increasing subsequence, i.e., the largest k such that there is a increasing subsequence of length k. We say the number y is a $(1-\epsilon)$-approximation of $LIS(x)$ if $(1-\epsilon) \cdot LIS(x) \leq y \leq LIS(x)$.

Definition 6.3.3. (**FULL RANK**)

Let $\sigma = \langle a_{1,1}, a_{1,2}, \cdots, a_{1,n}, a_{2,1}, a_{2,2}, \cdots, a_{2,n}, \cdots, a_{n,1}, a_{n,2}, \cdots, a_{n,n} \rangle$ be a

[1] If n is even, we can execute two copies of the algorithm in parallel. The first algorithm will evaluate the median of the stream $\langle \sigma, m+1 \rangle$, while the second algorithm will evaluate the median of the stream $\langle \sigma, 0 \rangle$. The average of these two medians is the actual median if n is even.

stream of length n^2, where $a_{i,j} \in [m]$. Define the $n \times n$ matrix A such that $A(i,j) = a_{i,j}$ for all $1 \leq i \leq n$ and $1 \leq j \leq n$. Consider $rank(A)$, the rank of the matrix A over $\mathbf{R}$. The FULL RANK problem is a decision problem which outputs 1 if and only if $rank(A) = n$.

Definition 6.3.4. (**Perfect matching**)

Let $\sigma = \langle a_{1,1}, a_{1,2}, \cdots, a_{1,n}, a_{2,1}, a_{2,2}, \cdots, a_{2,n}, \cdots, a_{n,1}, a_{n,2}, \cdots, a_{n,n} \rangle$ be a stream of length n^2, where $a_{i,j} \in \{0,1\}$. Define the $n \times n$ matrix A such that $A(i,j) = a_{i,j}$ for all $1 \leq i \leq n$ and $1 \leq j \leq n$. Let A be the adjacency matrix of a graph $G = (V, E)$. A matching $M \subseteq E$ is a subset of the edges such that no two edges share a common vertex. A perfect matching is a matching such that exactly one edge of the matching is incident on each vertex of the graph.

We give a summary of the protocols that we devised in our new model.

1. For the Median problem, our protocol has communication overhead $6+o(1)$, polylogarithmic time per symbol for both verifier and prover, verifier's space is $O(\log n + \log(1/\epsilon))$ with soundness error ϵ.

2. For the Longest Increasing Subsequence(LIS) problem, we computed a $(1-\epsilon)$-approximation using constant communication overhead, space $O(\log^2 n/\epsilon)$, which succeeds with probability $1 - 1/n$.

3. For the FULL RANK problem, given a square matrix A which is streamed row-wise, where $|a_{ij}| \leq n$, there is a streaming algorithm which can decide if A has full rank over the reals where the verifier uses $O(\log n)$ storage and the total communication overhead is constant with probability of failure $O\left(\frac{\log n}{n}\right)$.

4. For the perfect matching problem, given an adjacency matrix of the graph G which is streamed row-wise, there is a streaming algorithm

which can decide if G has a perfect matching where the verifier uses $O(\log n)$ storage and the total communication overhead is constant with probability of failure $O\left(\frac{\log n}{n}\right)$.

By relaxing the total communication restriction, we managed to find an algorithm for the perfect matching problem which is not known to be in $\mathcal{NC}$ (See Subsection 6.3.3.7 for details.). Our algorithm for the perfect matching problem maintains the fully online nature of streaming.

Our protocols work in phases. The main idea is to partition the stream in phases P_k of length $n/2^k$ for $k = 1, \ldots, \log n$. During the first phase P_1, we observe the first $n/2$ elements of the stream, followed by the next $n/4$ elements in phase P_2 etc. During phase P_{k+1}, we "restream" the elements seen during phase P_k and some additional data which "measures" the progress of the algorithm. Since we require that our communication overhead remains a constant, the size of the additional data which is streamed during P_{k+1} should be $O\left(n/2^k\right)$. We describe the measure of progress for the Median and the FULL RANK protocol briefly.

For the Median protocol, the size of the set of candidates for the median shrinks by a factor of two as the phases proceed. For the FULL RANK protocol, consider the row space spanned by the first $n\left(1 - \frac{1}{2^k}\right)$ rows of the matrix, which has dimension $n\left(1 - \frac{1}{2^k}\right)$ if the matrix has full rank. The measure of progress will be the dual space whose dimension is $\frac{n}{2^k}$ and thus gets smaller and smaller as the phases proceed.

6.3.1 Median

In this subsection, we show how to compute the median of n numbers from a size n^2 universe arriving on a data stream, with the help of a prover. Recall that the median of $a_1, \cdots, a_n$ is the element with rank $n/2$ in the

sorted sequence[2]. Our algorithm also serves as an introduction to the main ideas for all our algorithms: splitting the input into geometrically shrinking phases, re-streaming parts of the data stream (usually during the next phase), and using some measure of progress that allows us to restrict the length of the re-streaming.

While the Median problem is in $\mathcal{NC}$, and hence by Theorem 4.1.3, there is a streaming algorithm in the Merlin-Arthur streaming model that uses only polylog(n) space and communication, this general algorithm has two drawbacks. First, there is an additional verification phase after the stream has ended, including a $O(\log n)$ round conversation between the prover and verifier after the stream has ended[3]. Secondly, during this phase the prover has to perform a linear amount of computation, while being mostly idle while reading the stream. The algorithm given in this section increases the amount of communication between the prover and verifier, but is substantially simpler than the algorithm of Theorem 4.1.3, and has neither an additional verification phase nor a large computation overhead for the prover. Moreover in our protocol, the verifier does not need to "speak" to the prover. Only the prover communicates with the verifier at different phases of the streaming process.

It is important to recall that the prover in our model is not clairvoyant, i.e., does not know the median in advance. Otherwise he could simply send the median to the verifier, who could store it, and then count all the elements on the stream that are smaller. But for a moment, consider a setting, in which the prover and verifier are allowed to see the whole

[2]Throughout the rest of the thesis, we ignore floor and ceiling operations.

[3]One has to use the Inner product protocol in [30] to obtain a $O(\log n)$ round protocol in the Merlin-Arthur streaming model for Median as the generic protocol for $\mathcal{NC}$ will need $O(\log^2 n)$ rounds. Likewise, the reader can modify the $\log m$ round protocol that we gave for the exact computation of F_2 in Theorem 4.3.1 to obtain a $O(\log n)$ round protocol for Median in the Merlin-Arthur streaming model.

stream, and can exchange arbitrarily long messages after the end of the stream (the definition of our model does not allow this, since this would violate the communication overhead bound). In such a setting we can find a very simple algorithm as well: the verifier computes a fingerprint of the original stream while seeing the stream. The prover computes the median during the original stream, and then, after the stream has ended sends the median x to the verifier, and furthermore re-streams the whole original stream. The verifier can now operate as if in the clairvoyant model, and check if x is indeed the median of the re-streamed numbers. Furthermore the verifier must test whether the prover faithfully re-streams, using his fingerprint.

In our median algorithm we try to emulate this approach, by "folding" the re-streaming of parts of the data stream into the original stream while maintaining small communication overhead. Since the prover does not know the median after, say $3n/4$ steps, he has to provide some partial information to the verifier.

Our algorithm is based on the following observation. Consider a stream of n numbers $a_n, \cdots, a_1$ with $0 \leq a_i \leq n^2$ for concreteness (for convenience a_n arrives first). After $n/2$ numbers have appeared on the stream, each of them could still be the median (due to $n/2$ numbers still appearing in the future). However, after we have seen $n(1 - 1/2^k)$ numbers, at most $n/2^k$ of the numbers already seen are still candidates for becoming the median: all others either have rank smaller than $n/2 - n/2^k$ or larger than $n/2$ in the sorted sequence and are no longer candidates for being the median, no matter what the rest of the stream will look like.

The main idea now is to partition the stream in phases of length $n/2^k$ for $k = 1, \ldots, \log n$. Phase $k + 1$ is used to simulate the re-streaming

idea described above. But note that during this phase we can only re-stream $O(n/2^k)$ elements, and not the whole previous stream. It is now very convenient since there are only $n/2^k$ candidates left that are worth re-streaming. But this is not enough, because the verifier can only establish correctness of the re-streaming with his fingerprint if the whole stream he has computed the fingerprint from is repeated, not some selection from it.

We now describe the algorithm in detail. Denote by P_k the k-th phase of the algorithm, as well as the numbers appearing on the stream in the k-th phase, i.e., $a_{n/2^{k-1}}, \cdots, a_{n/2^k+1}$. In the following, let $\widetilde{FP}()$ denote the multiset fingerprint of a stream of length at most n with reliability $1-\frac{\epsilon}{2\log n-1}$. According to Lemma 3.1.3, such a fingerprint can be computed online with space $O(\log n - \log \epsilon)$.

- In P_1, the verifier computes the fingerprint $F_1 = \widetilde{FP}(P_1)$ of the first $n/2$ numbers, i.e. $a_n, \cdots, a_{n/2+1}$. The prover sorts the numbers. Let C_1 denote the sequence of the first $n/2$ numbers in the sorted stream.

- In P_2, the verifier computes the fingerprint $F_2 = \widetilde{FP}(P_2)$ of the next $n/4$ numbers, i.e. $a_{n/2}, \cdots, a_{n/4+1}$. The prover keeps sorting, and determines C_2, the sequence of numbers with ranks between $n/4$ and $n/2$ (relative to $P_1 \circ P_2 := a_n, \cdots, a_{n/4+1}$), in their sorted ordering. Let y_2 denote the minimum of C_2 and z_2 the maximum of C_2.

- In P_k for $k > 2$, the following happens. Let C_k denote the sequence of numbers with rank $n/2 - n/2^k$ up to $n/2$ in $P_1 \circ \cdots \circ P_k := a_n, \cdots, a_{n/2^k+1}$ in their sorted ordering. At the beginning of P_k, the verifier knows F_{k-1} (the fingerprint of P_{k-1}) and also the fingerprint of C_{k-2}. The prover knows C_{k-1}.

  1. The (honest) prover sends the elements y_{k-1} and z_{k-1} of C_{k-1}

that have rank $n/2 - n/2^{k-1}$ and rank $n/2$ (relative to $P_1 \circ \cdots \circ P_{k-1} := a_n, \cdots, a_{n/2^{k-1}+1}$).

2. The prover re-streams C_{k-2} and P_{k-1}. The verifier checks the re-streaming of C_{k-2} and P_{k-1} against his fingerprints, and computes the fingerprints of the sequence C_{k-1} as well as of the sequence P_k. Note that C_{k-1} is easy to determine from $C_{k-2}, P_{k-1}, y_{k-1}, z_{k-1}$. Indeed to determine C_{k-1}, the prover first divides the stream seen during P_{k-1} into three parts. Let P^1_{k-1} denote all the elements of P_{k-1} that are less than or equal to y_{k-2}, P^2_{k-1} be all the elements of P_{k-1} that are between y_{k-2} and z_{k-2} and lastly, P^3_{k-1} denote all the elements of P_{k-1} that are greater than or equal to z_{k-2}. The prover will send P^1_{k-1}, then $P^2_{k-1} \circ C_{k-2}$ in their sorted order (indicating whether the element is from P^2_{k-1} or C_{k-2}) and finally P^3_{k-1} to the verifier. It is easy to see that such a procedure enables the verifier to check the correctness of y_{k-1} and z_{k-1} and also to compute the fingerprint of C_{k-1}.
3. Note that $|C_{k-2}| = 4n/2^k + 1$ for $k \geq 4$, $|P_{k-1}| = 2n/2^k$ and $|P_k| = n/2^k$. As a result, the communication overhead is $6+o(1)$.
4. At the end of P_k, the prover determines the set C_k which consists of all elements that have rank between $n/2 - n/2^k$ and $n/2$.

- When $n/2^k = O(1)$, then the verifier stores C_{k-1}, instead of fingerprinting it. He then stops talking to the prover, and instead sorts the remaining sequence himself (note that elements smaller than y_{k-1} or larger than z_{k-1} can be ignored). The verifier can determine the median at the end of the stream.

Theorem 6.3.5. *The Median problem can be solved with communication overhead $6 + o(1)$, polylogarithmic time per symbol for both verifier and*

prover and the verifier's space is $O(\log n + \log(1/\epsilon))$. *The error of the protocol is* ϵ.

Proof. The communication overhead was shown to be 6 above, with the $o(1)$ accounting for the communication of the y_k and z_k. It is clear that this algorithm can be implemented using polylog time per symbol for both the verifier and the prover. The verifier at any given time needs to store at most two fingerprints, and a constant number of elements from the stream. Each fingerprint needs $O(\log n - \log \epsilon)$ bits of storage.

If the prover tries to cheat, he will succeed if any one of the fingerprint of $P_1, \cdots, P_{\log n}$ or $C_1, \cdots, C_{\log n - 1}$ fails. Since the reliability of each of the fingerprint is $1 - \frac{\epsilon}{2 \log n - 1}$ and there are at most $2 \log n - 1$ fingerprints computed by the verifier, by the union bound, the error of the protocol is ϵ. □

6.3.2 Longest Increasing Subsequence

In this subsection, we give an algorithm structurally similar to the algorithm from Subsection 6.3.1 to compute the Longest Increasing Subsequence (LIS) of a data stream with the help of a prover. Given a sequence/stream $x_1, \cdots, x_n$ of elements from $\{1, \cdots, n^2\}$ (n^2 again chosen for concreteness), the length of the longest increasing subsequence $LIS(x)$ is the largest k such there exist $i_1 < \cdots < i_k$ with $x_{i_j} \leq x_{i_{j+1}}$ for all $1 \leq j \leq k - 1$.

LIS is a natural and well-studied problem, for more information regarding this problem, the reader is referred to the survey by Aldous and Diaconis [5]. It is well known that LIS is in the complexity class $\mathcal{NC}$ (e.g See [108]). LIS is also related to the distance to monotonicity problem (under the edit distance), see [34, 54]. There is a classical algorithm that

computes $LIS(x)$, called *Patience Sort* [5].

The Patience Sort algorithm naturally works in the streaming model, but requires $O(k \log n)$ bits of storage to compute $k = LIS(x)$ exactly, which is not efficient enough (e.g., in the worst case, $k = O(n)$. Even for inputs in which each permutation is equally likely, the expected length of the longest increasing subsequence is approximately $2\sqrt{n}$ [5]).

Gopalan, Jayram, Krauthgamer and Kumar [54] described a deterministic streaming algorithm (without prover) that uses space $O\left(\sqrt{n/\epsilon}\right)$ to compute a $(1-\epsilon)$-approximation of $LIS(x)$. They also showed that any randomized algorithm that computes $LIS(x)$ exactly in the streaming model requires linear space in the worst case. Gal and Gopalan [44] as well as Ergün and Jowhari [34] showed lower bounds of the order $\Omega\left(\sqrt{n/\epsilon}\right)$ for the space needed to deterministically compute a $(1-\epsilon)$-approximation of $LIS(x)$. Finally, Chakrabarti [19] argues that extending these lower bounds to the randomized case needs different techniques, so the best randomized lower bounds are still only logarithmic. For the randomized case, we note that it is wide open whether one can construct a streaming algorithm which uses space $o\left(\sqrt{n}\right)$ to compute a $(1-\epsilon)$-approximation of $LIS(x)$ for constant $\epsilon \in (0, 1)$.

We will show how to compute a $(1-\epsilon)$-approximation of $LIS(x)$ using polylog(n) verifier space in the presence of a prover in our model, i.e., with small communication overhead but large total communication.

6.3.2.1 Patience Sort

Patience Sort works as follows. The algorithm maintains an ordered set of sets (which we will call *stacks*) of elements. Stacks are named from 1 up to some number k, and should be imagined as being arranged from left to

right. In the beginning, we start with an empty set of stacks. As a new input x_i arrives, we find the leftmost stack such that the top (minimum) element of that stack is strictly larger than x_i, and put x_i on top of that stack. If no such stack exists, we start a new stack to the right of the existing stacks, and put x_i on top of that stack. We refer to [5] for the many interesting properties of this algorithm. Notice that the original version of Patience Sort only needs to maintain the top elements of all stacks used. We will refer to the number of the stack as arranged from left to right as the *name* of the stack. This is important, because in our algorithm we will not maintain all stacks but only a small subset of them, but we will keep their names intact.

It is easy to see that in the end the largest name of a stack (the rightmost stack) corresponds to $LIS(x)$. Further observations include that the sequence of elements on the tops of the stacks (the sequence of stack minima) is increasing, but not necessarily an increasing subsequence. Each stack viewed from the bottom to the top is a strictly decreasing sequence (and indeed a subsequence of x).

Like our algorithm for computing the median, our algorithm for LIS works in $\log n$ phases, in which the prover re-streams part of the original stream, but in this case under various permutations. The main ingredient here is the following dual characterization of $LIS(x)$ in the spirit of Dilworth's theorem [64].

Definition 6.3.6. A *descending chain* is a subsequence $x_{i_1} > \cdots > x_{i_l}$, where $1 \leq i_1 < \cdots < i_l \leq n$. For convenience we set $x_0 = m+1$ and $i_0 = 0$, where m is the size of the universe.

The *extension* of a descending chain is a sequence $y_1, \ldots, y_n$ such that $y_k = x_{i_j}$, where i_j is the largest index from $\{i_0, \cdots, i_l\}$ which is at most k.

Two descending chains $x_{i_1} > \cdots > x_{i_l}$ and $x_{j_1} > \cdots > x_{j_m}$ *cross*, if for their extensions y and z, there are indices $u < v$ such that $y_u < z_u$ and $y_v > z_v$ or vice versa.

Example 6.3.7. (**Descending chains that do not cross**)

Let $\sigma = (7, 2, 8, 1, 6, 5)$. Here $n = 6$ and $m = 8$. Let $\sigma_1 = (7, 2, 1)$ and $\sigma_2 = (8, 6, 5)$ be two descending chains of σ. The extension of σ_1 is $(7, 2, 2, 1, 1, 1)$ while the extension of σ_2 is $(9, 9, 8, 8, 6, 5)$. From Figure 6.3.1, it is easy to see that the two descending chains σ_1 and σ_2 do not cross.

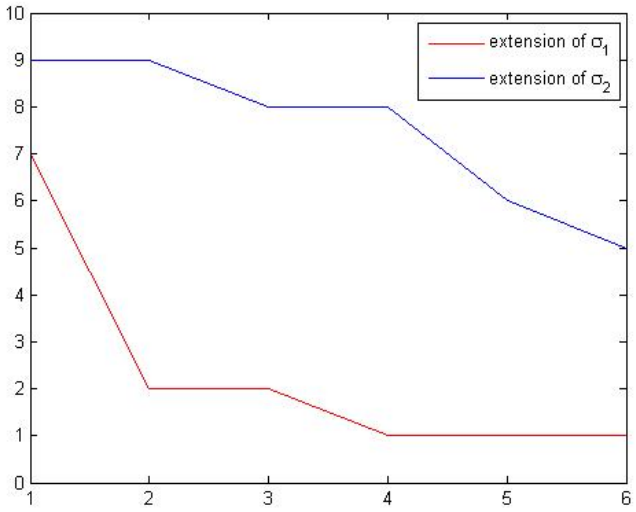


Figure 6.3.1: Descending chains that do not cross.

Example 6.3.8. (**Descending chains that cross**)

Let $\sigma = (7, 2, 8, 1, 6, 5)$. Here $n = 6$ and $m = 8$. Let $\sigma_3 = (7, 6, 5)$ and $\sigma_4 = (2, 1)$ be two descending chains of σ. The extension of σ_3 is $(7, 7, 7, 7, 6, 5)$ while the extension of σ_4 is $(9, 2, 2, 1, 1, 1)$. From Figure 6.3.2, it is easy to see that the two descending chains σ_3 and σ_4 cross.

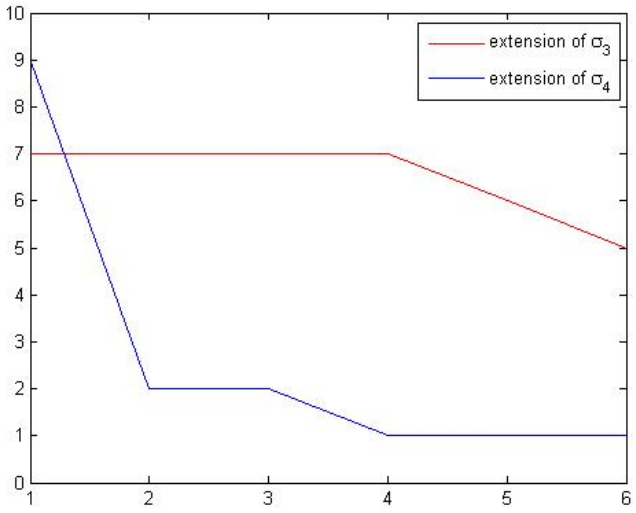


Figure 6.3.2: Descending chains that cross.

Lemma 6.3.9. *$LIS(x) = k$ if and only if there is a partition of $x = x_1, \ldots, x_n$ into k descending chains, such that no two of the chains cross and that the chains can be ordered so that the smallest elements of the chains form an increasing sequence.*

Proof. If $LIS(x) = k$, then Patience Sort will find a partition into exactly k descending chains (corresponding to the stacks maintained by the algorithm) whose smallest elements form an increasing sequence. These descending chains do not cross by construction: if two descending chains cross, then this means that an element that can fit on top of the left stack is placed on the stack further to the right at some point, which violates the algorithm.

Conversely, assume the minimum size of a partition with the above properties is k. The chains in the partitions are numbered 1 to k and we refer to lower numbered chains as being to the left of higher numbered chains.

First we observe that if we remove elements $x_l, \ldots, x_n$ from the chains, the property that the smallest elements of the chains form an increasing

sequence remains true. To see this, consider removing one element x_n. x_n must be the smallest element of some chain, because chains are subsequences. We remove the rightmost appearance of x_n if there are several elements equal to x_n which are smallest elements of chains. Clearly removing an element like this cannot make chains cross. Now suppose that the smallest elements of the resulting chains no longer form an increasing sequence. That means the smallest element of the chain to the right of the chain where we removed x_n is some x_j and $x_n < x_j < x_i$ where x_i is the new smallest element of the chain where we removed x_n. Let y denote the extension of the chain containing x_n and z the extension of the other chain.

Assume that $j < i < n$. In this case $y_j \geq y_i = x_i > x_j = z_j$. So assume that $i < j < n$. Then $y_j = y_i = x_i > x_j = z_j$. In both cases we have that $z_n = z_j = x_j > x_n = y_n$, so the chains cross, a contradiction.

We conclude that if we remove elements $x_l, \ldots, x_n$ from the chains, the smallest elements of the chains, left to right, still form an increasing sequence.

We have to show that $LIS(x) = k$. To show $\leq$, we simply observe that any partition into k descending chains is a witness for $LIS(x) \leq k$. For the other direction, take any partition into descending chains that do not cross and have increasing smallest elements. We construct an increasing subsequence of length k. Take the largest element of the rightmost chain, which is going to be the first (bottom) element of the chain. When this is x_j, we remove all elements $x_j, \ldots, x_n$. By the above argument we are left with $k-1$ chains that still have the desired properties. By induction we get an increasing subsequence of length $k-1$ that ends in the smallest element x_i of the rightmost chain (among $x_1, \ldots, x_{j-1}$). $x_i \leq x_j$, because among $x_1, \ldots, x_j$ the element x_j is the smallest element of the rightmost

chain. So we can find an increasing sequence of length k. □

The partition of x into k descending chains can easily be computed by modifying Patience Sort, so that it keeps all the elements of the stacks, instead of only the top elements. Note that this increases the space usage, but this work will be done by the prover.

6.3.2.2 Intuition for the Algorithm

The idea of our protocol is that the (honest) prover runs Patience Sort, and convinces the verifier via re-streaming of certain permutations of substreams that he did so correctly. Let us consider a scenario, where this verification is allowed to take place after the original input stream has ended (this is not allowed in our model and considered here only for motivation). While the prover executes the Patience Sort algorithm during the original stream, he announces the stack number of each new element to the verifier. The verifier computes the multiset fingerprint of the set of tuples i, x_i, l_i, where l_i is the announced stack number of element x_i arriving at time i. After the original stream has ended, the prover re-streams this set in two different ways, making it possible for the verifier to check that Patience Sort was executed correctly.

The first re-streaming consists of all the i, x_i, l_i ordered by the stack numbers and then (inside every stack) by the sequence number i, i.e., the verifier gets to see the sequence as partitioned into the descending chains. If any chain is not descending, he will reject the computation. The multiset fingerprint will allow him to check the re-streaming for correctness even though it is not in the original order. The verifier also tests that the minimum elements of all chains form an increasing sequence. For this, he just needs to remember the smallest element of the current chain.

With the second re-streaming we want to check that none of the chains cross. If two chains cross, then two neighboring chains cross. So for all neighboring pairs of chains, we need to check that they do not cross.

Definition 6.3.10. An *alignment* of two chains $x_{i_1}, \ldots, x_{i_l}$ and $x_{j_1}, \ldots, x_{j_k}$ is the sequence of all the elements of the two chains, sorted by their position in x, and accompanied by bits indicating whether elements belong to the first or the second sequence.

The prover will re-stream the alignments of all pairs of neighboring chains. The verifier can now easily check that those pairs do not cross: crossing means that an element is added to the right chain even if it would fit into the left chain. The verifier can also easily construct the fingerprint of the set of i, x_i, l_i for their first and second appearance in the re-streaming (all chains except the first and the last are re-streamed twice). These fingerprints can be checked against the multiset fingerprint the verifier knows.

In this way, two re-streams would allow us to check the correctness of the Patience Sort execution, and allow for an exact computation of $LIS(x)$.

6.3.2.3 The Algorithm

As in the algorithm for Median in Subsection 6.3.1, we partition $[n]$ into $\log n$ subsets of decreasing size, called the phases of the algorithm. Each phase is handled similarly to the above description (with the re-streaming taking place during the next phase). The main issue is how to connect the phases to check overall correctness. The first idea is to keep a fingerprint of the top of all stacks, i.e., the sequence visible to the sequential Patience Sort algorithm at all times. This allows to connect two phases, since all we need is to make sure that the tops of stacks are consistent across phases. However, the prover has to re-stream the k top elements to check consistency

(the verifier does not have enough memory to store them), and usually at some time the phases become too short to do that (e.g. if $LIS(x) = \sqrt{n}$ and the phase length gets to $o(\sqrt{n})$). Note that we could simply stop at that point, since we have already achieved some approximation. But in order to get a $(1-\epsilon)$-approximation, one would have to stop when the current estimate for $LIS(x)$ is at least $1/\epsilon$ times the current phase length. This would mean that the re-streaming used until now requires the communication overhead to be at least $1/\epsilon$, which is not desirable. Instead we reduce the number of stacks to $O(\log n/\epsilon)$ and continue, still achieving a good approximation, but leaving the communication overhead independent of ϵ. Intuitively we keep stacks whose name is a multiple of $\epsilon k/\log n$ if k is the current largest stack name. This gives us an approximation error of ϵ, since the reduction in the number of stacks happens only at the end of a phase, and there are only $\log n$ phases. The error analysis is completely analogous to the one given in [54].

Denote by P_ℓ the ℓ-th phase, i.e., $x_{n-n/2^{\ell-1}+1}, \ldots, x_{n-n/2^\ell}$. We now describe the algorithm in our usual model, i.e., without extra time for re-streaming at the end.

The algorithm keeps a variable k that contains the largest stack name used so far. It starts as $k = 0$. In phase P_1, the prover runs the Patience Sort algorithm, announcing for each element x_i, the number s_i of the stack he is putting the element on to the verifier. The verifier computes the multiset fingerprint of all the tuples i, x_i, s_i. We are using fingerprints with reliability $1-1/n^2$. The verifier increases k by one if necessary, and rejects, if the prover tries to increase k by more than 1.

In each phase P_ℓ for $\ell = 2, \ldots, \log n$ until $n/2^\ell < k$, the following happens. As before the prover will continue to run Patience Sort, and

communicate the stack numbers, and the verifier will create a (new) multiset fingerprint for the tuples i, x_i, s_i with $i \in P_\ell$. But furthermore the prover will use this phase to re-stream the permuted inputs from the previous phase. For $\ell \geq 2$, he re-streams the tuples i, x_i, s_i for $i \in P_{\ell-1}$ sorted by the third component (and for each value of the third component sorted by the first component). The verifier again takes the multiset fingerprint of this sequence, and eventually checks it against the one computed in the previous phase. If the fingerprints do not agree, he rejects and stops. Furthermore the verifier checks during this re-streaming that all stacks are indeed strictly decreasing sequences and that the minimal elements of all the stacks form an increasing sequence (if not he will reject). The verifier also computes the vector fingerprint of these minimal elements, which is used to connect two neighboring phases.

As described above, a second re-streaming of the elements from phase $\ell-1$ is needed to check that neighboring stacks do not cross. So, the prover re-streams for all pairs of stacks their alignment (as defined above). If the two stacks cross, the verifier rejects. To check consistency the verifier again computes the multiset fingerprints of the first and second occurrences of each of the stacks and checks these fingerprints against the one from $P_{\ell-1}$. Finally, the verifier takes the vector fingerprints of the two occurrences of a stack in the re-streaming (except for the first and last stack) and checks that they are the same. This finishes the phase.

For $\ell > 2$, we need to make sure that the re-streaming for consecutive phases fit together. Furthermore we do not want to re-stream the whole stacks, because we need the length of the re-streaming in a phase to be within a constant factor of the length of the phase. Note that the only relevant point here is that top (minimal elements) of all stacks at the end

of one phase coincides with the top of all stacks at the beginning of the next phase. Indeed the prover and verifier agree to discard all elements except the top of all stacks and re-streaming will only consider the elements that appeared in the previous phase.

Hence we modify the computations during a phase P_ℓ for $\ell \geq 3$ as follows. From phase $P_{\ell-1}$, we hold a vector fingerprint of the sequence of minimal elements of all the stacks at the end of $P_{\ell-2}$. Assume this sequence is $x_{i_1} \leq \cdots \leq x_{i_k}$. Then the re-streaming of phase P_ℓ concerns running Patience Sort on the sequence $x_{i_1}, \ldots, x_{i_k}$ followed by the inputs from phase $P_{\ell-1}$. These are $k + n/2^{\ell-1}$ elements. The verifier can now check that the first k elements are correct, and can compute the multiset fingerprint of the re-streaming of $P_{\ell-1}$, as well as check the correct execution of Patience Sort.

Once we reach the point where $n/2^\ell < k$, we cannot continue the above because the re-streaming of the top of the stacks is no longer possible with a small communication overhead. At this point we switch to approximation, similar to the deterministic streaming approximation algorithm from [54]. Say we have m stacks currently in use by the Patience Sort algorithm which is executed by the prover. These have top elements $x_{i_1}, \ldots, x_{i_m}$ (i.e., the decreasing chains in the stacks end in those elements). We now choose $\log n/\epsilon$ stacks from 1 up to m (evenly spaced) and discard the remaining stacks. This operation is referred to as cleanup. Note that stacks do not change their names during cleanup and that we will never remove the stack with the largest name (which is k).

Assume the set of stack names is $S = \{j_1, \ldots, j_m\}$ and that $m > 2\log n/\epsilon$ and that $k = j_m$ is the largest stack number. The prover then sets S to $\{\lceil \epsilon/\log n \cdot k \rceil, \lceil 2\epsilon/\log n \cdot k \rceil, \ldots, k\}$, discarding the remaining stacks.

The first time this happens S starts as $\{1, \ldots, k\}$, and the prover has to re-stream the sequence of top elements of all the stacks so that the verifier can compute the new sequence of top elements of the reduced set of stacks (and its fingerprint). This is still possible within constant overhead. The same needs to be done after any cleanup operation, but note that the number of stacks in phase ℓ can now never exceed $O\left(2 \log n/\epsilon + n/2^{\ell}\right)$.

We claim that the cleanup operations each introduce error at most $\epsilon/\log n$. Also note, that this reduction step takes place every time $n/2^{\ell}$ exceeds the current k by a constant factor, because k may start to grow again after a cleanup, but that this happens at most once per phase. There are $\log n$ phases and hence at most that many cleanup operations and the total error is bounded by ϵ.

Lemma 6.3.11. *The total error introduced by all the cleaning up steps is at most ϵ.*

Proof. Each reduction step removes some of the stacks maintained by the Patience Sort algorithm. Consider sequences of some length l that ends in an element a from the universe $U = \{1, \ldots, n^2\}$. We denote by $P(l)$ the smallest element a of U such that there is an increasing subsequence of length l that ends in a. Note that Patience Sort computes $P(l)$ as the top (smallest) element of stack l.

For simplicity assume that a cleanup operation happens at the end of every phase. If the first few phases do not include a cleanup, then the error can only decrease. Denote by k_{ℓ} the maximum stack name after phase P_{ℓ}. We define $P'(l)$ to be $P(l)$ if stack l is in S, i.e., has not been discarded, and otherwise $P'(l) = P(j)$ for the smallest j such that $j > l$ and $j \in S$. Naturally, $P'(l) \geq P(l)$. Furthermore denote by $P_t(l)$ the smallest element a such that there is an increasing subsequence of length l that ends in a

among $x_1, \ldots, x_t$. In a similar manner, define $P'_t(l)$ to be $P_t(l)$ if stack l is in S, and otherwise $P'_t(l) = P_t(j)$ for the smallest j such that $j > l$ and $j \in S$.

We will use the following claim, which basically states that at any time $t \leq \log n$, a slightly shorter sequence ending in a or a lesser element still exists. This claim is proved exactly like Lemma 2.2 in [54].

Claim 6.3.12.

$$P'_t(l - (t-1)\epsilon k_t / \log n) \leq P_t(l).$$

Note that there are at most $\log n$ phases and so in the end $P'((l-\epsilon)k) \leq P(l)$, where k is the final maximum name of any stack. This directly shows that in the end the error is at most ϵ. □

Finally, at some point $P_\ell < 10 \log n/\epsilon$. At this point the prover still sends the sequence of top elements from all stacks, but then prover and verifier stop communicating. The prover starts with the given sequence (which fits into his memory) and runs Patience Sort himself with the remaining input.

We can now state the overall result of this section.

Theorem 6.3.13. *There is a streaming algorithm with a prover for computing a $(1-\epsilon)$-approximation of $LIS(x)$ using constant communication overhead (independent of n and ϵ), space $O(\log^2 n/\epsilon)$, and success probability $1 - 1/n$. The computation overhead of the prover is* polylog n.

We note that the only source of error is a failure of fingerprinting, and that the underlying approximation is guaranteed if no fingerprinting operation fails.

6.3.3 FULL RANK

In this subsection, we consider the problem of determining whether an $n \times n$ integer matrix containing integers of size n (for concreteness) has full rank over the reals (the matrix is streamed row by row, each entry being one symbol on the stream).

Clarkson and Woodruff [24] studied the rank decision problem in the standard streaming model for turnstile updates. They showed that any streaming algorithm which succeeds with constant probability needs $\Omega(k^2)$ bits of space to determine if the rank of $A \in \mathcal{M}_n(\mathbf{Z})$ is at least k. They also gave a upper bound of $O\left(k^2 \log \frac{n}{\delta}\right)$ bits of space needed for the failure probability to be at most δ. It follows from a reduction we describe in Section 6.4 that $\Omega(n^2)$ bits of space are needed to determine if $A \in \mathcal{M}_n(\mathbf{Z})$ is full rank in the time-series model.

We study the problem in our model. One issue we run into immediately is that standard factorizations of matrices that reveal the rank (e.g., the Gauss-Jordan elimination) require exponential size numbers as entries of the factor matrices. Given $B \in \mathcal{M}_{m,n}(\mathbf{Z})$, let $||B||_\infty$ to be the maximum absolute value of any entry of B. Fang and Havas [36] showed that the Gauss-Jordan elimination over $\mathbf{Z}$ has worst-case exponential space complexity, i.e. the entries of the factor matrices can be bounded exponentially in $||A||_\infty$, where the Gauss-Jordan elimination is performed over $\mathbf{Z}$ on $A \in \mathcal{M}_{m,n}(\mathbf{Z})$. This phenomenon is known as "entry explosion". Havas et al. [57] claim that in practice, "entry explosion" often occurs when Gauss-Jordan elimination is performed over $\mathbf{Z}$.

That means even if an honest prover tries to convince us by means of such a factorization, he cannot even tell us what the entries are in an efficient manner! However, probabilistic techniques come to our rescue,

and it is possible to simply choose a large enough prime p, and test if the matrix has full rank over the field $\mathbf{F}_p$. On the other hand one has to be careful with standard concepts like orthogonality over a finite field, as we shall see below.

6.3.3.1 Verifying Matrix Multiplication in the Streaming Model

Suppose we are given $A \in \mathcal{M}_{m,n}(\mathbf{F}_p)$ and we wish to verify if $A = BC$, where $B \in \mathcal{M}_m(\mathbf{F}_p)$ and $C \in \mathcal{M}_{m,n}(\mathbf{F}_p)$, with $m \leq n$ and $n^2 \leq p \leq 2n^2$ is a prime. In the streaming model, we cannot store any of these matrices and compute the matrix multiplication and do the equality test. Even if the entries of B and C were streamed for us to verify the matrix multiplication, we would need to stream each row of B n times and each column of C n times.

Freivalds [42] introduced a simple randomized algorithm to verify matrix multiplication using $O(n^2)$ multiplications and addition, using n bits of randomness. But the verifier cannot store this n bits of randomness. Kimbrel and Sinha [68] improved Freivalds' algorithm so that we only need to use $O(\log n)$ random bits. Algorithm 6.3.1 illustrates how to verify $A = BC$ in the streaming model, where we are only allowed to stream A, B and C once and the verifier's space is $O(\log n)$.

1. Choose $r_* \in \mathbf{F}_p$ and $s_* \in \mathbf{F}_p$ uniformly at random. Let $r = (1, r_*^1, r_*^2, \cdots, r_*^{n-1})^T \in \mathbf{F}_p^n$ and $s = (1, s_*^1, s_*^2, \cdots, s_*^{m-1}) \in \mathbf{F}_p^m$.

2. Compute sAr and $sBCr$. Accept if and only if these two quantities are the same.

Algorithm 6.3.1: Verifying $A = BC$ in the Streaming Model.

Algorithm 6.3.1 uses $O(\log n)$ random bits. It is known that over any integral domain, every Vandermonde matrix of n distinct nonzero elements is nonsingular (See [78] for details). If $A \neq BC$, then there are only $n-1$ vectors r (out of $O(n^2)$ vectors) such that $Ar = BCr$. Note that $s(Ar)$ and $s(BCr)$ are the vector fingerprints of Ar and BCr respectively. The algorithm can only err when $A \neq BC$ and either $Ar = BCr$ or $Ar \neq BCr$ but $s(Ar) = s(BCr)$. This happens with with probability at most $O(1/n)$.

Since $sAr = \sum_{i=1}^{m} \sum_{j=1}^{n} s^{i-1} r^{i-1} A_{ij}$, the verifier can compute this quantity using $O(\log n)$ space if the prover re-streams matrix A once. Using the vector fingerprint[4] given in Lemma 3.1.4, the verifier can check if the matrix A which appeared on the stream before is the same as the one the prover is providing. If we let $x = Cr$ and $y^T = sB$, then the inner product of x and y is $(sB)(Cr)$. For $1 \leq k \leq m$, since $y_k = \sum_{i=1}^{m} s^{i-1} B_{ik}$ and $x_k = \sum_{j=1}^{n} r^{j-1} C_{kj}$, the verifier can compute $(sB)(Cr)$ if the prover streams the k-th column of B followed by the k-th row of C for all $k = 1, \cdots, m$.

6.3.3.2 Hermite Normal Form of an Integer Matrix

Given an integer matrix, we would like to obtain a canonical representation of the matrix so that we can calculate important properties of the matrix, like its rank and determinant easily. One standard technique in linear algebra is to transform the matrix into row echelon form using Gaussian elimination. But this involves divisions over $\mathbf{Z}$, and the resulting reduced row echelon matrix need not be an integer matrix. As such, we consider the Hermite normal form of a matrix.

Definition 6.3.14. Let $A \in \mathcal{M}_{m,n}(\mathbf{Z})$ with r nonzero rows. A is said to be in row Hermite normal form (HNF) if the following conditions are satisfied:

[4] We view an $m \times n$ matrix as a vector of dimension mn.

1. The first r rows of A are nonzero.

2. For $1 \leq i \leq r$, let A_{i,j_i} be the first nonzero entry in the i-th row of A. Then $j_1 < j_2 < \cdots < j_r$.

3. $A_{i,j_i} > 0$ for all $1 \leq i \leq r$.

4. If $1 \leq k < i \leq r$, then $0 \leq A_{k,j_i} < A_{i,j_i}$.

The entries A_{i,j_i} will be called the corner entries of A. Every integer matrix can be reduced to its Hermite normal form via row operations. See [98] for details. More precisely, we have the following.

Lemma 6.3.15. *Let $A \in \mathcal{M}_{m,n}(\mathbf{Z})$. Then there exists an unique matrix $H \in \mathcal{M}_{m,n}(\mathbf{Z})$ which is in row Hermite normal form and $U \in \mathbf{GL}_m(\mathbf{Z})$ such that $A = UH$.*

The matrix factorization in Lemma 6.3.15 is rank revealing. But the entries in H and U can be exponential in the size of the entries of A. For details of "entry explosion" in the computation of Hermite normal form, the reader is referred to [31] and the references therein. In our model, we may not be able to communicate the entries of H and U. Thus we consider reducing the matrix modulo d to its Hermite normal form.

For any integer matrix A, we let $\bar{A}$ be the result of reducing all its entries modulo d.

Definition 6.3.16. A is said to be in row Hermite normal form modulo d if the following conditions are satisfied:

1. $A = \bar{A}$.

2. A is in row Hermite normal form over $\mathbf{Z}$.

3. The corner entries of A divide d.

Similar to Lemma 6.3.15, given any $A \in \mathcal{M}_{m,n}(\mathbf{Z})$, there exist an unique matrix $H \in \mathcal{M}_{m,n}(\mathbf{Z_d})$ which is in row Hermite normal form modulo d and $U \in \mathbf{GL}_m(\mathbf{Z_d})$ such that $A = UH \pmod{d}$. The number of nonzero rows of H is called the d-rank of A.

For $A \in \mathcal{M}_{m,n}(\mathbf{F}_p)$ where $m \leq n$, by considering the Hermite normal form factorization over $\mathbf{F}_p$, we have

$$A = U \begin{pmatrix} G \\ \mathbf{O} \end{pmatrix} \tag{6.1}$$

where $rank(A) = r$ over $\mathbf{F}_p$, $U \in \mathbf{GL}_m(\mathbf{F}_p)$, $G \in \mathcal{M}_{r,n}(\mathbf{F}_p)$ is in row Hermite normal form modulo p and $\mathbf{O}$ is the all zero matrix of size $m - r$ by n. We define $\mathcal{C}_A$ to be the row space of A. Since A and $\begin{pmatrix} G & \mathbf{O} \end{pmatrix}^T$ are row equivalent, the rows of G is a basis for $\mathcal{C}_A$. It is known that the subspace

$$\mathcal{C}_A^{\perp} := \{\ y \in \mathbf{F}_p^n \mid \langle x, y \rangle = 0 \quad \forall\ x \in C_A\ \}$$

has dimension $n - r$. We describe how to obtain a basis of $\mathcal{C}_A^{\perp}$ efficiently from the HNF factorization of A given in (6.1). Since the corner entries of G divide p, all of them are 1. As such, there exists a permutation matrix $P \in \mathcal{M}_n(\mathbf{F}_p)$ such that $GP = \begin{pmatrix} I_r & D \end{pmatrix}$ where $D \in \mathcal{M}_{r,(n-r)}(\mathbf{F}_p)$. Consider the matrix

$$B := \begin{pmatrix} -D \\ I_{n-r} \end{pmatrix} \in \mathcal{M}_{n,(n-r)}(\mathbf{F}_p).$$

Since the rank of B can be at most $n - r$ and the last $n - r$ rows of B are linearly independent, it follows that the rank of B is $n - r$. Observe that $GPB = 0$. Let $C := PB$ which is full rank[5]. Every row of G is orthogonal

[5]A matrix $A \in \mathcal{M}_{m,n}(\mathbf{F})$ is said to be full rank if $rank(A) = \min(m, n)$.

to each of the columns of C. As such, the columns of C forms a basis for $\mathcal{C}_A^{\perp}$. In the literature of coding theory [104], G and C^T are called the generator and parity check matrix of the code $\mathcal{C}_A$ respectively.

In view of this, we can first permute the columns of A and then consider its HNF factorization.

Definition 6.3.17. Let $A \in \mathcal{M}_{m,n}(\mathbf{F}_p)$ where $m \leq n$ and $rank(A) = r$ over $\mathbf{F}_p$. There exist a $U \in \mathbf{GL}_m(\mathbf{F}_p)$ and a $n \times n$ permutation matrix P such that

$$AP = U \begin{pmatrix} I_r & D \\ O & O \end{pmatrix}$$

where $D \in \mathcal{M}_{r,(n-r)}(\mathbf{F}_p)$. We call this the HNF factorization of A in standard form.

6.3.3.3 The Result

Given a square matrix $A \in \mathcal{M}_n(\mathbf{Z})$ which is streamed row-wise, where $|a_{ij}| \leq n$, we wish to determine if A has full rank over the reals. Without loss of generality, we can assume that A has no zero rows. In Section 6.4, we show a reduction from INDEX which implies that this problem is hard for streaming algorithms in the standard streaming model, i.e., such algorithms would need space $\Omega(n^2)$. The problem is in $\mathcal{NC}$, so by Theorem 4.1.3 there is an efficient streaming algorithm with a prover, but our algorithm is much simpler and not based on arithmetization techniques and does not have an extra verification phase after the end of the stream. Our algorithm does not have polylogarithmic time overhead for the prover, but finding an algorithm with that much efficiency would imply a quasilinear time algorithm for the problem, which is a major open problem. The best randomized algorithm that we know currently to compute the rank

of a matrix A over a field requires $\widetilde{O}(|A| + r^{\omega})$ field operations [23], where $r = rank(A)$, $|A|$ denotes the number of nonzero entries of A and $\omega < 2.373$ is the matrix multiplication exponent [27, 79, 105]. We remind the reader that the matrix multiplication exponent ω is defined as the minimum value such that two $n \times n$ matrices over a field can be multiplied using $O(n^{\omega+\epsilon})$ arithmetic operations for any $\epsilon > 0$.

However, for the FULL RANK protocol in our model, the time spent by the prover is well spread out over the duration of the stream, and his work consists mostly of computing various matrix factorizations. We state the main theorem first before describing the protocol.

Theorem 6.3.18. *Given a square matrix $A \in \mathcal{M}_n(\mathbf{Z})$ which is streamed row-wise, where $|a_{ij}| \leq n$, there is a streaming algorithm with a prover which can decide if A has full rank over the reals where the verifier uses $O(\log n)$ storage and the total communication overhead is constant with probability of failure $O\left(\frac{\log n}{n}\right)$.*

Our general approach is to compute the rank over a random prime p instead of over the reals, in order to avoid blowing up the size of matrix entries, an often encountered problem with computing e.g. the QR factorization.

The following argument shows that if we choose a random prime p from a set of primes whose size is polynomial in n, the p-rank of A will be the same as the rank of A over the reals with high probability.

Let B be any $r \times r$ square submatrix of A. By the Hadamard bound for determinant [59], we have $|\det(B)| \leq (\sqrt{r}\, n)^r$. Hence, the number of distinct prime factors for $\det(B)$ is at most $r(\log r + \log n) \leq 2n \log n$. Let $b = 2s \ln s$, where $s = 2n^2 \log n$. It is known that $\pi(n) > n/\ln n$ for $n \geq 17$, where $\pi(n)$ is the prime counting function. Since $\pi(b) > s$, if we choose a

prime p randomly between $[1, b]$,

$$\mathbf{Pr}\left[\det(B) \equiv 0 \pmod{p}\right] \leq \frac{2n \log n}{2n^2 \log n} = \frac{1}{n}.$$

If the rank of A is k, there exist $\tilde{A}$, a $k \times k$ submatrix of A with nonzero determinant, and all $(k+1)\times(k+1)$ submatrices of A have zero determinant. If we choose p randomly between $[1, b]$, the probability that $\det(\tilde{A}) \not\equiv 0 \pmod{p}$ is at least $1 - 1/n$ and all $(k+1) \times (k+1)$ submatrices of A have zero determinant. As a result, with probability at least $1 - 1/n$, the p-rank of A is exactly k.

We introduce some notations first which will help us analyze the protocol. We let a_i to be the i-th row of A, $A^{(i)} := \text{span}\{a_1, \cdots, a_i\}$ and $W^{(i)} := (A^{(i)})^{\perp}$.

6.3.3.4 Protocol

The verifier and prover choose a prime p randomly between $[1, b]$ before seeing the stream, where $b = \widetilde{O}(n^2)$. We will assume that the p-rank of A is equal to the rank of A over the reals. Before the entries are being streamed, the verifier chooses a prime $n^3 < q < 2n^3$ and then chooses r uniformly at random from $\mathbf{F}_q$ (and keeps it private from $\mathcal{P}$). In phase 1, the verifier computes $FP_q(r, A_1)$, the fingerprint of A_1, which is a submatrix of A consisting of the first $n/2$ rows of A.

For phase 2, it starts when $a_{n/2+1,1}$ is streamed and ends when $a_{3n/4,n}$ is streamed. In this phase, the verifier computes the fingerprint of A_2, which is a submatrix of A consisting of the next $n/4$ rows. In this phase, $\mathcal{P}$ informs $\mathcal{V}$ if A_1 is full rank. If A_1 is full rank, $\mathcal{P}$ and $\mathcal{V}$ verify the matrix multiplication

$$A_1 P_1 = U_1 \begin{pmatrix} I_{n/2} & H_1 \end{pmatrix}, \tag{6.2}$$

which is the HNF factorization of A_1 in standard form. For the matrix in standard HNF, $\mathcal{P}$ should only stream the entries of H_1 and for each row of P_1, the prover should stream j such that $p_{i,j} = 1$. $\mathcal{V}$ should compute the fingerprints of U_1, H_1 and P_1 as they are streamed during the verification of the matrix multiplication. $\mathcal{P}$ should also convince $\mathcal{V}$ that U_1 is invertible by verifying the matrix multiplication $U_1 U_1^{-1} = I$. To check if P_1 is indeed a permutation matrix, suppose that the prover claims that $\mathcal{S} := \{j_1, \cdots, j_n\}$ satisfies $p_{i,j_i} = 1$ for $1 \leq i \leq n$. $\mathcal{V}$ needs to check if $\mathcal{S} = [n]$, which can be checked by using the multiset fingerprint in Lemma 3.1.3. $\mathcal{P}$ and $\mathcal{V}$ will also verify the matrix multiplication $W_1 := P_1 \begin{pmatrix} -H_1 \\ I_{n/2} \end{pmatrix}$, where the column space of W_1 is $W^{(n/2)}$. The verifier will also have to compute the fingerprint of W_1. They proceed to phase 3 if all fingerprints agree and the verifier accepts all the matrix multiplication verifications if A_1 is full rank.

If A_1 is not full rank, $\mathcal{P}$ and $\mathcal{V}$ will verify the matrix multiplication $xA_1 = 0$, where $0 \neq x \in \mathbf{F}_p^{n/2}$. In this case, the protocol terminates and $\mathcal{V}$ accepts only if the matrix multiplication verification passes.

Also during phase 2, for each $1 \leq i \leq n/4$, $\mathcal{P}$ determines if $a_{n/2+i} \in A^{(n/2+i-1)}$. If there exists an index $1 \leq i \leq n/4$ such that $a_{n/2+i} \in A^{(n/2+i-1)}$, $\mathcal{P}$ will inform $\mathcal{V}$ to stop. In this case, there exists $d \in \mathbf{F}_p^{i-1}$ such that $x := a_{n/2+i} + \sum_{j=1}^{i-1} d_j \, a_{n/2+j} \in A^{(n/2)}$. In this case, $\mathcal{P}$ and $\mathcal{V}$ will verify the matrix multiplication

$$\begin{pmatrix} d_1 & \cdots & d_{i-1} & 1 \end{pmatrix} \begin{pmatrix} a_{n/2+1} \\ \vdots \\ a_{n/2+i-1} \\ a_{n/2+i} \end{pmatrix} = x. \tag{6.3}$$

$\mathcal{V}$ will keep the fingerprint of x as they verify 6.3. Now, they will verify $xW_1 = 0$ and the protocol will terminate with $\mathcal{V}$ convinced that A is not full rank.

Otherwise, we have $a_{n/2+i} \notin A^{(n/2+i-1)}$ for all $1 \leq i \leq n/4$. Then for each such i, there exists a $z_i \in W^{(n/2+i-1)}$ such that $\langle a_{n/2+i}, z_i \rangle \neq 0$. The prover computes all such z_i for $i = 1, \cdots, n/4$.

Phase 3 starts when $a_{3n/4+1,1}$ is streamed and ends when $a_{7n/8,n}$ is streamed. $\mathcal{V}$ will compute the fingerprint of A_3, which is a submatrix of A consisting of the next $n/8$ rows. In this phase, $\mathcal{P}$ will convince $\mathcal{V}$ that the first $3n/4$ rows of A are linearly independent by verifying the following matrix multiplications:

(i) $A_2 [\ z_1 \cdots z_{n/4}\ ] = L_2$ for some $L_2 \in \mathbf{GL}_{n/4}(\mathbf{F}_p)$ which is lower triangular.

(ii) $W_1 C_1 = [\ z_1 \cdots z_{n/4}\ ]$ for some $C_1 \in \mathcal{M}_{n/2,n/4}(\mathbf{F}_p)$.

(iii) $A_2 W_2 = 0$, where $W_2 \in \mathcal{M}_{n,n/4}(\mathbf{Z})$ whose column space is $W^{(3n/4)}$. $\mathcal{P}$ should provide W_2 in HNF so that $\mathcal{V}$ can easily check that W_2 is full rank.

(iv) $W_1 C_2 = W_2$ for some $C_2 \in \mathcal{M}_{n/2,n/4}(\mathbf{F}_p)$.

Subroutine 6.3.2: To verify if the first $3n/4$ rows of A are linearly independent and the correctness of the dual space of $A^{(3n/4)}$.

All the four steps can be done in parallel with $\mathcal{V}$ keeping the fingerprints of $[\ z_1 \cdots z_{n/4}\ ]$ and W_2 to ensure that the right matrices are streamed in the different messages. They proceed as such in $O(\log n)$ phases to determine if A is full rank. Note that for the last phase of the matrix rank protocol, if we would continue just like before, we would have to stream back a vector of length n, which is not allowed in our model. The last phase should be performed differently. As both the prover and the verifier have seen

the second last row, the prover now knows the vector v which is the dual space of the first $n-1$ rows of A. The prover can convince the verifier that the first $n-1$ rows of A are linearly independent and that v indeed generates the dual space of the first $n-1$ rows by using a similar idea as in Subroutine 6.3.2. This verification is done when the last row of A is being streamed. Namely we can verify the correctness of the one-dimensional subspace (which is spanned by v) which is the dual to the first $n-1$ rows of A during the process when the last row of A is being streamed. If A is full rank, a_n will not belong to the span of the first $n-1$ rows of A, and hence $\langle a_n, v\rangle$ is not equal to zero. On the other hand, if a_n belongs to the span of the first $n-1$ rows of A, then $\langle a_n, v\rangle = 0$. The verifier can check this inner product in a streaming manner when a_n is being streamed. The prover needs to provide the vector v for the verifier to compute the inner product. So, when a_n is being streamed, the verifier should verify the correctness of the one-dimensional dual subspace and compute the inner product $\langle a_n, v\rangle$ as well. This ensures that our protocol has no additional verification phase after the end of the stream.

Note that once the last row of A_2 is seen, the prover knows the dual space of the first $3n/4$ rows, whose column space is W_2. The correctness of W_2 is checked in the next phase when A_3 is being streamed. This is the same in the final phase where we check the correctness of the one-dimensional dual space when a_n is being streamed. Now when the rows of A_3 are being streamed, the prover needs to compute some z_i for each row of A_3 and the correctness of these z_i is verified in phase 4. But when a_n is being streamed, we do not need to wait for the stream to end to verify that the inner product of $\langle a_n, z\rangle$ is not zero. This is because the vector z in this case can be replaced by v, where v generates the dual space of the first

$n-1$ rows of A. This is the main difference in the final phase as compared to the previous phases.

6.3.3.5 Proof of Completeness and Soundness of Protocol

Firstly, if A_1 is full rank, the matrix factorization (6.2) is a witness that A_1 is indeed full rank and if A_1 is not full rank, such a factorization does not exist. Now consider the case where $B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ is full rank. If the prover is honest, he can construct $\{z_i\}_{i=1}^{n/4}$ such that the diagonal entries of L_2 is nonzero. For $i < j$, since $a_{n/2+i} \in A^{(n/2+j-1)}$, it follows that L_2 is lower triangular. Moreover, span$\{ z_1 \cdots z_{n/4} \} \subseteq W^{(n/2)}$, which is established in part (ii) of Subroutine 6.3.2.

Now, we argue that if B is not full rank, $\mathcal{P}$ cannot satisfy both (i) and (ii) of Subroutine 6.3.2 simultaneously. Let $a_{n/2+i}$ be such that $a_{n/2+i} \in A^{(n/2+i-1)}$. By means of contradiction, suppose $\mathcal{P}$ can choose a $x \in \mathbf{F}_p^n$ such that $\langle a_{n/2+i}, x \rangle \neq 0$. Such a x belongs to $W^{(n/2)}$ and has to satisfy $\langle a_{n/2+j}, x \rangle = 0$ for all $j = 1, \cdots, i-1$. This means $x \in W^{(n/2+i-1)}$, which contradicts $\langle a_{n/2+i}, x \rangle \neq 0$.

If we know that rank of B is $3n/4$, then any $n/4$ dimensional subspace $V \subseteq \mathbf{F}_p^n$ which satisfy the requirements that the inner product with any row of A_2 and any vector in V is zero and that $V \subseteq W^{(n/2)}$ must be the dual code of B. Thus W_2 is the correct representation of the dual code of B.

On the other hand, if B is not full rank, it is clear that the honest prover can convince the verifier of this fact.

Inductively, it is clear that in phase 4, by streaming A_3 and W_2, they can determine if the rows of $\{A_i\}_{i=1}^3$ are linearly independent and they can construct W_3, the dual code generated by the rows of $\{A_i\}_{i=1}^3$. We do not

need to stream the rows of A_1 and A_2 in this phase.

In each phase, we are using a constant number of invocations of Freivalds' technique to verify matrix multiplications. The protocol will err if either Freivalds' technique errs or when $\mathcal{P}$ streams an incorrect matrix whose fingerprint is the same as the original matrix. By the union bound, the probability of failure is $O\left(\frac{\log n}{n}\right)$.

6.3.3.6 Analysis of Space and Communication

In each phase, $\mathcal{V}$ needs to store a constant number of fingerprints and $O(\log n)$ space to keep track of the rows of the matrix being streamed and to generate the random bits needed for the matrix multiplication verification. Hence $\mathcal{V}$ needs $O(\log n)$ space in this protocol.

As we proceed to the next phase in the protocol, the size of the sub-matrix of A that we re-stream gets halved, and the dimension of the dual code also gets halved. Hence, the communication overhead is a constant. Also, we note that as the last row of A is being streamed, the verifier will be checking the correctness of the 1-dimensional space, which is the dual space of the first $n-1$ rows of A. As he checks the correctness of this 1-dimensional subspace, say spanned by v, he should also compute the inner product of the last row of A with v. This ensures that our protocol is fully online.

6.3.3.7 Application: Perfect Matching

We show that our FULL RANK protocol can be used to determine whether a graph has a perfect matching (See Definition 6.3.4) in our model. The decision problem of deciding whether a given graph G contains a perfect matching is known to be in the complexity class $\mathcal{RNC}$ but it is still

open whether it belongs to $\mathcal{NC}$ or not. So if one resorts to the GKR protocol to determine if a given graph G has a perfect matching, it is not known how to do it with polylog(m, n) cost and using polylog(m, n) rounds of interaction.

We show that our FULL RANK protocol can be used to determine if a graph $G = (V, E)$ with even vertices has a perfect matching in our model, where the rows of A_G, the adjacency matrix of G are being streamed. Let $|V| = n$, where n is even. We will make use of Tutte's Theorem [103] to devise a protocol for the perfect matching problem.

Theorem 6.3.19. *(Tutte's Theorem)*

Let A be a $n \times n$ Tutte matrix of indeterminates obtained from $G = (V, E)$ as follows: a distinct indeterminate x_{ij} is associated with the edge (v_i, v_j) where $i < j$, and the corresponding matrix entries are given by

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (v_i, v_j) \in E \text{ and } i < j, \\ -x_{ji} & \text{if } (v_i, v_j) \in E \text{ and } i > j, \\ 0 & \text{if } (v_i, v_j) \notin E. \end{cases}$$

Then G has a perfect matching if and only if $det(A)$ is not identically zero.

Since

$$det(A) = \sum_{\pi \in \mathcal{S}_n} sgn(\pi) \prod_{i=1}^{n} A_{i,\pi(i)},$$

where $\mathcal{S}_n$ is the symmetric group of permutations of size n and $sgn(\pi)$ is the sign of the permutation π, it follows that $det(A)$ is a polynomial of degree at most n. Let $\widetilde{A}$ be the matrix such that we replace each indeterminate x_{ij} for $i < j$ of A with a random integer $r_{ij} \in \{1, \cdots, O(n^2)\}$ chosen uniformly at random. Then with probability at least $1 - 1/n$, if G has a perfect matching, then $det(\widetilde{A}) \neq 0$ by the Schwartz-Zippel lemma (See Appendix A.1). On

the other hand, if G does not have a perfect matching, then $det(\widetilde{A}) = 0$ for any choice of the random r_{ij}. As a result, by determining if $\widetilde{A}$ has full rank over $\mathbf{R}$, we can decide with high probability if G has a perfect matching.

We note the following simple observation. Define matrix $C \in \mathcal{M}_n(\mathbf{R})$ such that $|c_{ij}| \leq m$. Suppose C is streamed row-wise. By using the vector fingerprinting technique (See (3.1)), it is easy to see that the verifier can check if C has the following structure: $c_{ij} = -c_{ji}$ for all $i \neq j$ and $c_{ii} = 0$ for all $i = 1, \cdots, n$. The space needed for the verifier is $O(\log m + \log n - \log \delta)$, where δ is the error of the protocol.

We now describe the perfect matching protocol. When the first row of A_G is streamed, for every non-zero entry $a_{1,i}$, the verifier chooses a random integer $r_{1,i}$ uniformly at random from the set $\{1, \cdots, O(n^2)\}$ and redefines $a_{1,i} := r_{1,i}$. The verifier sends the value of $r_{1,i}$ to the prover after the symbol $a_{1,i}$ appears on the stream. When the i^{th} row of A_G is streamed, for all $j < i$ such that $a_{i,j} \neq 0$, the prover will send $r_{j,i}$ (this was chosen by the verifier when $a_{j,i}$ was being streamed previously) and the verifier will redefine $a_{i,j} = -r_{j,i}$. For all $j > i$ such that $a_{i,j} \neq 0$, the verifier chooses a random number $r_{i,j}$ uniformly at random from the set $\{1, \cdots, O(n^2)\}$ and redefines $a_{i,j} := r_{i,j}$. The verifier sends the value of $r_{i,j}$ to the prover after the symbol $a_{i,j}$ appears on the stream. From the discussion in the previous paragraph, the verifier can check if the value of $r_{j,i}$ provided by the prover is correct whenever $j < i$. By using our FULL RANK protocol in Theorem 6.3.18, we have a protocol which decides if G has a perfect matching.

Theorem 6.3.20. *Let $G = (V, E)$ be a graph where $|V| = n$ is even. Let A_G be the adjacency matrix of G which is streamed row-wise. There is a streaming algorithm with a prover which can decide if G has a perfect*

matching where the verifier uses $O(\log n)$ storage and the total communication overhead is constant with probability of failure $O\left(\frac{\log n}{n}\right)$.

By relaxing the total communication restriction, we managed to find an algorithm for a problem that is not known to be in $\mathcal{NC}$ while maintaining the full online nature of streaming.

6.4 A Lower Bound on the Number of Rounds

The streaming algorithms with a prover in [29,30,74] are based on techniques from interactive proofs in complexity theory. All of these streaming interactive protocols (SIP) are based on the famous sum check protocol of Lund et al. [84]. A careful inspection of SIP designed from the sum check protocol reveals that they hold in the Merlin-Arthur streaming model. They share the feature that after the stream has ended, a short ($O(\log m)$ round for some problems, see Section 4.3) verification phase takes place between the prover and verifier, i.e., the computation is not fully online in that sense that the answer can be given right away at the end of the stream. More importantly, the prover has to work very hard during this phase, computing certain polynomials depending on the previously hidden random bits of the verifier. This computation takes at least linear time in n, and is hence exponentially longer than the communication taking place during that phase. Is this additional phase really necessary? In Section 6.3, we have shown that for certain problems, the additional verification phase can be removed and the prover's computation overhead can be reduced to polynomial.

As we have seen in Section 3.4, there is an exponential gap between

streaming protocols in the Merlin-Arthur and IP models. In this section, we show that for any problem that is as hard as the INDEX function in a very strong sense defined below, any SIP in the Merlin-Arthur streaming model requires at least $\Omega(\log n/\log\log n)$ rounds of interaction between the prover and verifier after the end of the stream, otherwise either the communication complexity during that last phase increases to above polylog(n) or the space complexity of the verifier increases to above polylog(n). We stress that this only holds for SIP in the Merlin-Arthur streaming model, where the messages from the verifier to the prover are random challenges that neither depend on the stream nor the previous messages from the prover. Hence the extra verification phase is an inherent feature of Merlin-Arthur streaming protocols solving such problems, e.g., computing exact frequency moments $F_k\,(k \neq 1)$. This is true for both our model and previous models that bound the total communication but the protocols have to be in the Merlin-Arthur streaming model. Our proof also shows that for many other problems (such as computing the rank), at least $\Omega(\log n/\log\log n)$ rounds are needed for any Merlin-Arthur streaming protocol, or either the total communication or the verifier's space will increase to above polylog(n).

We start by defining what it means for a streaming problem to be as hard as another streaming problem. Let P be a streaming problem, where we assume that inputs are streams of length n, where each symbol is from the same universe U_P. Similarly, denote by Q another streaming problem with input length m and universe U_Q.

Definition 6.4.1.

1. We say that P strictly reduces to Q if there are functions $f_1, \cdots, f_n$ that map U_P to U_Q, $m = n$, and the output of Q on $f_1(x_1), \cdots, f_n(x_n)$ is the same as the output of P on $x_1, \cdots, x_n$.

2. We say P weakly reduces to Q if there are functions $f_1, \cdots, f_n$ where $f_i : U_P \to U_Q^{l_i}$, $\sum_{i=1}^{n} l_i = m$, and the output of Q on $f_1(x_1), \cdots, f_n(x_n)$ is the same as the output of P on $x_1, \cdots, x_n$.

Hence in a weak reduction, one may inflate a single symbol into many. We give a few examples to illustrate the difference between these two reductions. The function we reduce from is the INDEX function.

1. There is a strict reduction from INDEX to $(2-\epsilon)$-approximation of F_∞: map inputs x_j to $2j + x_j$ and then maps the index i to $2i + 1$. Clearly if INDEX$(x, i) = 1$ then F_∞ will be 2 for the image stream, else it will be 1.

2. There is a weak reduction from INDEX to Median: map all x_j to $2j+x_j$ and map i to $n - 2i$ times the number 0 if $i \leq n/2$. Otherwise, if $i > n/2$, map i to $2i - n$ times the number $2n + 2$. Our algorithm for Median implies that there can be no strict reduction from INDEX to Median.

3. Next we show that the problem of deciding whether if a matrix has full rank or not has a weak reduction from INDEX. Consider INDEX, where $x \in \{0,1\}^{\frac{n^2}{4}}$ and $i \in \left[\frac{n^2}{4}\right]$. Define $X \in \mathcal{M}_{n/2}(\mathbf{R})$ whose entries are as follows: $X(r, c) = x_{\frac{n}{2}(r-1)+c}$ for $1 \leq r, c \leq n/2$. Suppose $x_i = X(s, t)$ for some $1 \leq s, t \leq n/2$. Also define $\widetilde{Z} \in \mathcal{M}_{n/2}(\mathbf{R})$ such that $\widetilde{Z}(t, s) = 1$ and all other entries of $\widetilde{Z}$ are zero. Consider the following matrix

$$A = \begin{pmatrix} I_{n/2} & X \\ \widetilde{Z} & I_{n/2} \end{pmatrix}.$$

It is easy to see that if $x_i = 1$, then rank(A) $= n - 1$. Otherwise, if $x_i = 0$, then A is full rank. Hence any streaming algorithm that

decides if $A \in \mathcal{M}_n(\mathbf{R})$ is full rank or not (where the entries of A are streamed row-wise) also decides INDEX on $\Theta(n^2)$ input bits and we get a weak reduction from that problem.

We have the following obvious application of our reductions.

Lemma 6.4.2. *Let P be a streaming problem where the length of the input stream is ℓ_P and each symbol is from the same universe U_P. Similarly, denote by Q another streaming problem with input length ℓ_Q and universe U_Q. Let $n := \max\{\ell_P, \ell_Q, |U_P|, |U_Q|\}$.*

1. *If Problem P can be strictly reduced to problem Q and if every Merlin-Arthur streaming algorithm for P needs either more than* $\mathrm{polylog}(n)$ *space, or more than $t(n)$ rounds of communication between the prover and verifier after the last symbol has been read, or more than* $\mathrm{polylog}(n)$ *communication after the last symbol has been read, then the same is true for Q.*

2. *If P can be weakly reduced to Q and if every Merlin-Arthur streaming algorithm for P needs either more than* $\mathrm{polylog}(n)$ *space, or more than $t(n)$ rounds of communication between the prover and verifier, or more than* $\mathrm{polylog}(n)$ *total communication, then so does Q.*

We can simulate the Merlin-Arthur streaming algorithm by an online Merlin-Arthur communication protocol. The general plan is as follows. A Merlin-Arthur streaming algorithm for INDEX can be simulated by a communication protocol in a pretty straightforward way. Here Alice receives x, Bob receives i, and there are two provers, $Merlin_A$ and $Merlin_B$. $Merlin_A$ only sees x, and can only talk to Alice. $Merlin_B$ sees x, i and will communicate with Bob. Alice and $Merlin_A$ simulate the streaming algorithm

during x, and Bob and $Merlin_B$ take over afterwards. The protocol is one-way, i.e., Alice sends one message to Bob. It becomes clear immediately, that $Merlin_A$ is useless and can be removed, because he does not see anything that Alice doesn't see. Note that it does not matter how much the streaming verifier and prover communicate during x. Hence we arrive at a model where Alice sends one message to Bob and Merlin and Bob exchange messages. As a result of Theorem 3.3.13, it follows that any SIP which solves INDEX in the Merlin-Arthur streaming model needs either more than polylogarithmic space for the verifier, or at least $\Omega(\log n/\log\log n)$ rounds between the prover and verifier after the last symbol has appeared on the stream, or more than polylogarithmic communication after i has appeared on the stream. Since the first and third possibilities are not allowed in our model, we can conclude that there has to be a lot of interaction between prover and verifier in a special verification phase.

For any problem P, given a strict reduction from INDEX on size n strings, if we insist on polylog(n) communication overhead and space, this means that the prover and verifier need a verification phase where they need to interact over $\Omega(\log n/\log\log n)$ rounds after the end of the stream for any Merlin-Arthur SIP. We note that the generic protocol in Theorem 4.1.3, the protocols in [29, 30, 74] are all in the Merlin-Arthur streaming model.

A weak reduction from INDEX suffices to show that for the models that bound the total communication and the verifier's space, many rounds are needed if one resorts to using Merlin-Arthur streaming protocols, but those could potentially take place during the time the stream is observed. Note that this does not contradict the fact that there exist a protocol which can determine the exact median of a stream from a universe of $[n]$ using only three messages and cost $\widetilde{O}(\log n)$ [22]. This is because their 3 message

protocol for Median is in the IP streaming model. We note that weak reductions do not imply anything for the model defined in this chapter though, because we do not restrict the total communication.

We now state the main consequence for streaming algorithms.

Theorem 6.4.3.

1. *In the Merlin-Arthur streaming model, any data streaming algorithm for a problem that has a strict reduction from* Index *either has to use more than polylogarithmic communication overhead, or the verifier's space is more than polylogarithmic, or there are at least* $\Omega(\log n / \log\log n)$ *communication rounds between the prover and verifier after the end of the stream.*

2. *In the Merlin-Arthur streaming model, any data streaming algorithm for a problem that has a weak reduction from* Index *either has to use more than polylogarithmic total communication, or the verifier's space is more than polylogarithmic, or there are at least* $\Omega(\log n / \log\log n)$ *communication rounds between the prover and verifier.*

As an example let us consider the problem of approximating F_∞. It is quite easy to see that the problem F_∞ can be $\log n$-approximated, with the total communication and verifier's space being $\text{polylog}(n)$ in a fully online fashion. To do so, we split the input into $\log n$ phases like the phases in Section 6.3.1, and the prover simply announces the most frequent element of the previous phase in the next phase, where this claim can be checked either via re-streaming, or, more communication efficiently, via the sum check protocol on the arithmetization of the formula performing this check. Clearly, this gives a $(\log n)$-approximation. This idea can be improved as follows: instead of reducing the interval size by half in each phase, we

reduce the interval size exponentially. The same ideas apply, and we get a $(\log^* n)$-approximation, although we need to employ the sum check protocol now.

On the other hand, a $(2-\epsilon)$-approximation is not possible in the Merlin-Arthur streaming model without an extra verification phase due to Theorem 6.4.3, while by the results in [29,30], F_∞ can be computed exactly with $O(\log^2 n)$ rounds of interaction after the stream has ended. Hence a somewhat good approximation can be computed without the extra verification phase, but not an arbitrarily good approximation, while exact computation is possible with such a phase.

Chapter 7

Conclusion and Open Problems

In this thesis, we have closely examined the data streaming model with a prover. The main practical motivation is for outsourcing computations on massive data streams to cloud computing services. But there are many reasons for the client/verifier to be able to check the correctness of the answer provided as well. For example, the service provider can have financial incentives to provide the wrong answer to the the client. Even if the service provider is honest, errors could occur due to a buggy algorithm or the algorithm could have encountered a memory error when reading the massive data which is being streamed at a high speed. We have looked at designing protocols in both the annotation model and the interactive streaming model. The verifier, who is severely space restricted, needs to compute a sketch of the data. This sketch is often the low degree extension of an appropriate polynomial which depends on the stream. The client then uses this sketch of the data to reject wrong claims with high probability.

We have seen that for most interesting functions like both the exact and approximate frequency moments, INDEX etc., we can get at most a

quadratic speedup in the annotation model as compared to the standard model. As in the standard streaming model, the main tool used to show lower bounds for the prover assisted data streaming model is communication complexity. Data streaming protocols with a prover can be simulated by Merlin-Arthur communication protocols, where Merlin is the prover and the data stream input is split across some players, who together constitute the verifier Arthur. We stress again that there is a huge difference in complexity between whether the verifier's message to the prover depends on the input or just consists of random coin tosses.

For the case where the prover and verifier are allowed to interact for polylogarithmic many rounds, we can get exponentially cheaper algorithms than in the annotation model for many problems. Every function in $\mathcal{NC}$ has a streaming interactive protocol whose cost is polylog(m, n) where polylog(m, n) messages need to be exchanged between the prover and the verifier. For functions like frequency moments, we require $O(\log^2 m)$ rounds of interaction which makes these protocols highly impractical. By using the sum check protocol as a tool, we can devise protocols with polylogarithmic cost for F_2, F_0 and many other problems with $O(\log m)$ rounds of interaction.

Using lower bounds from the theory of Merlin-Arthur communication complexity, we have investigated the necessity of the additional verification phase after the stream has ended in the Merlin-Arthur streaming model.

We summarize all the original contributions made in this thesis.

(i) In Corollary 3.3.10, we showed that for $k \geq 3$, it is hard to approximate F_k in the annotation model. Previously it was only known that for $k \geq 6$, it is hard to approximate F_k in the annotation model [21]. For $k \geq 6$, our lower bound in Corollary 3.3.10 is only a polynomial

improvement when compared to the lower bound presented in [21].

(ii) We showed a lower bound for the online Merlin-Arthur communication complexity model with k messages in Theorem 3.3.13. This lower bound can be used to study the theoretical properties of the Merlin-Arthur streaming model. Using our lower bound on the OMA^k communication model together with an OIP^2 protocol for the INDEX function, Chakrabarti et al. [22] showed a exponential separation between OMA^k and OIP^k communication models. This is in contrast to the non-online version of the Merlin-Arthur communication model where it is known that the communication complexity classes IP (Alice and Bob have access to private randomness) and AM (Alice and Bob have access to public randomness) behave similarly to their Turing machine counterparts [82].

(iii) In Section 4.2, we showed that "IP=PSPACE" holds for online communication complexity. We note that Lokam [82] proved that the result "IP=PSPACE" holds for the non-online version of communication complexity. In view of the discussion in (ii), if two non-online communication complexity classes obey a certain relationship, it is not always the case that the same relationship will hold for online communication complexity classes.

(iv) In Chapter 5, we gave a streaming interactive protocol with $\log m$ rounds and polylog(m, n) cost for the exact computation of F_0. The reader is referred to Theorem 5.2.5 for the precise bounds on the help and verification costs. Prior to our work, the best algorithm known to compute the exact value of F_0 using $O(\log m)$ rounds required the space of the verifier to be $O\left(\log m \log n + \log^2 m\right)$ and the total

communication was $O\left(\sqrt{n}\log m\left(\log n+\log m\right)\right)$ [30].

(v) In Chapter 6, we defined a new streaming model that only bounds the communication *overhead*, i.e., the amount of communication sent from the prover to the client per symbol of the data stream. We gave algorithms in our new model for four different streaming problems. They are Median (Theorem 6.3.5), Longest Increasing Subsequence (Theorem 6.3.13), FULL RANK (Theorem 6.3.18) and the perfect matching problem (Theorem 6.3.20). By relaxing the total communication requirement, we managed to find an algorithm for the perfect matching problem for which previously no efficient streaming protocol with a prover was known. All our algorithms have a similar structure with phases whose length shrinks geometrically, and phase i is used to verify certain properties of the stream up to phase $i-1$ using re-streaming of parts of the previous stream. The challenge in each case is to tie the different phases together.

(vi) In previous work [29, 30], it was shown that all problems in the class $\mathcal{NC}$ can be computed in a streaming model with a prover. This general purpose algorithm tends to be inefficient, and this as well as related algorithms based on arithmetization techniques suffer from the following bottleneck: they employ a final verification phase (taking place after the end of the stream), which uses polylogarithmic communication (essentially the only communication in the whole protocol), yet the prover needs to perform computations that take at least linear time during that phase. In Section 6.4, we showed that such a verification phase with a large number of communication rounds between the prover and the verifier is unavoidable for certain problems in the Merlin-Arthur streaming model.

7.1 Open Problems

We list some open problems related to this thesis.

1. We conjecture that the protocol given in Theorem 3.2.2 for the exact computation of F_0 with complexity $O(n^{2/3}\log^{4/3} n)$ in the annotation model is tight, up to polylogarithmic factors. In the online MA model, it might be easier to prove lower bounds larger than $\sqrt{n}$, as compared to proving lower bounds in the general Merlin-Arthur model where such bounds are not known for any explicit function. A similar problem holds for the computation of the exact value of F_∞ in the annotation model.

2. Recall that in the standard streaming model, to obtain a $(1 \pm \epsilon)$-approximation of F_0, we need $O(\epsilon^{-2} + \log m)$ space. For an arbitrarily good approximation, the space complexity is high, as the $1/\epsilon^2$ additive factor can be prohibitively large. One can study the problem of approximating F_0 in the annotation model and see if one can obtain any tradeoffs in the error parameter ϵ.

3. Corollary 3.3.10 shows that approximating F_k in the annotation model is hard for $k \geq 3$. A related problem is to obtain improved tradeoffs on the help and verification cost for approximating F_k in the annotation model. An easier problem to consider is to design a (h, v) protocol in the annotation model which approximates F_k up to constant factors for $k \geq 3$ with $h = v = o(\sqrt{m})$.

4. An interesting theoretical problem would be to obtain non-trivial lower bounds on the Arthur-Merlin (AM) communication complexity of the Disj function or any explicit function. Proving any superlogarithmic lower bounds on the AM complexity of Disj will rule out

constant round streaming interactive protocols for exact $F_k\,(k \neq 1)$ with polylogarithmic complexity. The IP streaming model is the most natural model for delegating computations on data streams. We hope future work will focus on proving lower bounds in this model, or similarly in the $\widetilde{OIP^k}$ communication complexity model (See page 39).

5. We can study the quantum analogue of OIP^k_{cc} communication complexity classes which we denote by $QOIP^k_{cc}$. A natural question to ask is what is the quantum analogue of (3.8)? Aaronson [1] showed that the power of the complexity class quantum online interactive proofs with one message is the same as the quantum one-way communication class. He developed an amplification procedure to boost the error probability without repeating the proof. To obtain a quantum one-way protocol, he showed that it is enough to loop over all possible classical messages from Merlin. It is well know that the quantum one-way communication complexity of the INDEX function is $\Omega(n)$ [87]. Since INDEX $\in OIP^2_{cc}$, we have $QOIP^1_{cc} \subsetneq QOIP^2_{cc}$. For $QOIP^2_{cc}$, we conjecture that this communication complexity class is the same as the complexity class $Q^{(2,B)}_{cc}$, where $Q^{(2,B)}$ denotes the quantum communication complexity where 2 messages are exchanged between Alice and Bob with Bob sending the first message. One can use the parallelization of Kitaev and Watrous [69] to show that quantum OIP communication classes with 3 messages are enough to simulate any quantum OIP communication class with more than 3 messages.

6. Can we obtain an interactive protocol for the exact computation of F_∞ with $\log m$ rounds and polylog(m, n) space and communication? Since giving a constant approximation of F_∞ is hard in the standard streaming model, the problem of approximating F_∞ with $\log m$

rounds and polylog(m, n) complexity is interesting as well.

7. The fully online algorithm for the new streaming model which bounds the communication overhead (See Chapter 6) needs to know the length of the stream in advance. An interesting improvement would be to modify our protocols to work with unknown stream lengths while maintaining the structure of our algorithm.

8. For the FULL RANK problem in Subsection 6.3.3, our protocol requires the matrix to be streamed in row major or column major order. An interesting question is whether our protocol can be modified such that the matrix entries can appear in any order on the stream? Such an improvement would lead to an algorithm that decides if an input graph has a perfect matching, where the edges of the graph can be presented in any order. Currently for our algorithm, to decide if a graph has a perfect matching, all the edges of one vertex have to be presented first before proceeding to the next.

Bibliography

[1] Scott Aaronson. QMA/qpoly⊆ PSPACE/poly: De-Merlinizing quantum protocols. In *IEEE Conference on Computational Complexity*, pages 261–273, 2006. 43, 44, 140

[2] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory*, 1(1), 2009. 27, 33

[3] Farid Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 157(2):139–159, May 1996. 14, 15, 26

[4] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In John H. Reif, editor, *STOC*, pages 370–379. ACM, 2002. 20

[5] D. Aldous and P. Diaconis. Longest increasing subsequences: From patience sorting to the Baik-Deift Johansson theorem. *Bulleting of the American Mathematical Society*, 36:413–432, 1999. 100, 101, 102

[6] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. Earlier version in STOC'96. 2, 8, 16, 18, 19, 45, 49

[7] S. Arora and B. Barak. *Complexity Theory: A Modern Approach*. Cambridge University Press, 2009. 23, 56, 63, 67

[8] L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity theory. In *Proceedings of 27th IEEE FOCS*, pages 337–347, 1986. 4, 16, 32, 34, 42, 55, 58, 65

[9] László Babai and Shlomo Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988. 4, 50, 55, 60

[10] László Babai. Trading group theory for randomness. In Robert Sedgewick, editor, *STOC*, pages 421–429. ACM, 1985. 4, 34

[11] Eric Bach and Jeffrey Shallit. *Algorithmic number theory, volume 1: efficient algorithms*. MIT Press, Cambridge, Massachusetts, 1996. URL: `http://www.math.uwaterloo.ca/ shallit/ant.html`. 86, 87

[12] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *Proceedings of 43rd IEEE FOCS*, pages 209–218, 2002. 19, 49

[13] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In José D. P. Rolim and Salil P. Vadhan, editors, *RANDOM*, volume 2483 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002. 18

[14] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in

graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 623–632, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. 2, 18

[15] Kevin S. Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. On synopses for distinct-value estimation under multiset operations. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *SIGMOD Conference*, pages 199–210. ACM, 2007. 18

[16] Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *SODA*, pages 708–713. ACM Press, 2006. 19

[17] Joshua Brody and Amit Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *IEEE Conference on Computational Complexity*, pages 358–368. IEEE Computer Society, 2009. 18

[18] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *Proceedings of 18th IEEE Conference on Computational Complexity*, pages 107–117, 2003. 2, 19, 47, 48, 49

[19] Amit Chakrabarti. A note on randomized streaming space bounds for the longest increasing subsequence problem. *Information Processing Letters*, 112(7):261–263, 2012. 101

[20] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In *SODA*, pages 687–706, 2014. vii, 2, 22, 27, 59, 89

[21] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Algorithms*, 11(1):7:1–7:30, August 2014. A preliminary version of this paper by A. Chakrabarti, G. Cormode and A. McGregor appeared in ICALP 2009. vii, 1, 2, 3, 4, 5, 22, 27, 30, 31, 43, 45, 46, 50, 59, 88, 89, 136, 137

[22] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and Arthur-Merlin communication. In David Zuckerman, editor, *Conference on Computational Complexity*, volume 33 of *LIPIcs*, pages 217–243. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. 2, 4, 39, 55, 56, 58, 60, 132, 137

[23] Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31, 2013. 119

[24] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 205–214, New York, NY, USA, 2009. ACM. 20, 113

[25] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997. 18

[26] Don Coppersmith and Ravi Kumar. An improved data stream algorithm for frequency moments. In J. Ian Munro, editor, *SODA*, pages 151–156. SIAM, 2004. 19

[27] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. 119

[28] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I*, ESA'10, pages 231–242, Berlin, Heidelberg, 2010. Springer-Verlag. vii, 2, 22, 27, 59, 89

[29] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 90–112, New York, NY, USA, 2012. ACM. 5, 6, 62, 64, 70, 75, 88, 91, 128, 132, 134, 138

[30] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.*, 5(1):25–36, 2011. vi, vii, 2, 4, 5, 6, 62, 70, 71, 73, 74, 75, 88, 89, 91, 96, 128, 132, 134, 138

[31] P. D. Domich, R. Kannan, and L. E. Trotter, Jr. Hermite normal form computation using modulo determinant arithmetic. *Math. Oper. Res.*, 12(1):50–59, February 1987. 116

[32] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In Giuseppe Di Battista and Uri Zwick, editors, *ESA*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2003. 18

[33] Pavol Duris, Juraj Hromkovic, Jos'e D. P. Rolim, and Georg Schnitger. On the power of Las Vegas for one-way communication complex-

ity, finite automata, and polynomial-time computations. *Information and Computation*, 169:284–296, 1997. 14

[34] Funda Ergün and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *SODA*, pages 730–736, 2008. 100, 101

[35] Cristian Estan, George Varghese, and Michael E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006. 18

[36] Xin Gui Fang and George Havas. On the worst-case complexity of integer Gaussian elimination. In Bruce W. Char, Paul S. Wang, and Wolfgang Küchlin, editors, *ISSAC*, pages 28–31. ACM, 1997. 113

[37] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang 0004. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008. 19

[38] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005. 2, 20

[39] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. *HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm.*, pages 127–146, electronic only. Nancy: The Association. Discrete Mathematics & Theoretical Computer Science (DMTCS), 2007. 18

[40] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *FOCS*, pages 76–82. IEEE Computer Society, 1983. 18

[41] John Foley. As big data explodes, are you ready for yottabytes? http://www.forbes.com/sites/oracle/2013/06/21/as-big-data-explodes-are-you-ready-for-yottabytes/, June 2013. 70

[42] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP Congress*, pages 839–842, 1977. 114

[43] Bin Fu. Derandomizing polynomial identity over finite fields implies super-polynomial circuit lower bounds for NEXP. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:157, 2013. 62

[44] Anna Gal and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *FOCS*, pages 297–304, 2007. 101

[45] Sumit Ganguly. Estimating frequency moments of data streams using random linear combinations. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *APPROX-RANDOM*, volume 3122 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 2004. 19

[46] Sumit Ganguly. A hybrid technique for estimating frequency moments over data streams. Unpublished Manuscript, 2004. 19

[47] Phillip B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *VLDB*, pages 541–550. Morgan Kaufmann, 2001. 18

[48] Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *SPAA*, pages 281–291, 2001. 18

[49] Oded Goldreich. *Computational complexity - A conceptual perspective.* Cambridge University Press, 2008. 61, 62

[50] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989. 36

[51] S Goldwasser and M Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 59–68, New York, NY, USA, 1986. ACM. 55, 60, 64

[52] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 113–122, New York, NY, USA, 2008. ACM. vii, 5, 6, 62

[53] I. J. Good. Surprise indexes and p-values. *Journal of Statistical Computation and Simulation*, 32(1-2):90–92, 1989. 16

[54] P. Gopalan, T.S. Jayram, R. Krauthgamer, and R. Kumar. Estimating the sortedness of a data stream. In *SODA*, pages 318–327, 2007. 20, 100, 101, 108, 110, 112

[55] Andre Gronemeier. Asymptotically optimal lower bounds on the NIH-multi-party information complexity of the and-function and disjointness. In Susanne Albers and Jean-Yves Marion, editors, *STACS*,

volume 3 of *LIPIcs*, pages 505–516. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. 47

[56] Tom Gur and Ran Raz. Arthur-Merlin streaming complexity. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part I*, ICALP'13, pages 528–539, Berlin, Heidelberg, 2013. Springer-Verlag. vii, 2, 27, 76, 77, 79, 88, 89

[57] George Havas, Derek F. Holt, and Sarah Rees. Recognizing badly presented z-modules. *Linear Algebra and its Applications*, 192:137 – 163, 1993. 113

[58] Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms*, pages 107–118. American Mathematical Society, Boston, MA, USA, 1999. 20

[59] Roger A. Horn and Charles R. Johnson, editors. *Matrix Analysis*. Cambridge University Press, New York, NY, USA, 1986. 119

[60] Juraj Hromkovic. *Communication complexity and parallel computing*. Texts in theoretical computer science. Springer, 1997. 14

[61] Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *FOCS*, pages 283–288. IEEE Computer Society, 2003. 18

[62] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 202–208. ACM, 2005. 19

[63] T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008. 15

[64] Stasys Jukna. *Extremal Combinatorics with Applications in Computer Science*. Springer, Heidelberg, 2011. 102

[65] J. Justesen. A class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18:652–656, 1972, 1972. 79, 158

[66] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. 5(4):545–557, 1992. Earlier version in Structures'87. 16

[67] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. ACM. 16, 18

[68] Tracy Kimbrel and Rakesh Kumar Sinha. A probabilistic algorithm for verifying matrix products using o(n^2) time and $\log_2$n+o(1) random bits. *Inf. Process. Lett.*, 45(2):107–110, February 1993. 114

[69] A. Kitaev and J. Watrous. Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In *Proceedings of 32nd ACM STOC*, pages 608–617, 2000. 140

[70] Hartmut Klauck. On quantum and probabilistic communication: Las Vegas and one-way protocols. In *Proceedings of the Thirty-second*

Annual ACM Symposium on Theory of Computing, STOC '00, pages 644–651, New York, NY, USA, 2000. ACM. 14

[71] Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *IEEE Conference on Computational Complexity*, pages 118–134. IEEE Computer Society, 2003. 32, 33

[72] Hartmut Klauck. On Arthur Merlin games in communication complexity. In *IEEE Conference on Computational Complexity*, pages 189–199. IEEE Computer Society, 2011. 32

[73] Hartmut Klauck and Ved Prakash. Streaming computations with a loquacious prover. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 305–320, New York, NY, USA, 2013. ACM. 1, 52

[74] Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 919–930, 2014. 1, 2, 89, 91, 128, 132

[75] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999. Earlier version in STOC'95. Correction at `http://www.eng.tau.ac.il/~danar/Public/KNR-fix.ps`. 14, 15, 26

[76] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 11 2006. 12, 14

[77] Oded Lachish, Ilan Newman, and Asaf Shapira. Space complexity vs. query complexity. *Computational Complexity*, 17(1):70–93, 2008. 158

[78] Serge Lang. *Algebra (3. ed.)*. Addison-Wesley, 1993. 115

[79] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM. 119

[80] Troy Lee and Adi Shraibman. *Lower Bounds in Communication Complexity*. Now Publishers Inc., Hanover, MA, USA, 2009. 14

[81] David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. *J. Comb. Optim.*, 11(2):155–175, 2006. 20

[82] S. Lokam. Spectral methods for matrix rigidity with applications to size-depth trade-offs and communication complexity. *Journal of Computer and System Sciences*, 63(3):449–473, 2001. Earlier version in FOCS'95. 55, 60, 64, 69, 137

[83] S. Lokam. Complexity lower bounds using linear algebra. *Foundations and Trends in Theoretical Computer Science*, 4(1–2):1–155, 2008. 14

[84] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992. 63, 128

[85] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995. 60

[86] M. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers, 2005. 11

[87] A. Nayak. Optimal lower bounds for quantum automata and random access codes. In *Proceedings of 40th IEEE FOCS*, pages 369–376, 1999. quant-ph/9904093. 15, 26, 140

[88] I. Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2):67–71, 1991. 14

[89] I. Newman and M. Szegedy. Public vs. private coin flips in one round communication games. In *Proceedings of 28th ACM STOC*, pages 561–570, 1996. 14

[90] C. H. Papadimitriou and M. Sipser. Communication complexity. *Journal of Computer and System Sciences*, 28(2):260–269, 1984. Earlier version in STOC'82. 14

[91] Ved Prakash. Efficient delegation protocols for data streams. In *Proceedings of the 2014 SIGMOD PhD Symposium*, SIGMOD'14 PhD Symposium, pages 6–10, New York, NY, USA, 2014. ACM. 1

[92] Ran Raz. Quantum information and the PCP theorem. *Algorithmica*, 55(3):462–489, 2009. 56

[93] A. Razborov. Lower bounds on the size of bounded-depth networks over a complete basis with logical addition (russian). *Matematicheskie Zametki*, 41(4):333–338, 1987. 77, 79

[94] A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992. 16

[95] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980. 157

[96] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, October 1992. 64, 65

[97] A. Shen. IP = PSPACE: Simplified proof. *J. ACM*, 39(4):878–880, October 1992. 64, 65, 67

[98] Charles C. Sims. *Computation with Finitely Presented Groups (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, 2010. 116

[99] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of 19th ACM STOC*, pages 77–82, 1987. 77, 79

[100] Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 336–345. SIAM, 2007. 20

[101] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013. 2, 63, 64

[102] Justin Thaler. Semi-streaming algorithms for annotated graph streams. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:90, 2014. vii, 2, 22, 60, 89

[103] W. T. Tutte. The factorization of linear graphs. *J. Lond. Math. Soc.*, 22:107–110, 1947. 126

[104] J.H. van Lint. *Introduction to Coding Theory (Graduate Texts in Mathematics)*. Springer, 3rd rev. and exp. ed. edition, 12 1998. 118, 157

[105] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM. 119

[106] David P. Woodruff. Optimal space lower bounds for all frequency moments. In J. Ian Munro, editor, *SODA*, pages 167–175. SIAM, 2004. 18

[107] A. C-C. Yao. Some complexity questions related to distributive computing. In *Proceedings of 11th ACM STOC*, pages 209–213, 1979. 11, 14

[108] Chang-Wu Yu and Gen-Huey Chen. Parallel algorithms for permutation graphs. *BIT*, 33(3):413–419, 1993. 100

[109] Jian Zhang. A survey on streaming algorithms for massive graphs. In Charu C. Aggarwal and Haixun Wang, editors, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 393–420. Springer, 2010. 20

Appendix

A.1 Schwartz-Zippel Lemma

Lemma A.1.1. *[95]*

Let $Q(x_1, \cdots, x_n) \in \mathbf{F}[x_1, \cdots, x_n]$ be a multivariate polynomial of total degree d. Fix any finite subset $S \subseteq \mathbf{F}$ and let $r_1, \cdots, r_n$ be chosen independently and uniformly at random from S. Then

$$\mathbf{Pr}\left(Q(r_1, \cdots, r_n) = 0 \,|\, Q(x_1, \cdots, x_n) \not\equiv 0\right) \leq \frac{d}{|S|}.$$

A.2 Coding Theory

We give a brief background of coding theory which we will use in this thesis. For more details of standard definitions, the reader is referred to [104]. A q-ary linear code $\mathcal{C}$ of length n is a linear subspace of $\mathbf{F}_q^n$, where q is some prime power. If $\mathcal{C}$ has dimension k, then we call it a $[n, k]_q$ code. The (Hamming) distance between two codewords $x, y \in \mathcal{C}$, denoted by $d(x, y)$ is the number of indices $i \in [n]$ such that $x_i \neq y_i$. The distance of the code is defined as the minimum distance over all pairs of distinct codewords in $\mathcal{C}$. If the minimum distance of the code is d, we denote it by $[n, k, d]_q$. The generator matrix $G \in \mathcal{M}_{n,k}(\mathbf{F}_q)$ of the code is a $n \times k$ matrix where the column span of G gives $\mathcal{C}$. The relative distance of the code is d/n and

the rate of the code is k/n. A linear code is called a good code if both its relative distance and rate is at most some constant. The Reed-Solomon code is an example of a good code with alphabet size $q = n + 1$. But in our case, we need the alphabet size to be much smaller than n. Justesen codes [65] are a class of good codes with a constant alphabet size. We say that a linear code is locally logspace constructible if the (i, j)-entry of the generator matrix G can be constructed using space $O(\log n)$. It is known that Justesen codes are locally logspace constructible (see Lemma 3.3 of [77]).

A.3 Interactive Proof Systems

The notion of interactive proof systems can be seen as a probabilistic analogue of the complexity class $\mathcal{NP}$.

Definition A.3.1. (**Turing Machine Interactive Proof (IP) model**) For any integer $k \geq 1$ which may depend on the input length, we say a language L is in IP^k_{TM} if there is a probabilistic polynomial-time Turing machine V that interacts with a prover P such that P and V exchange k messages in total with P sending the last message. Their interaction should satisfy the following conditions:

- If $x \in L$, then the probability that the verifier accepts is greater than $2/3$.
- If $x \notin L$, then even against an optimal prover, the probability that the verifier accepts is less than $1/3$.

Note that all the probabilities are over the random choices of V. The prover messages do not depend upon the random strings of V, but only on

the messages or questions that the verifier sends. We say that the random strings of V is private.

We define $\mathcal{IP}_{\mathrm{TM}} := \bigcup_{k=poly(n)} IP^k_{TM}$. We say L has an interactive proof system if $L \in \mathcal{IP}_{\mathrm{TM}}$.

The Turing machine IP model is sometimes called the private coin interactive proof model. We can also consider the public coin interactive proof model (Arthur-Merlin model) where all the questions of V to the prover are obtained by tossing coins and revealing them to the verifier.

Definition A.3.2. For every k, the complexity class AM^k_{TM} is a subset of IP^k_{TM} obtained when we restrict the messages from V to the prover to be random bits only. V is not allowed to use any other random bits that are not contained in these messages. We define $\mathcal{AM}_{\mathrm{TM}} := AM^2_{TM}$.